User-configurability in classroom mathematics software

User-configurability in classroom mathematics software

by

Mark Twells M.A.

A Master's project submitted in partial fulfilment of the requirements for the award of the degree of MSc in Computer Education of Loughborough University of Technology.

January 1990

Supervisor : D R Green. MSc MEd PhD

Section	Page	Contents
1	2	**Configurability & educational software : the need**
2	9	Outline of the software and its function
2.1	9	General description
2.2	10	Interactive Help
2.3	12	Screen photographs of FLOW
3	36	**A comparison of similar software**
4	40	**System chosen for the project software**
5	43	**Method of use**
6	44	**USER DOCUMENTATION**
6.1	44	Terminology
6.2	45	Starting the package
6.3	46	The Control Panel
6.4	48	A Quick Guided Tour
6.4.1	48	Tutorial (1) : a simple flowchart
6.4.2	49	Tutorial (2) :Loading and running a flowchart
6.4.3	51	Tutorial (3) : Graphing a flowchart
6.4.4	52	Tutorial (4) : Editing a flowchart
6.4.5	53	Tutorial (5) : Named variables
6.4.6	54	Tutorial (6) : Configuring FLOW
6.4.7	56	Tutorial (7) :Editing and adding to the HELP facilities
6.4.8	57	Adding and editing : general points
6.4.9	61	Entering boxes
6.4.10	64	Running a flowchart : General points
6.5	69	A more detailed examination of FLOW
6.5.1	69	The EDIT MENU
6.5.2	74	Flowchart functions provided
6.5.3	77	Working with named variables

6.5.4	78	Accessing named variables
6.5.5	79	The variables window
6.5.6	80	Memory operations
6.5.7	81	The F <n> operation
6.5.8	81	Graphing flowchart input and output
6.5.9	83	Configuring FLOW
6.5.10	89	Editing the HELP files
6.6	90	Error reporting
6.6.1	90	Syntax Errors
6.6.2	91	Run Time Errors
6.6.3	91	Operating System Errors
6.6.4	91	Error messages
7	**95**	**Educational Approach**
7.1	96	Investigation 1 : Chaos
7.2	97	Newton Raphson
7.3	100	Investigation : Newton Raphson & Square Roots
7.4	103	Investigation : Box Volumes
7.5	104	An investigation with 11 year olds
7.5.1	105	Investigation 4 : sequences and series
7.6	106	Investigation : Formula construction
8	**107**	**References**
9	**108**	**Listings**
9.1	108	Preliminary notes
9.2	109	Software used in program construction
9.3	109	Program structure
9.3.1	109	Structural analysis of FLOW.main modules
9.3.2	121	Index to Structural analysis
9.4	124	Program listings
9.4.1	124	H.FLOW

9.4.2	137	H.HOURGLASS
9.4.3	138	H.WINDASM
9.4.4	139	H.WINDS
9.4.5	143	C.F_MAIN
9.4.6	174	C.F_OPS
9.4.7	208	C.F_PLOT
9.4.8	214	C.F_HELP
9.4.9	220	C.F_SETUP
9.4.10	238	C.W_SHELL
9.4.11	256	ASM.F_ASM
9.4.12	259	ASM.HOURGLASS
9.4.13	260	Index to C routines in listings

Abstract

The rapid pace of hardware development has presented education with a new generation of hardware having vastly improved capabilities. There is now room to fit general-purpose programs inside these machines which are capable of operating in more than one knowledge domain, with more than one style or at more than one level. In addition, it is now possible to afford programming time to insulate the user well from the intricacies of running the machine and to provide help for him in a reasonably complex way.

Controlling these packages, configuring them and teaching using them is far more complex than using educational software which operates in only one knowledge domain or at one level. The question "who should control the software" is the subject of some academic debate at present, and some methods are advanced in this project.

This project describes FLOW, a software package which has potentially wide application in the sphere of mathematics, from 11 years old to A level. FLOW is intended to provide a useful classroom tool, and as a testbed of methods of allowing the teacher to configure the software to allow precisely the functions he or she wants available to the student.

FLOW is intended as a general purpose package to allow users to :

1) Generate FLOWchart models of functions of one variable and relate these models to key presses on a calculator.

2) Use the "programming language" of FLOW to investigate convergence and iterative methods.

Acknowledgements

My Colleagues Richard Davies and Ian Flynn for their Help in evaluating FLOW.

My wife Sarah for her patience in moments of trauma.

The various members of Loughborough Grammar School who experimented with FLOW and with the testbed worksheets.

Declaration

This project, report and software is entirely the work of the author.

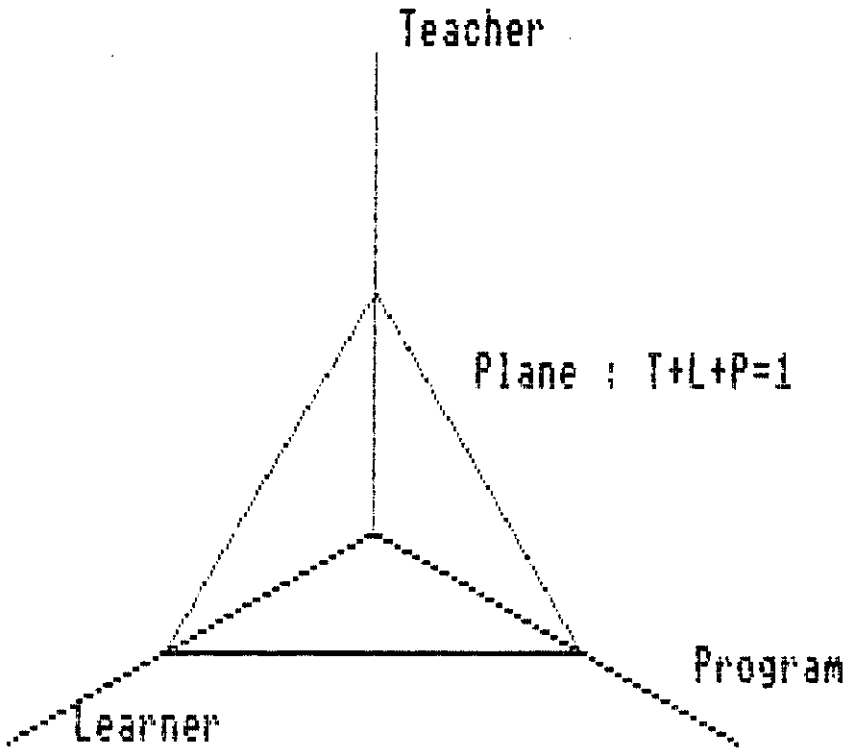
The !EDIT application, supplied on the disk in order to allow examiner configuration of FLOW, and !SYSTEM, supplied in order to allow programs using central resources to run are both (C) Acorn Computers and do not form part of the software constructed for this project. Both are supplied "bundled" with RISC OS, the current operating system.

1 Configurability and educational software : the need

It has been recognised for some time that not all learners achieve their aim by working in the same way. The proponents of the ideas generally grouped together as *matching theories* try to identify the characteristics of the learner in order to identify the required treatment. The ideas are grounded in psychology. For instance, Lewin's rather mathematical treatment of this area states that the behaviour of a person is a function of both *the person himself* and *his environment*. As software authors, we have consistently failed to address either of these two variables, producing software which is not adaptable in any way by any one to different learning strategies. In fact, most existing software assumes that the behaviour is a function of **nothing** : a constant function! It is surprising to find evidence of this in the commercial software scene, but it is there. Take for instance, Lansdell (1988) who, in a review of a prototyping tool for the Macintosh series of machines, states "*..have someone responsible for the user interface design to decide if the program interacts with the user in* ***the*** *most effective way*", (my emboldening on "the"). Surprising on two counts : firstly, he assumes that the programmer or program team has the sole responsibility for the user interface, and secondly, he seems to suppose that there is only one way which is "the most effective" for all users. Put this way, that we ever expect educational software to succeed in its aims is rather surprising! Shackel(1981) sums this up neatly :"*Computer designers are primarily and quite rightly concerned to improve the performance of the computer hardware and software; they often forget what matters most is efficiency and performance of the total man-computer system*".

If one examines the theory behind matching in more depth one finds that the experimental investigations are at a rather crude level. Generally, only one (necessarily psychometric) quantity is measured, and is plotted against "treatment". Cronbach & Snow (1977), for instance, used IQ scores looking for disordinal responses to treatments (that is, significantly different responses to differing teaching treatments). They did not find differing responses, and concluded that IQ was an inadequate psychometric measurement for this purpose. There are a number of other instances of research too numerous to consider here. My point is that although nearly all of this research points to the conclusion that the learning outcome and efficiency is a function of both the person and his learning environment, research is not at a stage where it will provide a constructive method of tailoring a piece of software. We must look elsewhere, bearing in mind that the motivation is provided by the work on matching.

Chandler(1984) and others describe what they call the "locus of control" of a program. The locus stretches from one extreme (the programmed instruction learning machines of the behaviourists) to the other (the investigative environments, like LOGO). Are all learners optimising their learning rate on the same software presented in the same way? It seems highly unlikely in the light of the matching results. In addition, the level of student control is not the only factor in the game. We ought to consider at least *teacher control* and *program control*, which may be seen as sitting at the three corners of a triangle, a sort of three-cornered joystick where currently the ***programmer*** decides the placement.



Who should decide the appearance of the program ? J.R. Hartley (1981) for instance, says "*Clearly there are difficulties in allowing student control when the learner has an inaccurate view of his own competence and processing style*". Recent researches into learning style have suggested that a level of matching is important, and thus a program with one fixed structure will not be ideal for all students. Watson(1987)[1] makes another equally important point : "*The task of designing an appropriate screen and associated user interface is made harder by the fact that the end-users, the teachers and pupils, are composed of a combination of individuals who will have their own idiosyncrasies and preferences of how to view a program.*" Thus some means of changing the style or presentation of the program is desirable from both a practical

[1]Developing CAL : Computers in The Curriculum, page 120, section entitled "The User in Control".

point of view : we are all idiosyncratic, and also from a point of view of matching program style to student in the sense of trying to optimise learning rate.

If we are suggesting we ought to structure the program to optimise the learning rate, we need to be aware of a point that systems psychologists make quite forcibly. Take, for instance De Greene (1970): "*Optimum design of the man-machine package alone does not guarantee the effectiveness we desire... and our success...may introduce problems that seem less easy to handle*". Perhaps we ought to consider the methods of Systems Psychology more when constructing large educational programs which are intended to "work" across a broad spectrum of pupils and teacher styles. The current generation of educational software seems to have paid scant regard to these ideas. Prof. Edmonds' plea[2] to consider Human Factors **early** in the design cycle is worth remembering..but balanced with his warning that as yet we know little about the human components in a system. He states three points made by Shackel, which, he considers, should be embodied in any program design stage, one of which is especially relevant in educational computing and in particular here : Shackel's first precept is that the design should be "User centred". Flexibility should be ***built in*** and should in fact be the ***central concern***, rather than the fundamental algorithms of the program.

Bearing in mind Hartley and other's work mentioned earlier, the level of control and configurability available to the user should depend to some extent on decisions made by the teacher. Most

---

[2] Talk at Loughborough University of Technology, November 1988, entitled "The Human-Computer Interface"

teachers would probably agree that the user control provided by a WIMP environment is fine, and likely to be well-used, **provided** one is prepared to spend time teaching the user how to use WIMPs! In order to be able to drive the package after a short period of time the student must be presented with an environment which is as familiar as possible to him. This means that the program must use terminology and graphics which are drawn from the particular course which the student is following, and which conform to the choice of words as used by the teacher. Few, if any, packages have allowed this level of configurability (low though it is) in the past, and those that do have also allowed the student (or user) perhaps too much freedom to experiment.

We are left, then, with a dilemma. Too much student control of a program is undesirable, but there is a need to enable a substantial level of configurability by the teacher. The level of control by the program is likely to be low in a program which sells. Proper program control suggests techniques of artificial intelligence, and this is unlikely to appear in commercial educational software whilst the techniques involved are still essentially experimental. Although "expert systems" are used in business the topics to which they are devoted are rather less abstract than most topics in mathematics (not intending to place the subject on a pedestal : it ***is*** abstract). In fact, reliable tools for representing the interrelationships between knowable facts must be a precursor to any such control by a program. Research into such methods is occurring (e.g. Pask & Pangaro 1982), but is not yet widely used. Most teachers I have spoken too would be unhappy with software which made strategic decisions behind their backs using methods of which they were unaware. Practically then, we must

look at methods of allowing teacher control. Teacher control has a number of advantages built in to it. The teacher presumably knows both the pupils and the course well. He is in the privileged position of being able to identify pupil requirements and being able to adjust the program to match pupil and course, even if the formal methods for such adjustments are the subject of hot academic debate. Such adjustments may range from the relatively trivial (notation used, words used for key concepts, colours, speed, difficulty, adjustments to printer type, and so on) to fairly major configurations of the user interface such as the graphical style of the program, terminology used and functions provided by the program which are visible to the user. Of course, if he is to make these adjustments successfully then (s)he must be familiar with the software and with the methods used to configure it.

FLOW, the subject of this dissertation, is intended primarily as a testbed of mechanisms for setting up teacher control. The actual features controlled are not of major significance, although they have been chosen from features which my immediate colleagues in the subject have found irritating about packages in informal discussions with one another. Notation is a good example : does the course use exp(x) or e^x for instance? Does sinh(x) appear in the software before it has been introduced in the course? Does the program round its answers without indicating that this has been done? Does it provide 10 decimal places when the teacher always gets worked up about unrounded answers? These are all trivial points, but they do cause endless frustration with practising teachers. The programmer is not in a position to make value judgements about which approach is right, and should therefore permit the teacher to make

the decision, on an individual basis for each pupil, if necessary.

The next problem (and this is not tackled here) is one of teacher naivety. Even in 1989 there are a number of teachers who are positively frightened by computers. They may be sufficiently confident to use them in the classroom and use them effectively, but cannot or will not delve below the application level. How do these people "configure" software for other users? This area needs investigation. There are at least two potential pitfalls :

Firstly, the teacher is unlikely to be a programmer. He may have some experience in using computers, but to talk of "setting environment variables", or of "configuring the .SYS file" is likely to put most teachers off. The configurability must be easy. The obvious solution is to set the program up through a "front end" which eliminates the need for the teacher to get his or her hands "dirty".

Secondly, the teacher is unlikely to have much time available. Again, we need a quick method of configuring the software.

My experience suggests that even the use of a friendly text editor is unlikely to be taken up since the syntax imposed by such a system would be rigid : the equivalent of an old "command-line" interface. A tentative suggestion seems to be some sort of natural language interaction with the machine in which the teacher is grilled about the nature of the configuration required. Here, I document a package which has the *potential* for reconfiguration by the teacher, but the method used in this software will consist of environment variable settings, miscellaneous text files and so on.

This is not the end of the story!

2 Outline of the software and its function

2.1 General description

FLOW is intended to provide the teacher and pupil with an environment in which experimentation with simple functions of a single variable is possible using a notation which is familiar to the pupil and which is used within the pupil's GCSE or A level mathematics course. FLOW is intended to be easy to use and quick to learn. It is intended to be used without extensive time spent in studying a manual on how to run the program and on how the program works. Given limited time availability in the current school timetable, extensive instruction in the use of *one* program nowadays is simply not possible. Any use of technology must have as one of its spin-offs the high probability of saving time. With this in mind the old non-WIMP systems (such as the BBC B or the RML series of machines) do not provide an alluring environment in which to develop software for educational use within the secondary school. Accordingly, the requirement was for software which would run rapidly, reliably and with a minimum of time spent learning the package, and which conforms as closely as possible to a standard presentation. The notation used needs to be as familiar as possible, and the operations required to run the package need to be as intuitive as possible. The WIMP environment provides a modern interactive environment in which users feel familiar and find easy to use. Although criticised for being difficult to learn, the environment is almost standard across differing machines and may be regarded as a standard interface to which educational software should conform.

I have not used symbolic icons within the program, but have replaced them with textual ones. In my experience, students of this age range do not find symbolic icons particularly meaningful. Textual icons provide a sensible halfway house, allowing the use of the WIMP environment without obliging students to learn what the icons stand for.

FLOW allows the user to build flowcharts by tapping keys on a simulated on-screen calculator. Once flowcharts have been completed they can then be run with numbers input to investigate their effect, convergence, and other properties which the teacher may be interested in. A number of facilities are provided to help the user. It is possible to obtain graphical output easily, and a record of runs made is kept without any action on the user's part. On-line help has been seen as essential for this package, and provision has been made for it to be teacher configurable.

2.2 Interactive HELP

The problem with on-line help in educational software has always been that it could only be present in small amounts because:

(1) Calling it up would destroy amounts of the screen which the user required.

(2) Screen "bandwidth" has been limited, and much textual abbreviation has taken place when placing the help within the program structure.

(3) It has not been possible to include graphics within the help system.

(4) A "flat" system of filing the help is inadequate. It does not seem to correspond to the structures we have in our

minds of the subject so far.

The WIMP environment gets round the first of these problems, and allows the programmer to pop up items on the screen without destroying the underlying screen contents. 80 column screens have made an improvement in the screen bandwidth. Tree-structured directories on both Acorn and Apple WIMP systems allows a structured filing approach to "help". This is used in this software.

The "help" facility should be regarded as an integral part of the program and also as configurable by the teacher. FLOW's help facility is both integral and configurable. Under FLOW's help it is possible to call up help on any topic at any stage without losing either details of runs or of flowcharts. The help window will disappear when closed by the user, leaving all beneath intact. In addition, the help topics are stored and presented hierarchically. The menu structure within help reflects the layout on the disk. Thus the top level of help consists of further options (usually general ones, like "windows", "flowcharts", and so on) which if selected by moving the mouse rightwards over the arrow will reveal a further level, and so on until a "leaf" topic is selected. The leaves on this particular tree correspond to text files in the help area on disk. The non-leaves correspond to directories.

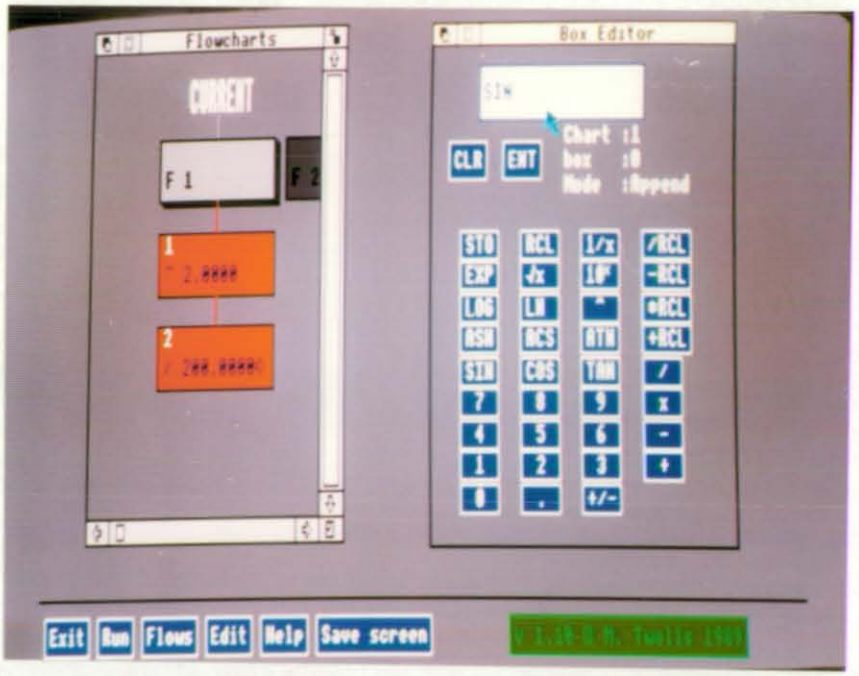
Help files themselves are straight text files though it is possible to incorporate graphics in them (in a number of different ways) because of the way the Acorn "terminal" works. Thus the teacher can juggle with the help files, rearrange them, create her own and so on. No programming experience is needed, merely the ability to use an editor. This I see as vital. Teachers differ in their opinion on the presentation and language in these packages.

The solution is not to argue about it, but to allow individuals to change the package to suit their needs. It is possible to create multiple different help facilities with the package to use with different ability ranges or years within a school. The user would not even be aware of the ability to change the help setup.

The files are placed in directories which are rooted in a designated "help" directory, whose name can be defined in the configuration of the program. The teacher, if he or she is familiar with the operation of a word processor or text editor, can completely redesign the help files supplied with the system, and can rearrange them too. The structure is copied into the program as a treed menu structure, the user merely flips through until he finds the topic required (on a leaf of the help tree), and selects by pressing ***SELECT*** on that item. The help window opens, displays the text, and is closed in the usual way by clicking on the cross.

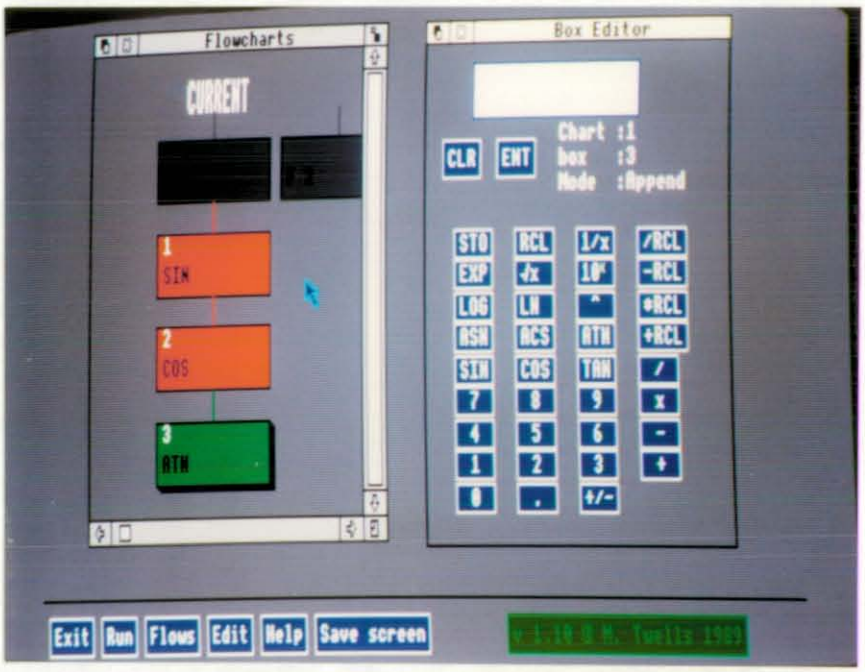
2.3 Screen photographs of FLOW

A short section intended to demonstrate the appearance of some aspects of FLOW whilst it is running.



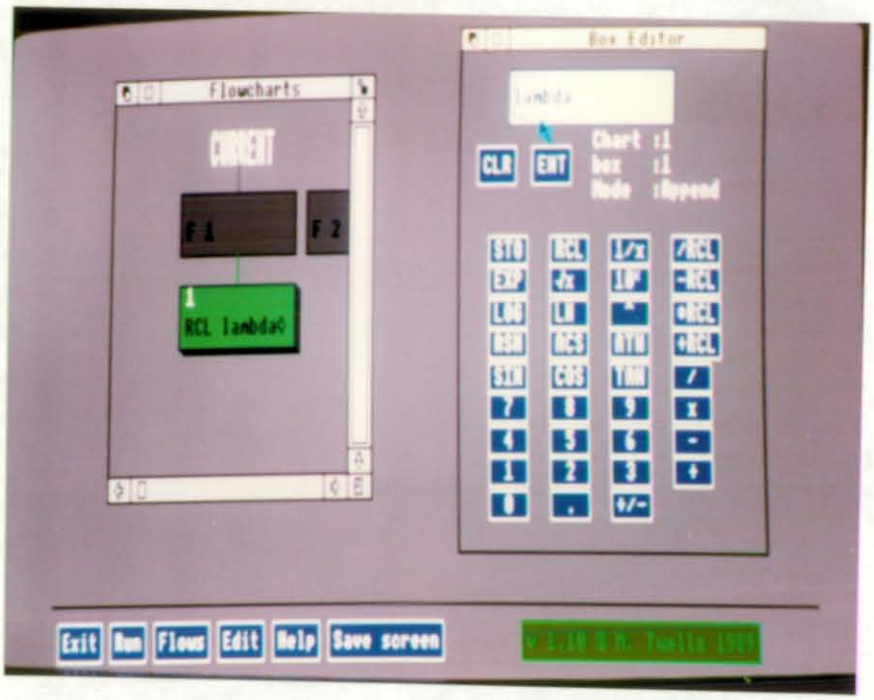
Editing in FLOW

The user is about to add "SIN" to the end of the current flowchart, F1, as box 3.



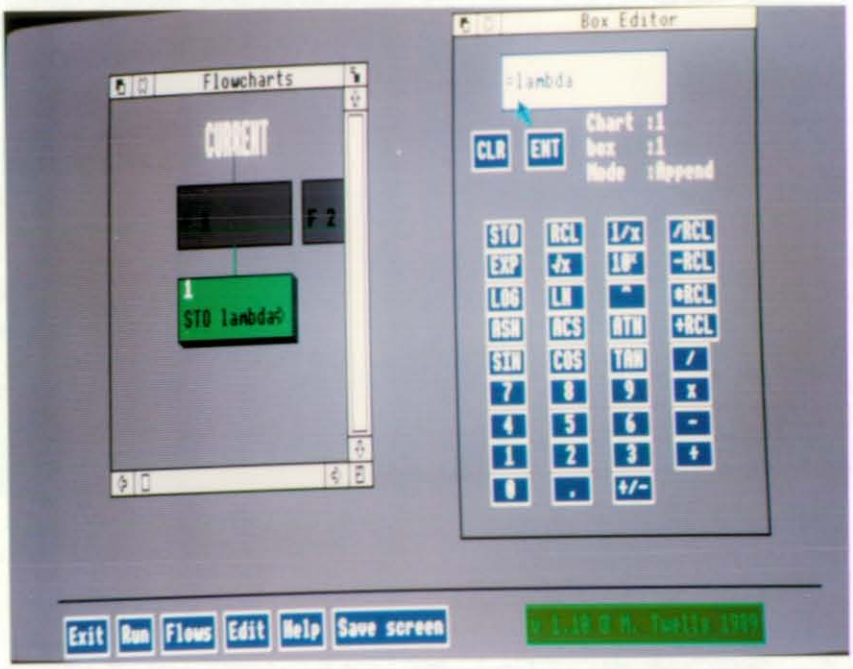
Entering a function

The user has just added ATN to the flowchart as the last box.



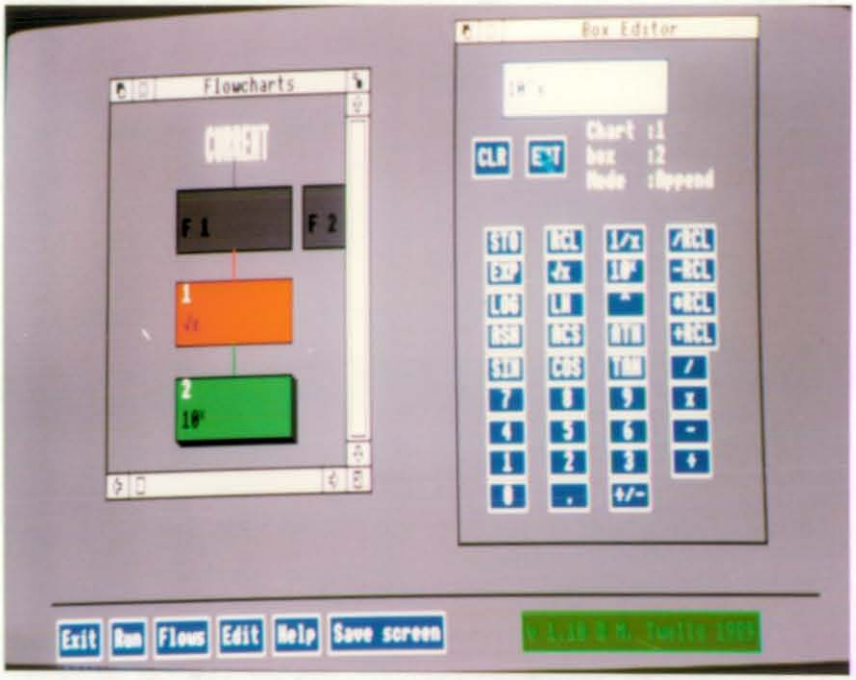
Alias for RCL of a named variable

Although the user typed "lambda" as the box contents, the system has worked out that this is a named variable, and the user intends to RCL its value.



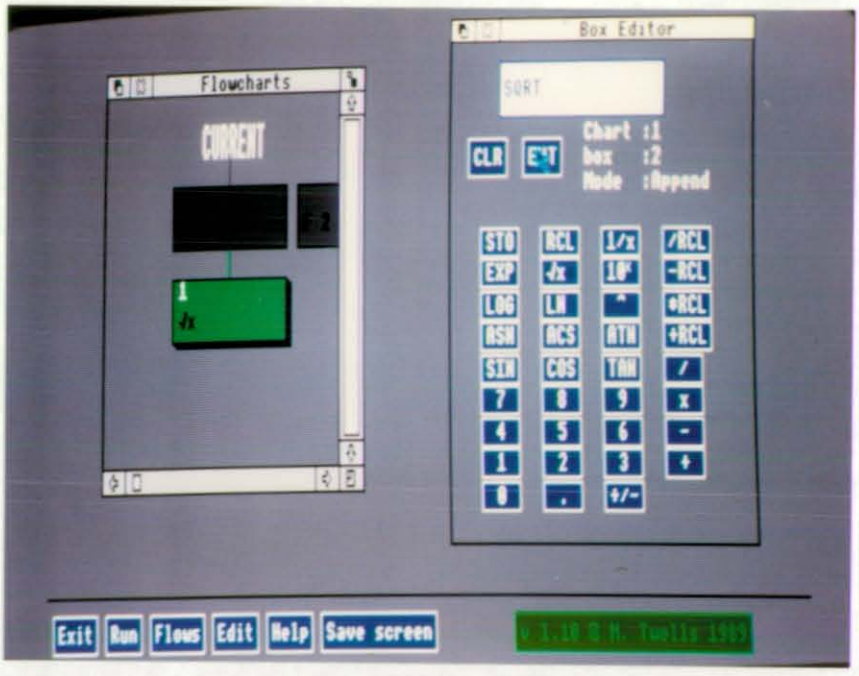
Another Alias

Again, the system has changed the users input. The user typed "=lambda", which has been interpreted as "STO lambda".



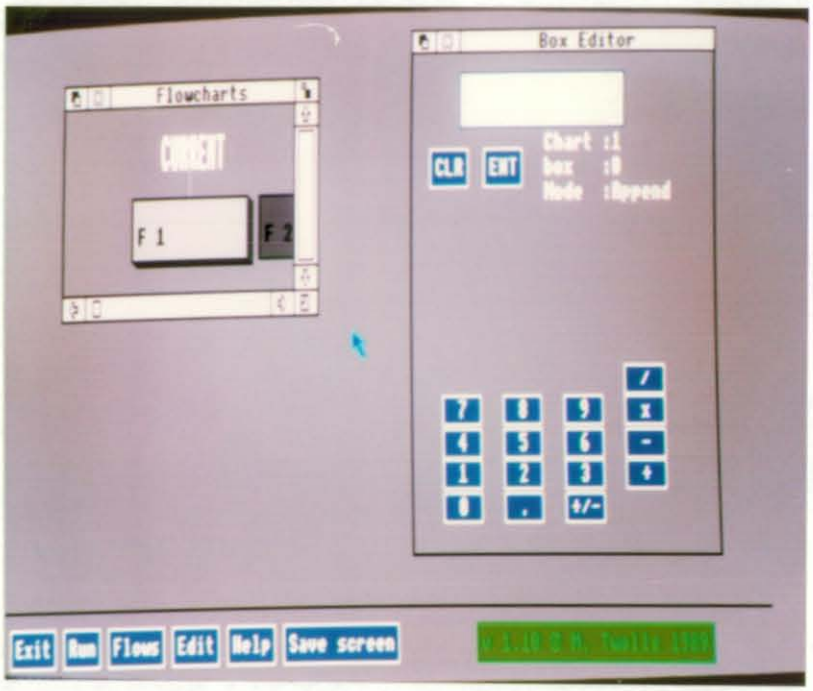
Alias for 10*

If the user types "10^x" at the keyboard, this will be interpreted as 10*



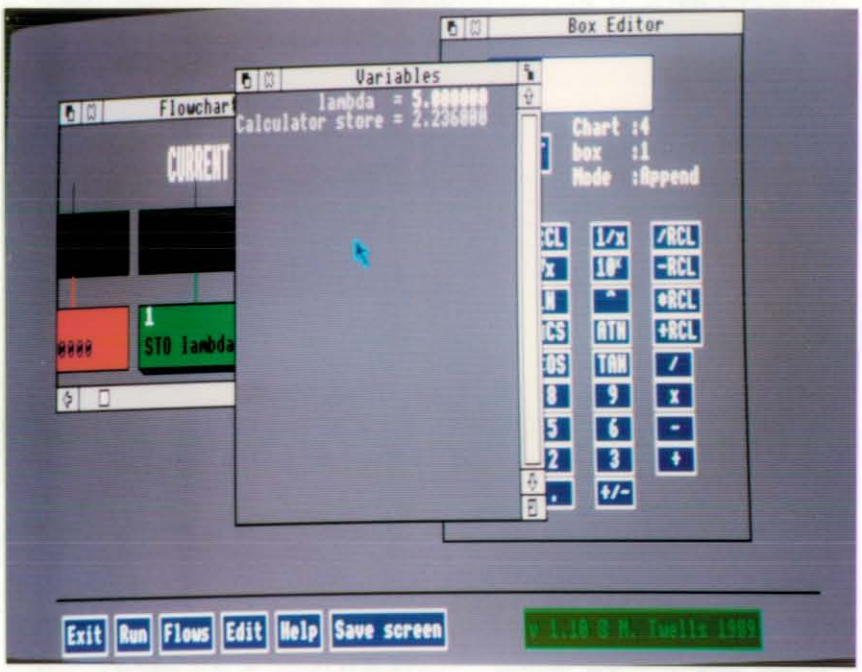
Alias for √x

The user can type , for instance, "SQRT", and the system will spot that this is intended to be √x



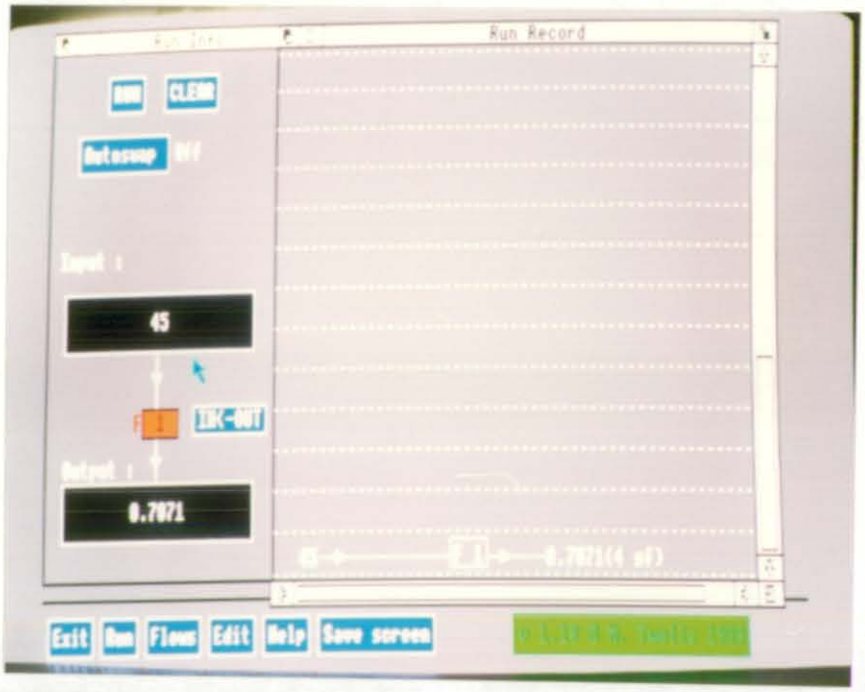
Level 1 Boxeditor

FLOW can be configured to operate at several levels. Level 1 has a very restricted range of functions available, and is intended to be used with younger children.



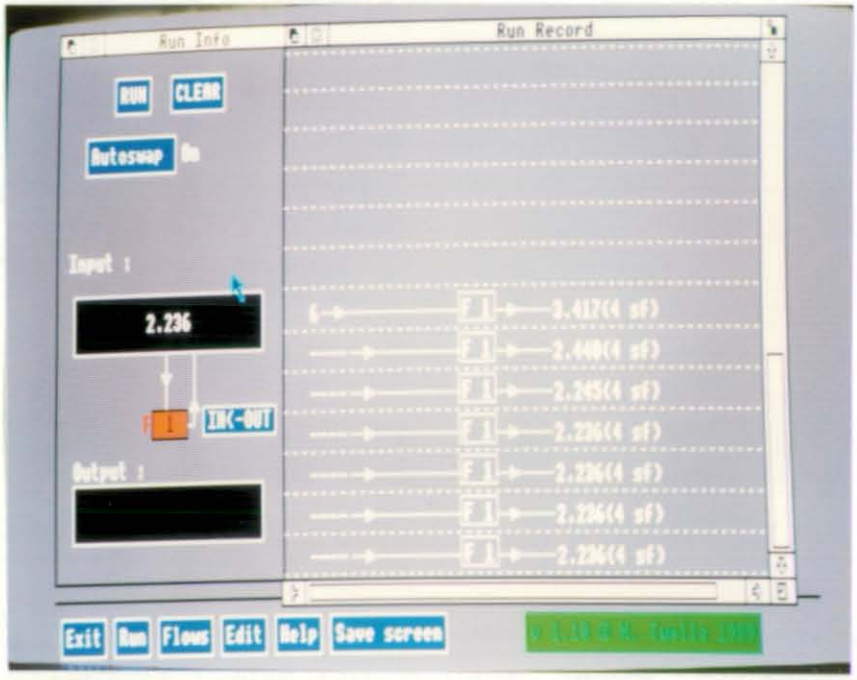
Variables window

If the variables window is open, it will reflect the defined named variables within the system, and also the contents of the anonymous store on the calculator. Using the Variables window menu, (obtained by depressing ***MENU*** over this window) the user can alter, create and destroy named variables.



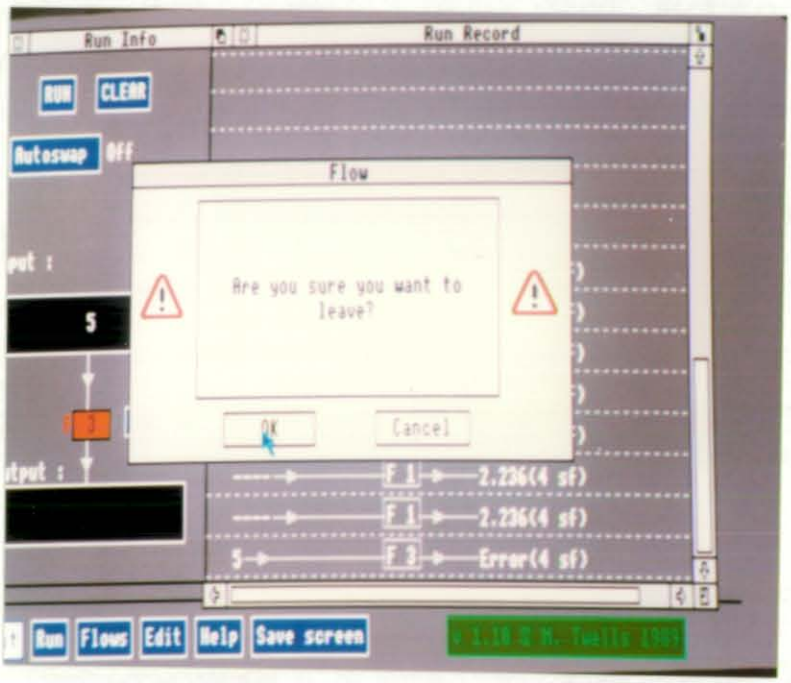
Running without AUTOSWAP

In this mode, FLOW does not use the output of one run as the input for the next one. The old input is retained, unless the user types another number as input.



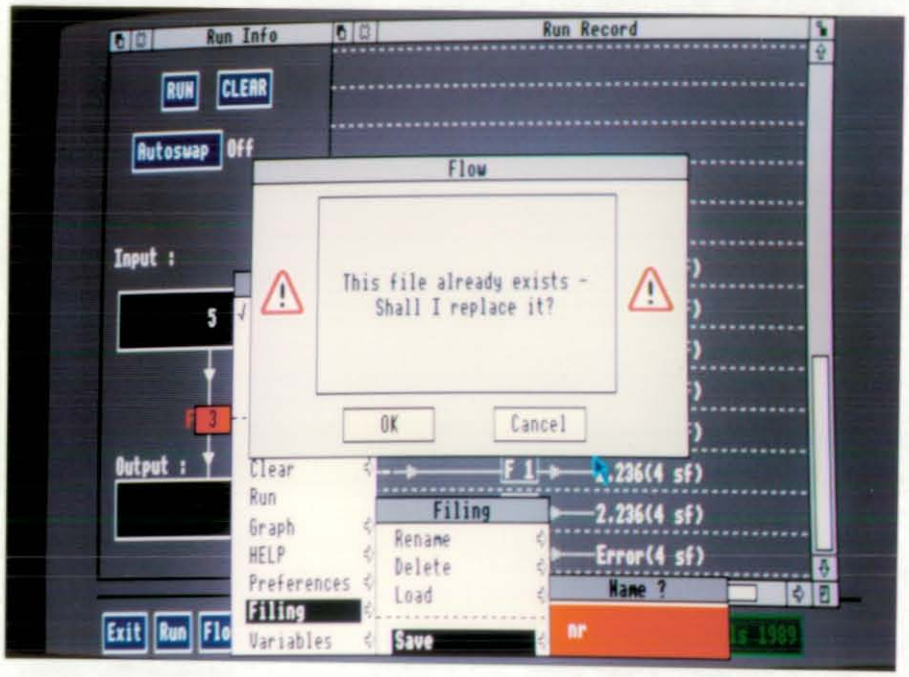
Running with AUTOSWAP

Here, FLOW will use the output from one run as the input for the next run automatically. The output box is not used.



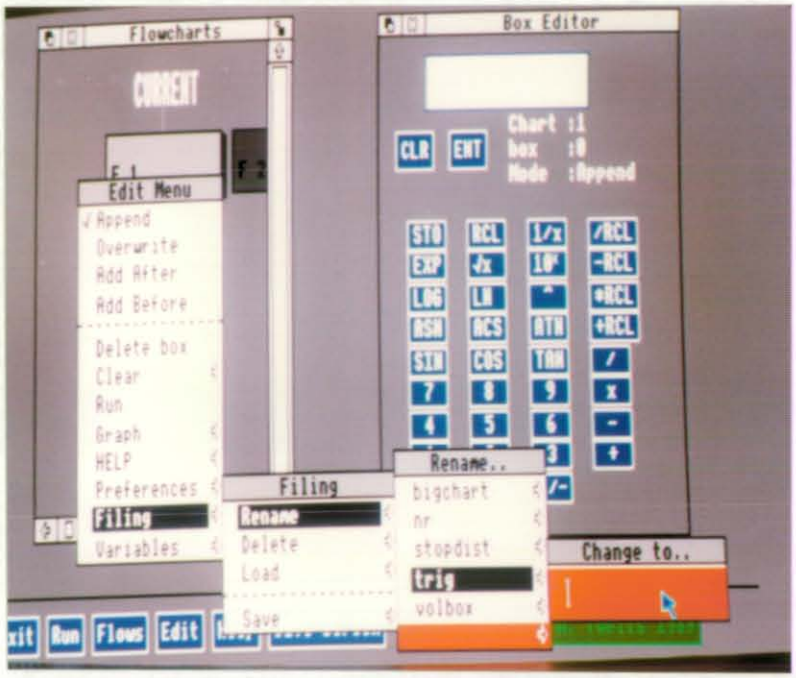
Dialogue box : EXIT

FLOW uses dialogue boxes extensively to communicate problems or to ask the user questions. Here, the user has clicked on "EXIT" on the control panel. FLOW is making sure the user does want to leave.



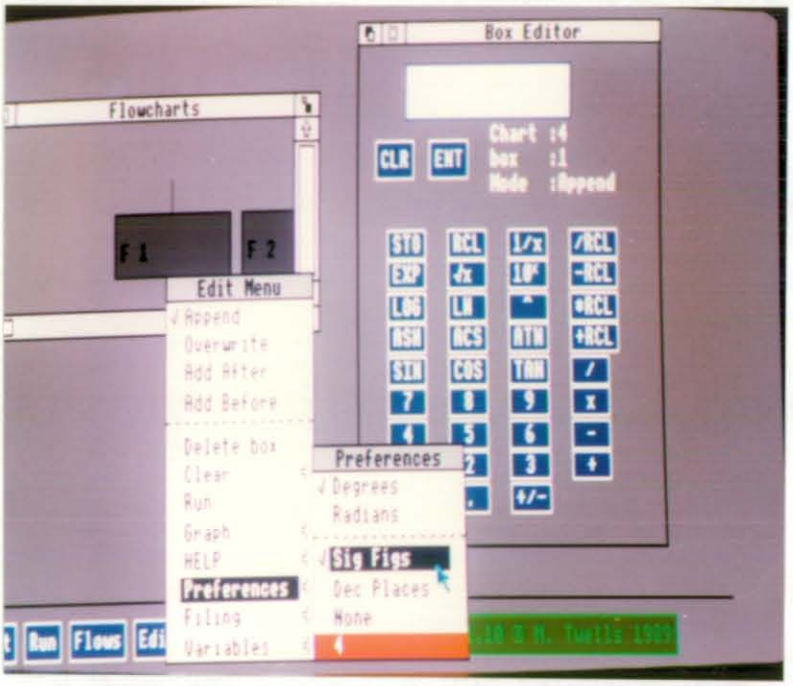
Saving existing FILE

Another example of a dialogue box. The user has chosen to save a file, but FLOW has spotted that the file exists already. Note that not all users are allowed to save, depending on the settings the teacher has made.



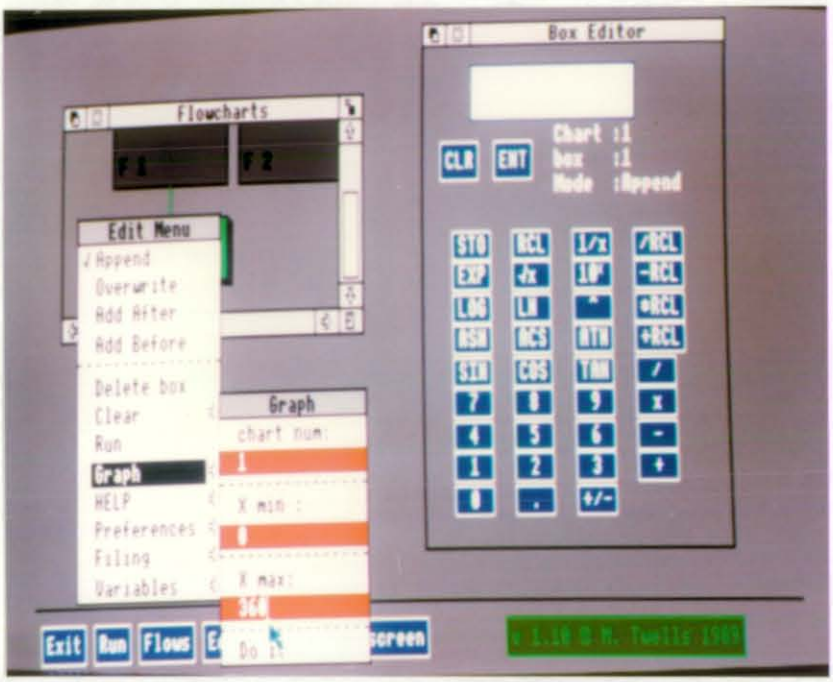
Tree menu structure:RENAME

FLOW uses hierarchical menus to select various options within the package. This is one of the deepest in FLOW.



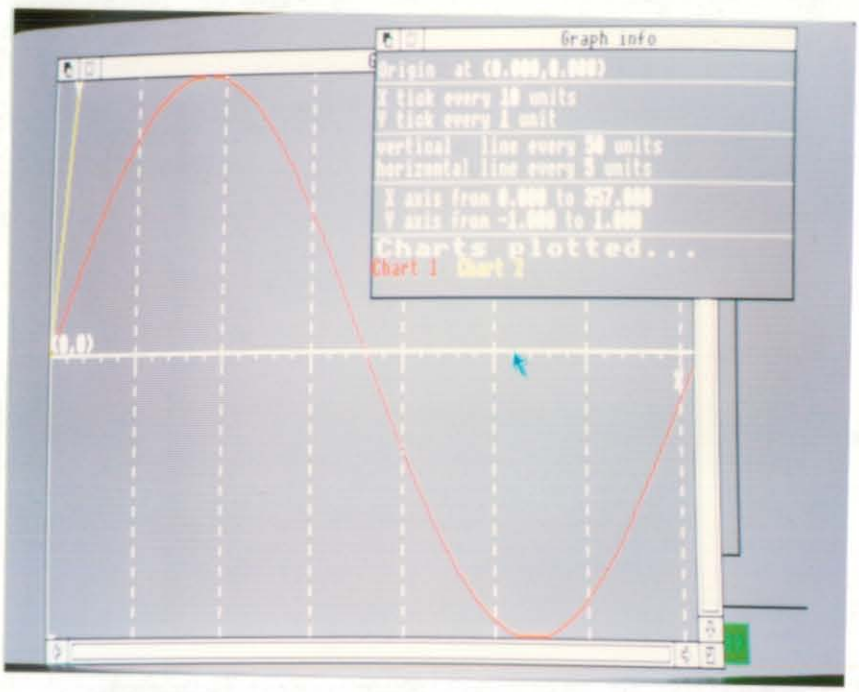
Preferences

Certain user preferences may be set this way. The teacher has an alternative way of controlling these options, and others.



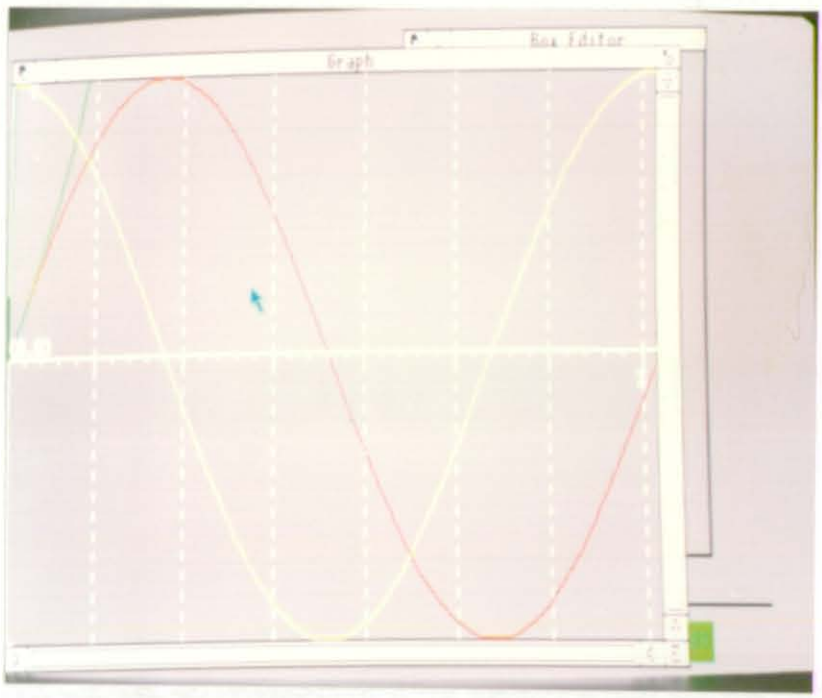
Graphing a chart

Flow provides an easy way of graphing a function represented as a flowchart. The Y axis scaling is automatic.



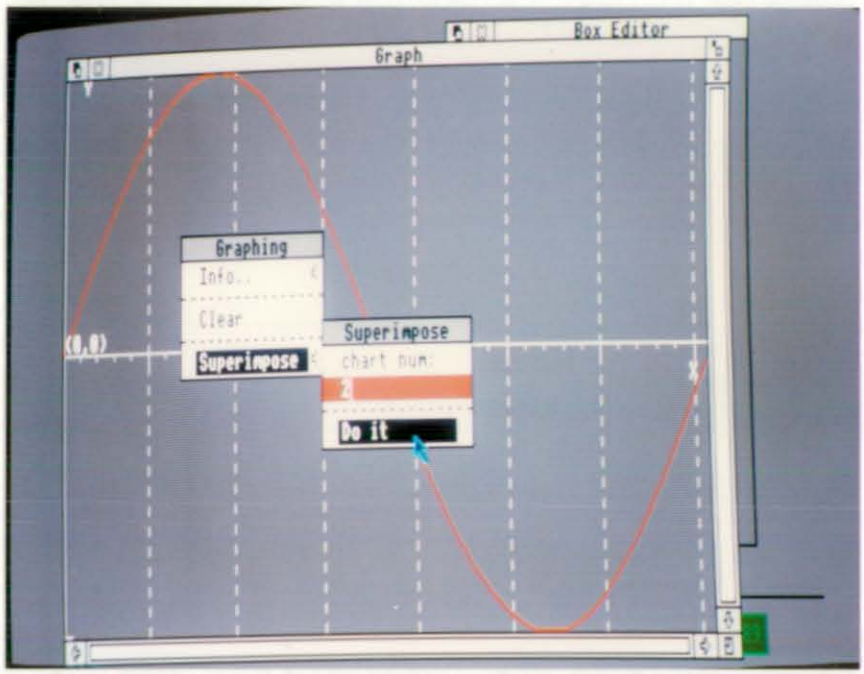
The Graph and Info windows

The Graph window can show up to four curves superimposed. The Graph Info window can be popped up permanently (must be closed explicitly by the user), or temporarily (closes as pointer moves off the window) and gives most information the user requires about the graphs and axes.



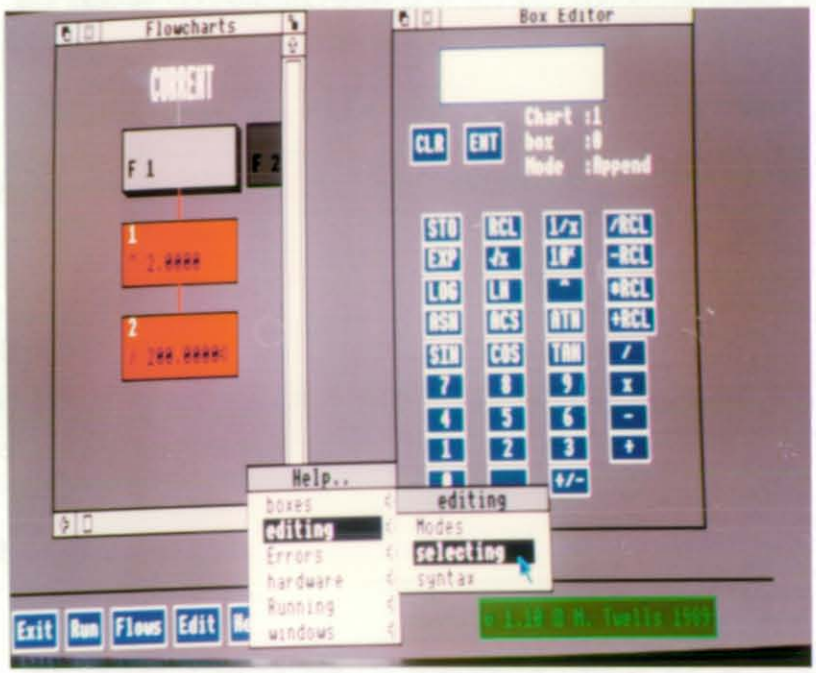
The GRAPH window

Shown with curves superimposed on another. The option to superimpose is contained in the Graph Menu.



Graph menu

Showing the options available to the user. Info pops up the information window permanently if INFO itself is clicked on, and temporarily if the arrow is travelled over. Clear clears and closes the graph window, and superimpose allows a user to add another curve to the graph.

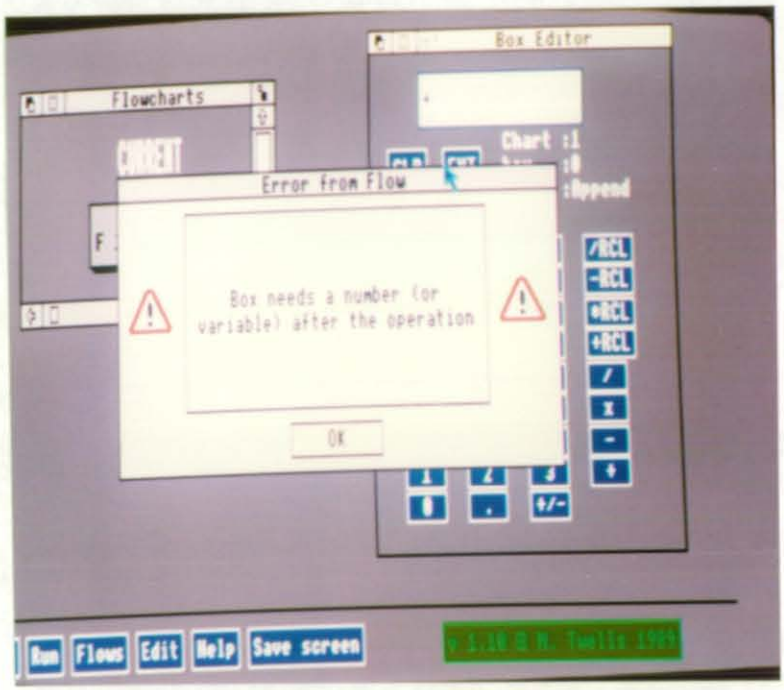


HELP : the MENU

Flow has an extensive help system, completely configurable by the teacher. The menu tree reflects the help directory structure.

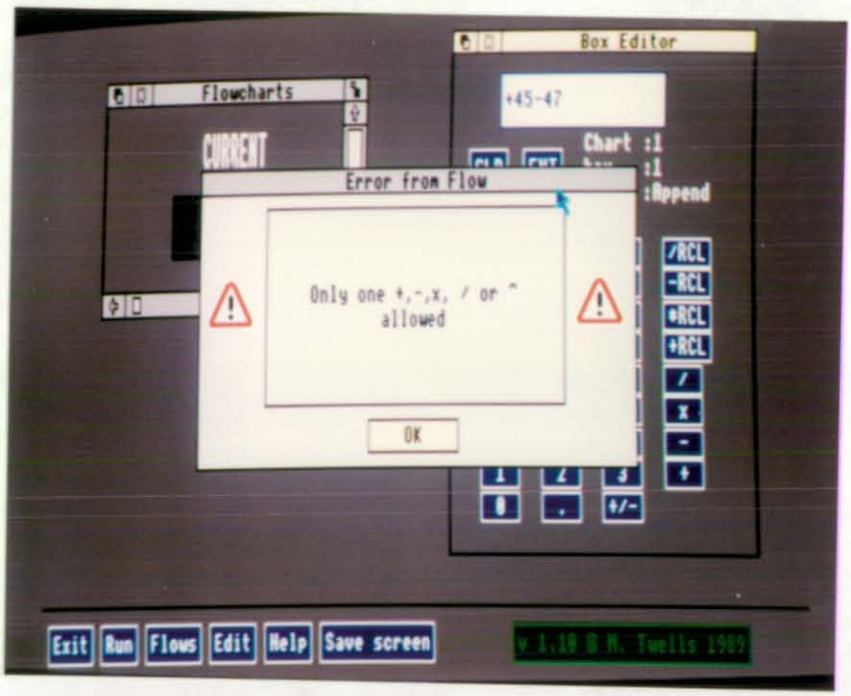
HELP Window

Help windows can be left on screen whilst other operations are carried out

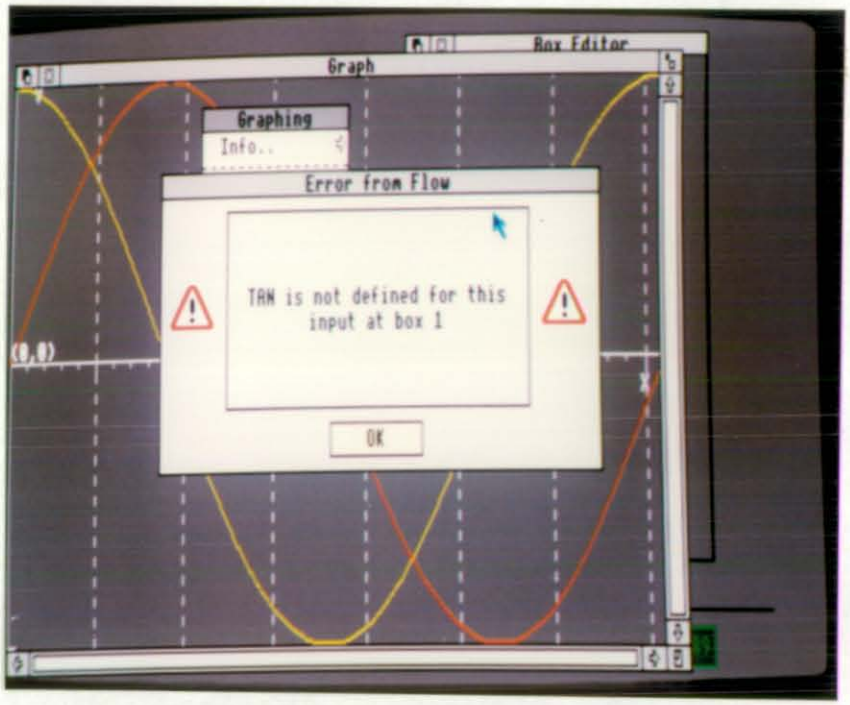


Error message (1)

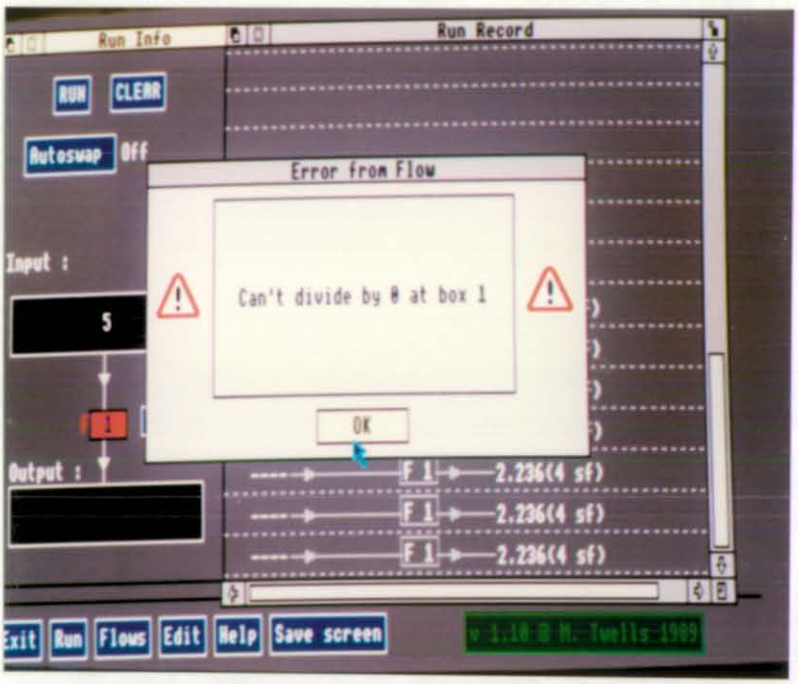
Error messages are intended to be as full and helpful as possible, and will generally indicate the box causing the problem, if it is a run time error.


Box Editor
+45-47
Chart :1
:1
:Append
Flowcharts
CURRENT
Error from Flow
Only one +,-,x, / or ^ allowed
OK
/RCL
-RCL
*RCL
+RCL
/
x
-
+
+/-
Exit
Run
Flows
Edit
Help
Save screen

Errors (2)


Box Editor
Graph
Graphing
Info..
Error from Flow
TAN is not defined for this input at box 1
OK

Errors (3)


Run Info
Run Record
RUN
CLEAR
Autoswap
Off
Error from Flow
Input :
5
Can't divide by 0 at box 1
OK
1
Output :
F 1
2.236(4 sf)
F 1
2.236(4 sf)
F 1
2.236(4 sf)
Exit
Run
Flows
Edit
Help
Save screen

Errors (4)

3 A comparison of similar software

I consider here only the mathematical content of packages which may allow some modelling of functions of one or more variables.

There is little software available in this field for education at present, and it is accordingly difficult to find much with which to compare the package. Many software packages attempt to cover a specific point, or attempt to enable the student to model a specific idea in an unusual way in order to make a teaching point. This is not entirely surprising, since memory and graphics limitations on earlier machines made the manufacture of a multiple or general purpose package which operated satisfactorily in more than one knowledge domain or mode difficult to achieve. Now that graphics have improved, and memory constraints are less severe it is easier to manufacture packages which are not specific to one educational model and which allow the user to "explore" a world and to model in it.

Logotron's Numerator is such a package. It has been developed to run on the BBC range of machines, and an extended version is available for the Archimedes. This package bears a strong resemblance (in terminology) and slight resemblance in terms of control to a well-established package for the Macintosh series of machines, **Stella Modeller**[3].

Both these packages encourage the user to build machines which operate upon numbers. Stella is a particularly powerful package allowing the user to investigate large, complex systems of interacting variables, to model differential equations and hence

[3]Reviewed by Weber,November 1989

solve problems of flow, growth, decay or change using a particularly effective graphical analogy : that of flow along pipes. Thus the user can set or change flow rates by applying a tap to a pipe which is controlled by another "process" elsewhere on the desktop. Numerator uses a similar terminology (Tanks, and pipes) but is nowhere near as powerful. Indeed, those who market the package have not aimed the software at the secondary school user (for some odd reason) but have chosen instead to aim at the junior area of education. With some modifications, the package would be very useful in the secondary classroom in order to allow the student to explore a problem by building a model to represent it, through Numerator.

Both packages allow the user to create processes (my own term, not one specific to the packages under discussion) which to some extent model some real world problems. In particular, it is relatively straightforward to produce functions of one variable in each package, and to graph them.

The terminology used in these two packages is not common in schools. For instance, before Numerator become educationally productive it is necessary to spend some time explaining exactly how number tanks, pipes, operator tanks and so on fit together. Once a period or so is used, then the package begins to become useful. No attempt has been made to use standard GCSE classroom terminology. The same is true of Stella, although in neither case is this a criticism of the software, since neither package is aimed at the GCSE school student. Neither package is configurable in the sense that the teacher can change the appearance or scope of the package in a substantial way. For instance, in Numerator he cannot add or

remove operators (though the student can choose to do so), add tools, or restrict the facilities available to the user. Neither package draws an analogy between a function of one variable and a sequence of keystrokes on a calculator, though of course this is merely a different approach,not a criticism.

FLOW has been built from the start to use a language familiar to the SMP GCSE student. The flowchart is a common way of expressing algorithms. Even though the method has limitations it is still widely used and has the advantage of being easily interpreted even if the user is not familiar with the method. In addition the SMP's habit of considering each (functional) box of a flowchart as an operator, a "machine" with one input and one output, is preserved here. FLOW allows the user first to build flowcharts by tapping keys on a simulated calculator and then to run and investigate properties of the flowcharts.

The calculator image as a representation for various concepts in educational software has been used before. Fox (1988) describes a design for a visual calculator for arithmetic which uses a screen display of a calculator. The author (in the paper) states that his next stage is to implement a prototype on a Macintosh system, so it is unclear whether this system is running yet. In any event, the calculator is not used in the same way as in FLOW. Fox's program is intended to be used by children in helping them to understand arithmetic. To this end, the calculator is used in three ways (Fox, p5) :

(a) by demonstrating how answers can be obtained....

(b) By enabling children to carry out addition and subtraction by their own methods...

(c) By substituting for an ordinary calculator

The calculator is *never* used in the mode used in FLOW : it is used only as an editor in the construction of a stored program.

Evertsz, Hennessey and Devi (1988) have produced a language, GADL, which is intended to facilitate the description of mental arithmetic processes. The concept is that manipulation of numbers and expressions is made possible in a graphical way using tools provided by the system. They report (p1) that "***In general, children learn to use the interface very quickly...We argue that this is is due to the appropriateness of Direct Manipulation in this domain, which in turn is dependent on the success of numbers-as-objects metaphor***". It would seem to be a good idea to try to concretise the concept of a function as an object, and in particular, to use the same method as is already used in existing course textbooks. The representation in Evertsz et al's work uses box-like objects (either single digits or operators) which are hooked together horizontally by the user. The concept is not functional, but the analogy is compelling, certainly for lower school work. At the secondary level a clearer distinction is required between numbers and operators which their notation does not provide. The flowchart - box - sausage machine analogy has been around for some time and is already familiar to most students of mathematics, certainly to students following SMP courses.

4 System chosen for the project software

FLOW runs under the WIMP environment on the Acorn Archimedes computer . requiring a minimum of 1 Megabyte of RAM in which to run. It conforms to the suggested Acorn guidelines for (non-multitasking) WIMP based software and accordingly presents to the user familiar with these machines a standard interface. The Archimedes was the computer of choice for a number of reasons :

1) It is becoming available in schools
2) It is cheap (especially the newer 3000 series) . and offers full colour, unlike the original Macintosh.
3) It is much faster than the Macintosh in almost every respect. The rate of redraw on the Macintosh is slow and users have found this to be irritating.

Although the software has been written for the Archimedes the design is modular in the sense that it is possible to rewrite the "low level" modules (the interface to the WIMP, for instance) and leave the high level modules alone. The program should then work. Although modularity was felt to be useful, I have not made the compartments completely watertight and some modifications to the high level modules will be required in order to run, for instance, on a monochrome set up. Ultimate portability without complete rewriting is possible since the program is written almost entirely in ANSI C using only one non-standard library (The Acorn RISCOS library in order to use the WIMP). Most of the RISCOS WIMP functions are called only in one module, w_shell . and only a few others are used in the other modules.

The software allows the user to construct and run flowcharts in a fashion analogous to the style used in the SMP GCSE courses. The user builds the flowcharts by tapping keys on what appears to be a calculator, in an exact analogy of evaluation of a function in "real life". As the user presses the keys (using the mouse or by typing) the flowchart builds up in the other window. Rudimentary editing functions are provided to allow the user to erase boxes, edit boxes and so on. When the user has completed a flowchart he can then "run" numbers through the flowchart and record the results. They correspond mathematically to functions of one variable although using a contorted method involving named variables the user can construct a function of more than one variable. This is not intended to be used frequently. The named variables are provided in order that the teacher can, if it is wished, demonstrate the similarity (and differences) between a sequence of key presses on a calculator and a stored program.

The core of the program is a simple language interpreter and editor, but the presentation of the program "listing" and the method of editing it is an obvious difference from standard interpreters. As is normal with educational programs, the core interpreter is small, simple and straightforward. The surrounding presentation is complex and involves (probably) 80% of the source code, the help system accounting for a substantial portion of code and text files.

Configurability is not, of course, limited to help contents and structure as examined earlier. Many of the features of the package are wholly configurable. The package, when it runs, looks for a configuration settings in system variables, placed there by the file starting FLOW. It adopts these settings, which the user is

subsequently unable to change. Thus the teacher may disable features such as the availability of the trigonometric functions SIN, COS, and TAN with a class of 11 year olds. Later on, complex facilities such as named variables, hyperbolic functions and so on can be introduced easily, by using a different configuration file. Trivial but no less important features such as colour are completely programmable by the teacher in a standard way (using the !palette file of the archimedes, which is easily editable from the desktop). Further features are controlled by environment variables set by a boot file before starting.

5 Method of use

Clearly the exact method of using this package depends on the teacher involved. It is intended as a package in which the user is able to **explore** mathematical ideas. but can easily be adapted to work in the manner of an electronic blackboard. Worksheets can be constructed and a sample collection is attached. Care needs to be exercised in constructing worksheets since the terminology must agree with both the program and the course being followed.

The program pursues an analogy between presses of keys and progress through a flowchart. This analogy is found in many elementary courses in mathematics[4] and is the source of some frustration for less mathematically inclined pupils. The ability to relate function key presses to parts of a functional flowchart is important. In addition, the program permits investigation of more complex ideas such as convergence and iteration. The "language" used is based upon the keys found in many elementary scientific calculators although additions and alternatives have been provided. The major omission as far as teaching programming is concerned is that no loop or decision-based flow control is possible. This does not detract from the programs usefulness in elementary modelling of simple mathematical functions is concerned. It merely prevents it from being used to teach elementary programming graphically. It is in any case arguable that flowcharting is a good way of teaching it.

---

[4] e.g. The SMP GCSE course, SMP New Book 3 part 1, New Book 4 Part 1

6 USER DOCUMENTATION

This documentation details FLOW v1.20

6.1 Terminology

Some jargon is unavoidable. and I assume some slight familiarity with WIMP systems in general. The following are brief notes documenting some peculiarities of the Acorn system.

Upon turning the machine on. the bar across the bottom of the screen is known as the ICON BAR. Devices appear at the left hand end of the bar. "managers" at the right.

The mouse has three buttons. Holding the mouse with the cable emerging away from the user. the left hand button is called ***SELECT***. the centre one ***MENU***. and the right hand one is ***ADJUST***. These names are generally indicative of the function the button is intended to perform. FLOW does not make much use of ***ADJUST***. When the pointer is placed over an icon. window or the desktop background. pressing ***MENU*** will generally pop up a menu indicating what options are available to the user. With ***SELECT***. one press generally "selects" something. a second press following on rapidly from the first will "run" it..the nature of "run" depending on the object selected. If you pause too long. a second press will "deselect" the object.

The package makes extensive use of windows. The way the system deals with of menus and keypresses depends on the position of the

pointer in relation to the windows on screen. The novice user is actively encouraged to experiment with moving the windows around on-screen. If the user examines a normal window, he will find 3 buttons along the title bar at the top of the window. The left-hand most symbol, rather like a sheet of dark paper almost covering a light sheet, will, if clicked on with ***SELECT***, send the window to the back of the pile of open windows. The cross, to its immediate right, will close the window. The button in the top right hand corner will toggle the window to full size, or shrink it. In the bottom right hand corner is a button resembling a sheet of paper with a corner missing which if dragged (by holding ***SELECT*** down and moving the mouse)will resize the window. The bars along the bottom and up the right hand side scroll the window around over the full work area.

Clearly a description of the window environment is two paragraphs is less than full, and a suitable description can be found in the Acorn User Guide (1989). It is recommended that users new to the system spend some time trying these buttons out with (for example) directory viewers before trying applications

6.2 Starting the package.

Insert the disk provided into the disk drive, label uppermost. Click once with ***SELECT*** on the disk drive icon in the lower left hand corner of the screen. This should open a window onto the disk contents. Click ***SELECT*** twice in rapid succession on the FLOW application icon, which is a green flowchart box and a letter "f". The text "!flow" appears nearby. FLOW chart files consist of a similar icon , but with a number of linked boxes in, one of which is red. They may be started by double-clicking ***SELECT*** on them in an essentially similar way - FLOW itself is loaded automatically.

followed by the flowchart file.

There will normally be a short pause during which FLOW installs itself, probably accompanied by the appearance of an hourglass on-screen. The software will present itself to the user with no windows open initially, and only a menu bar (most conveniently thought of as the "control panel") across the bottom of the screen.

6.3 THE CONTROL PANEL

There are 5 icons towards the left hand end of the control panel which, when clicked on with the ***SELECT*** button (the left hand button) will normally open windows which have functions as indicated by the text on the bar. Clicking ***MENU***, the middle button, anywhere, will open the Edit menu, from which selections can be made, unless the pointer is over the HELP icon, in which case the HELP menu will be selected. ***ADJUST*** has no effect, unless over the help menu.

Clicking ONCE with ***SELECT*** (left hand button) on the ICON	Will...
EXIT	Will exit from the program completely, back into the desktop, after a prompt of the "Are you sure" variety.
RUN	Will open two windows, the runentry window, and the runrecord window. The runentry window is used to enter numbers as input to flowcharts. The runrecord window is used to record the results of the last thirty runs. The Run windows cannot be used to edit the contents of flowcharts themselves.
FLOWS	Opens the flowchart window. This window is used to display currently stored flowcharts.
EDIT	Opens the boxeditor window (the "calculator") and the flowcharts window, ready for editing or creating flowcharts.
HELP	Opens a menu containing help items. The precise nature of the help available depends upon the configuration of files on the supplied disk, and will vary considerably. In fact clicking ANY button on this icon will open the help menu, unlike the other icons on the panel. The philosophy here being that if the user needs help at this stage he is unlikely to know which button on the mouse to press!

SAVE SCREEN Is intended mainly for program development, and for the production of the illustrations. It may prove to be a useful facility for teachers developing their own documentation and worksheets. The entire screen area is saved to disk as a sprite called "screen", in a file called "screen". This icon will only be present if flow$advanced is set on (see later)

6.4 A QUICK GUIDED TOUR

6.4.1 Tutorial (1) : a simple flowchart

* Click ***SELECT*** (the left mouse button) once whilst the pointer is over the EDIT icon on the control panel. This will open both the flowchart and boxeditor windows. The user can then edit and construct a flowchart.

* If necessary, scroll the windows until you can see the top boxes in the flowcharts. If the box labelled F1 does not have the word CURRENT above it, click ***SELECT*** on the grey header box, containing the text "F1". It should become the current flowchart. Any modifications or additions will only affect this flowchart.

* Enter "x 1.83" into the boxeditor by clicking ***SELECT*** on the required keys in succession, and watch the box contents build up in the boxeditor window. When you have finished, click with ***SELECT*** on the ENT (short for ENTer)

button to transfer the contents into the flowchart. Any typing errors found will be indicated at this stage, and the box will not transfer. If you make a mistake, click ***SELECT*** on OK in the error dialogue box, then CLR on the BoxEditor window, and start again (there are better ways of editing : see later).

* Tap the middle mouse button, ***MENU***, and move the pointer down to "Run" in the menu. Click ***SELECT*** on "Run". This should open the RunRecord and RunEntry windows.

* Ensure the pointer is over one of the two RUN windows. Type the digit "5", and follow this by depressing not RETURN, but the red key F1. The flowchart should run, and the output should be the results of multiplying 5 by 1.83, displayed to the preset precision.

* Quit FLOW by clicking ***SELECT*** over Exit, on the Control panel, then click ***SELECT*** on "Ok" in the dialogue box.

6.4.2 Tutorial (2) : Loading and running a flowchart

* Start up FLOW in the manner described above. Depress the middle mouse button, ***MENU*** anywhere over the main screen. The Edit menu should pop up. Move the mouse pointer over "Filing", and then carefully to the right over the right-facing arrow. Follow that with the same operation over "Load", and click ***SELECT*** over the item "nr" from the Load menu.

* Open the flowchart window by clicking ***SELECT*** on "FLOWS", if the window is not already open.

* After a short pause you should see the flowchart window resize itself, and the flowchart appear in the window. You should be able to see most of the flowchart (which calculates √5 using the well-known Newton-Raphson method. A sample worksheet is included later in this volume) by scrolling the window, or resizing it.

* Click ***MENU*** anywhere over the windows or background, and then click ***SELECT*** over "Run" in the menu.

* Click ***SELECT*** on "AUTOSWAP", which has the effect of placing the output from one run of the flowchart back into the input.

* Ensure the pointer is over either the RunEntry or RunInfo windows.

* Type in an initial guess (say 10).

* Depress F1 repeatedly, and watch the iteration build up in the RunInfo window.

* Click ***MENU*** whilst the pointer is over a window or the background. Move the pointer to the right over "Clear" and its associated arrow, and then click ***SELECT*** over "All Charts", to clear the flowcharts. Repeat the operation, this time clearing the "Run Record".

6.4.3 Tutorial (3) : Graphing a flowchart

* Load FLOW in the usual way, if you have not already done so.

* If you had flow loaded from the previous exampie, Click on "AUTOSWAP" to ensure it is turned off.

* Click ***MENU*** over the main area above the control panei, select "Filing", "Load" as in the previous example, and eventually seiect the file "volbox". This flowchart calculates the volume of a box which is obtained by cutting a variable sized square (which is the input to the flowchart) out of each corner of a rectangular sheet of cardboard of dimension 297 x 210. Examine the flowchart by scrolling the flowchart window over it.

* Click ***MENU*** again, this time moving to "Graph". Move the pointer over the right pointing arrow, and down to the first RED entry in the menu. This entry is writeable, as indicated by the presence of the I-shaped caret in the box.Using the DELETE key (in the group of six, just to the right of the main keyboard group) erase the default entry and insert "1" (by merely typing on the numeric keys) underneath the "Chart Number" entry (do NOT press RETURN when you have finished!). Deleting and inserting as necessary, type "0" underneath "x min", and "100" underneath "x max".

* Click ***SELECT*** on "do it". The graph window should appear, and a graph of the dimensions of the missing square (x

axis) should be plotted against the resulting volume (y axis).

6.4.4 Tutorial (4) : Editing a flowchart

* Load FLOW as documented. and Load "volbox". as shown above.

* Click ***SELECT*** on the Edit icon on the control panel.

* Click ***SELECT*** on (say) "SIN" in the boxeditor. then click ***SELECT*** on ENT in the boxeditor. The contents will be transferred across to the end of the current flowchart. You have "appended" a box to the current flowchart.

* Scroll the window up by dragging the scroll-bars until you can see box 4. "+ 210.000"

* Click ***SELECT*** over this box. It will turn Green. indicating that it is the currently selected box. Its contents are transferred across to the boxeditor.

* Click ***MENU*** whilst the pointer is over the flowcharts window. then ***SELECT*** on "Overwrite".

* Move the pointer into the boxeditor. and click the pointer between the "1" and the "0" of "210". The Caret should appear.

* Depress the DELETE key. then type "5". The number should have changed to "+ 250.0000"

* Click ***SELECT*** on "ENT" to transfer the contents back to

the chart. The original box will now have changed to read "+ 250.000"

* Drag the scroll bars down until you can see box 12. the SIN box you added to the chart.

* CLick ***SELECT*** whilst the pointer is over this box. followed by ***MENU*** to pop up the Edit Menu. Finally. click ***SELECT*** over "Delete Box". You have removed a box from the chart. Had this box been in the middle of the chart. the one above and the box below would have joined up over the gap.

6.4.5 Tutorial (5) : Named variables

* Load. as before. the flowchart file "QROOTS".

* Examine its contents by scrolling around the window (making it full sized by clicking on the top right-hand corner button may help you). You will see that F1 computes the discriminant. $\sqrt{(b^2-4ac)}$ of a quadratic equation

$$ax^2+bx+c = 0$$

and that F2 and F3 compute the roots of this equation. using F1 as a "subroutine".

* Open the Edit Menu by clicking ***MENU***. move down to "Variables". and click ***SELECT*** over this - moving over the right pointing arrow *will* open the window. but only temporarily. that is until the pointer moves off it. Clicking on it opens the window until it is explicitly closed by the user.

* Click ***MENU*** whilst the pointer is over the variables window. This opens the variables menu.

* Move to "Create Var". and over the right pointing arrow.

* Type in "a". and over the next arrow. Type in "1" here. and press RETURN.

* Repeat the above process to create variables "b" = "5" and "c" = "-8"

* Send the variables window to the back of the pile by clicking ***SELECT*** on the icon in the very top left hand corner (to the left of the cross).

* Move the pointer over the flowcharts window. click ***MENU***. click ***SELECT*** over RUN.

* Depress F2 (no input is required). The output.1.275. is one root of the quadratic equation

$$x^2 + 5x - 8 = 0$$

expressed to 4 sf.

* Depress F3 to obtain the other root. -6.275

* Juggle with the windows. closing if necessary until you can see the variables window.

* You should see that the variable "disc" has been created.and has a value currently of 7.549834.

6.4.6 Tutorial 6 : Configuring FLOW

* Open a directory viewer showing the !FLOW application. On the supplied disk this should also have the !EDIT application visible.
* Double click ***SELECT*** whilst the pointer is over the !EDIT icon. The icon should reappear on the right hand end of the icon bar.
* Whilst holding down the SHIFT key, double click ***SELECT*** on the !FLOW application. Instead of installing it (which would have occurred without SHIFT) you will now be able to see the contents of the application.
* Drag the !RUN file icon by depressing the ***SELECT*** button over it, and dragging with ***SELECT*** pressed, until it (in fact, the pointer) is over the !EDIT icon on the icon bar, and let go of ***SELECT***. This "loads" the text into the editor.
* Make the window full-sized by clicking ***SELECT*** on the toggle to full-size icon in the top right hand corner of the window.
* The Caret, an I-like pointer, should be visible in the top left hand corner of the window itself. This responds to clicks with ***SELECT***, which will position it, and also to cursor key movements. Move it down until it is positioned at the end of the line which starts

  ```
  set flow$oplevel 5
  ```

* Delete the 5, by tapping the DELETE key, and replace it with a 1. This configures FLOW to work at its lowest

operation level, allowing only the basic 4 functions, and is documented fully later

* Click ***MENU*** on the mouse, move down to "save", and over the right facing arrow.

* Click on OK in the box which opens.

* Close the EDIT window by clicking on the cross in the top left hand corner.

* Unload the EDIT application by clicking ***MENU***, whilst the pointer is over the EDIT icon on the icon bar (the pen and ink), then Click ***SELECT*** on "Quit".

* Now load FLOW, and examine the boxeditor. You will see that only the basic 4 functions remain.

6.4.7 Tutorial (7) : Editing and adding to the HELP facilities

* Open a directory viewer onto FLOW.

* Hold down the SHIFT key and double-click ***SELECT*** to get "inside" the FLOW application.

* Double click ***SELECT*** on the Help folder (coloured blue).

* Suppose we want to edit the help filed as "editing => syntax". Double click on the "editing" folder.

* Now either drag the syntax icon to !EDIT, or double click on it (EDIT recognises it as its file, so there is no need to drag, unlike !RUN in the previous tutorial). The text will be loaded into EDIT, and can be saved as previously

described. If you choose to create new folders (Use the ***MENU*** button over a directory viewer) or new applications (use "CREATE => Text file" in the EDIT menu obtained by clicking ***MENU*** over EDIT on the icon bar) then the menu structure inside FLOW is adjusted automatically.

6.4.8 Adding and editing : general points

To add or edit a flowchart's boxes there must be two windows open on screen : the Flowcharts window, and the boxeditor window. These may be opened (if they are not already open) by clicking ***SELECT*** on the appropriate icon on the panel at the bottom of the screen. Clicking ***SELECT*** on the EDIT icon will open **both** windows. clicking ***SELECT*** on FLOWS will open only the flowchart window. The Flowcharts window will open initially to the left of the editor window. The editor window appears rather like a calculator.

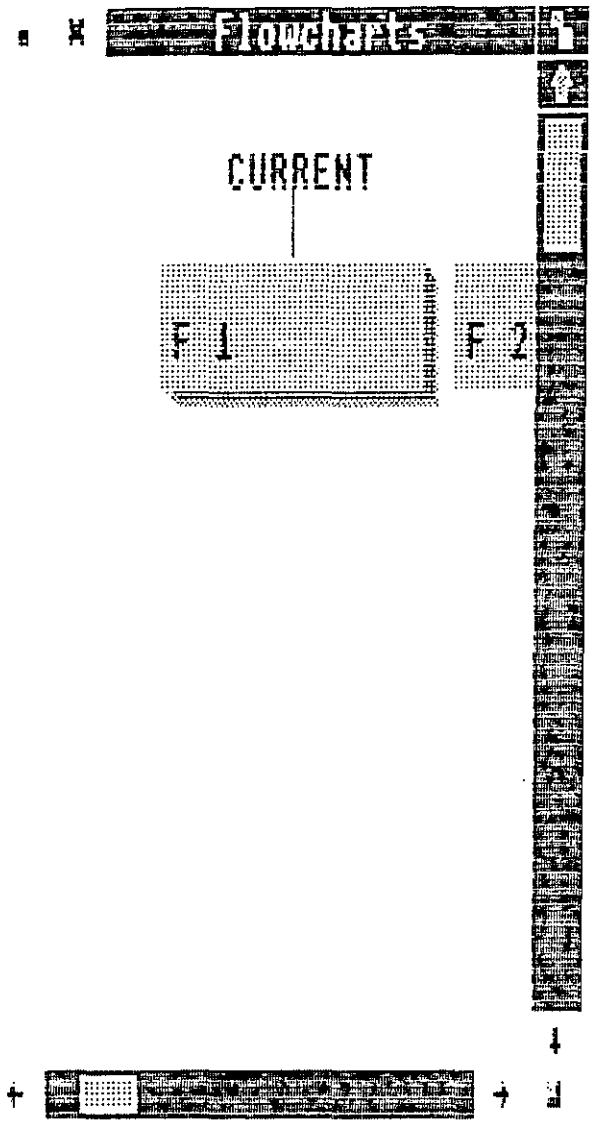


Fig (1)
The flowchart window

The window can be manipulated as described earlier. it can be reopened by clicking on the FLOWS icon at the bottom of the screen, on the control panel. The work area of the window will grow vertically as longer and longer flowcharts are constructed, and will shrink to its initial state if all flowcharts are cleared. The window may of course be resized and scrolled over the work area in the usual way.

The flowcharts window is opened initially to show the contents of flowchart 1, which will normally be empty unless the teacher has

set up an autoload sequence.

The currently selected flowchart is distinguished from others by being surmounted by the word "CURRENT". In addition, it will normally contain a box which is coloured green, and has a heavier border (for colourblind and monochrome monitor users). This is the *currently selected box*. FLOW does its best to keep this box visible inside the Flowcharts window, in order that you do not need to scroll around to find it. Only the current flowchart and (in some edit modes) the currently selected box can be changed by the boxeditor. The "header box" for the flowchart, the top one, contains the number of the flowchart, and is coloured differently to the rest because it cannot be changed or deleted. It can be selected, but any attempt to overwrite the contents will cause an error.

You can load a prepared flowchart for editing or running by opening the EDIT MENU. You do this by depressing ***MENU***, the middle mouse button, anywhere over the background (it will work on most windows as well). Move the pointer down to "filing", and over the right facing arrow (right facing arrows indicate the option leads to a sub-menu). Move the pointer over "Load", and select the chart you wish to load. All the FLOW charts found in the default area for saving and loading are listed. The red bar at the bottom is writeable, and you may type your own choice here. Alternatively, and safely for a first time load, simply click ***SELECT*** over one of the items in the load menu.

To select a flowchart for editing you must make it the CURRENT flowchart. If you cannot see the flowchart in the window, scroll or

resize the flowchart window by dragging[3] the elevator bars in the required direction (the flowcharts run from 1 to 30 horizontally across the top of the window), or click ***SELECT*** on the arrows until the one required is visible. Then click ***SELECT*** anywhere underneath the flowchart header box, or in the header box itself. This should result in the CURRENT indicator moving across to above your chosen flowchart. If you clicked in the header box itself then that will turn a lighter shade of grey, otherwise clicking on any other box will turn it green with a heavier border. The colour and border change indicate the currently selected box.

Pressing the middle mouse button, ***MENU***, whilst the pointer is anywhere within a window, or on the control panel, or on the desktop background will pop up a menu. This is almost always the EDIT MENU, the only exception being the Graph window menu which is documented later.

The first four items on the edit menu, APPEND, OVERWRITE, ADD AFTER and ADD BEFORE change the way the boxes are added into the flowchart. APPEND will always add new boxes on to the end of the currently selected flowchart and ignores the currently selected box. OVERWRITE will overwrite the currently selected box (if there is one) with the boxeditor contents. ADD AFTER will add after the currently selected box, and ADD BEFORE will add the box in before the currently selected box, unless the currently selected box is the header box. The edit mode currently selected is indicated by a tick against the left-hand end of the menu entry. APPEND is the usual, default, method of working. Note that it is not possible to enter any operations in the header box, the first one on the flowchart. In

[3] Move the mouse with ***SELECT*** depressed

this version of FLOW this box merely indicates the flowchart number to the user.

The next section of the edit menu (below the first dotted line) contains two entries : DELETE BOX and CLEAR.

DELETE BOX Will delete the current(GREEN) box, if one is currently selected. Use this if you make a mistake.

CLEAR Has a right pointing arrow after it, indicating that it leads to a sub-menu. Moving the mouse pointer from left to right over the arrow will move the user into a sub-menu, entitled CLEAR. This has four options, documented in more detail later. If you want to start with a clean sheet, click ***SELECT*** on ALL CHARTS.

The first two sections of the edit menu are the only ones directly appropriate to editing flowcharts in memory.

6.4.9 Entering boxes

To enter a box into the currently selected flowchart the user may employ several different methods. (S)he may wish to enter box contents by clicking with the mouse on the boxeditor (calculator) buttons. This is the simplest way to begin with, and is the only method recommended for younger users. Alternatively the user may type the required contents directly. To do this, ensure that the boxeditor is highlighted (the title bar is coloured cream, not grey)...no input will be accepted unless this is the case. The

boxeditor will be highlighted if the mouse pointer is over the boxeditor, or over the flowcharts window. Typing in directly is the only way to enter some more advanced operations: named variables and hyperbolics, for instance, can only be entered this way.

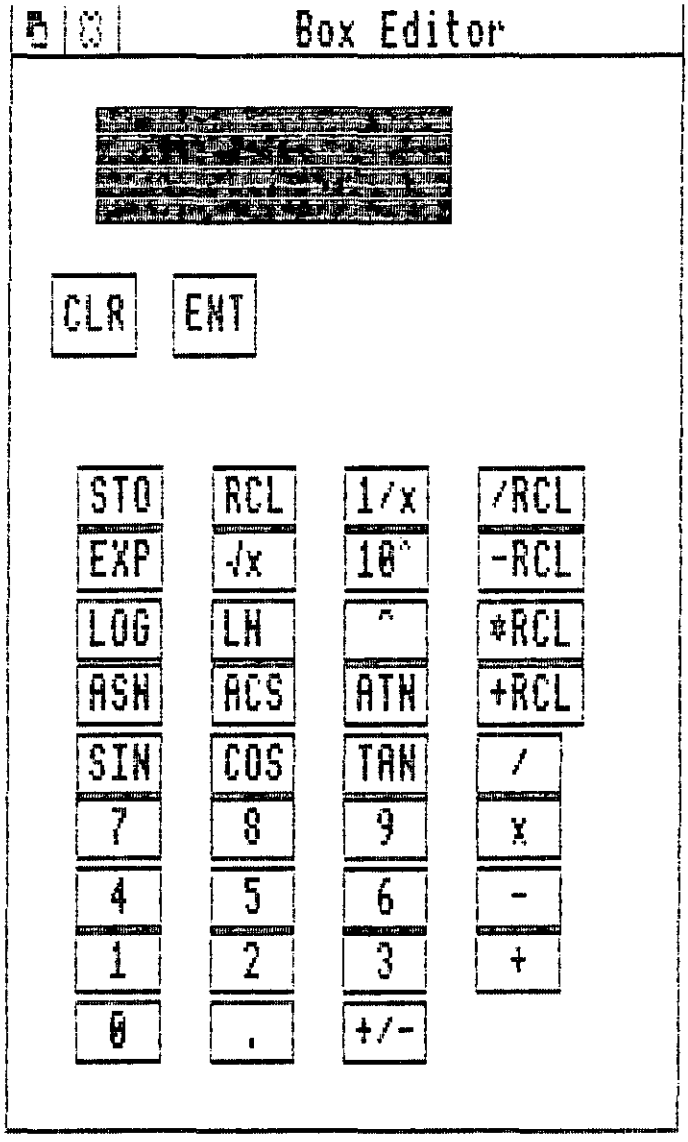


fig (2)
The boxeditor
(set at level 4)

Errors are generally caused by omitting numbers in operations which require them (e.g. +3.4 is ok, but + on its own is NOT), or by including expressions where only numbers are allowed - for instance, +2*6 is not allowed. See the appendix for error messages.

Note that if the contents of the box are too long to display in the flowchart box the right hand character in this box will be a right facing arrow. Clicking on a box will always transfer the contents into the boxeditor window, which is wider. Should this *still* fail to display the entire contents of the box, click ***SELECT*** inside the ***boxeditor screen*** (NOT just the boxeditor). This will place the caret (a red I-like character) inside the screen. The caret can be moved character-by-character with the cursor keys, and will scroll if necessary, and, incidentally, edited character-by-character using the cursor keys and the delete key. The full length actually stored should be ample for all user requirements.

There are two buttons immediately underneath the boxeditor screen display, labelled CLR and ENT :

CLR will clear the contents of the boxeditor window completely. It will not affect the current flowchart unless the contents are transferred over to the flowchart.

ENT transfers the boxeditor contents into the flowchart, using the currently selected method, after performing a syntax check. Pressing the RETURN key does the same job. Note that box contents are all "operators", being "done" on one input, and sending the output on to the next box in the chain.

You may edit the contents of the boxeditor using the mouse by clicking ***SELECT*** whilst the pointer is over the box itself. This will cause an I-like pointer, called the caret, to slip in between the characters of the box. You may use DELETE, and the left-right arrow

keys to move the caret about inside the string. Depressing a "printable" key will insert the corresponding letter at the caret.

In addition to simply clicking on boxeditor buttons, you may type the function in directly, but you will not obtain editing functions with the caret unless you can see the caret in the boxeditor box. Loading and saving is accomplished by selecting the "Filing" option from within the Edit menu, and selecting the options required at each stage. Whenever a RED entry in the menu is encountered, the user may type in his own choice, and this is the only way of loading flowcharts outside the directory specified by the teacher. You should note that this "writeable" option, together with renaming and deleting, is not always available to end users, and can be removed by the teacher.

6.4.10 Running a flowchart : General points

Given F1 is set up with some sort of flowchart along the lines of the one in tutorial 1 above, we may run it by clicking ***SELECT*** on the RUN icon on the control panel, or alternatively (and equivalently) by popping up the EDIT menu by pressing the middle mouse button, and then clicking ***SELECT*** on RUN in the EDIT menu. Use whichever method is more convenient.

Either way will pop up two windows over the previous two, the RUN info window, and the RUNRECORD window. RUN INFO is used to input numbers to be run through flowcharts.

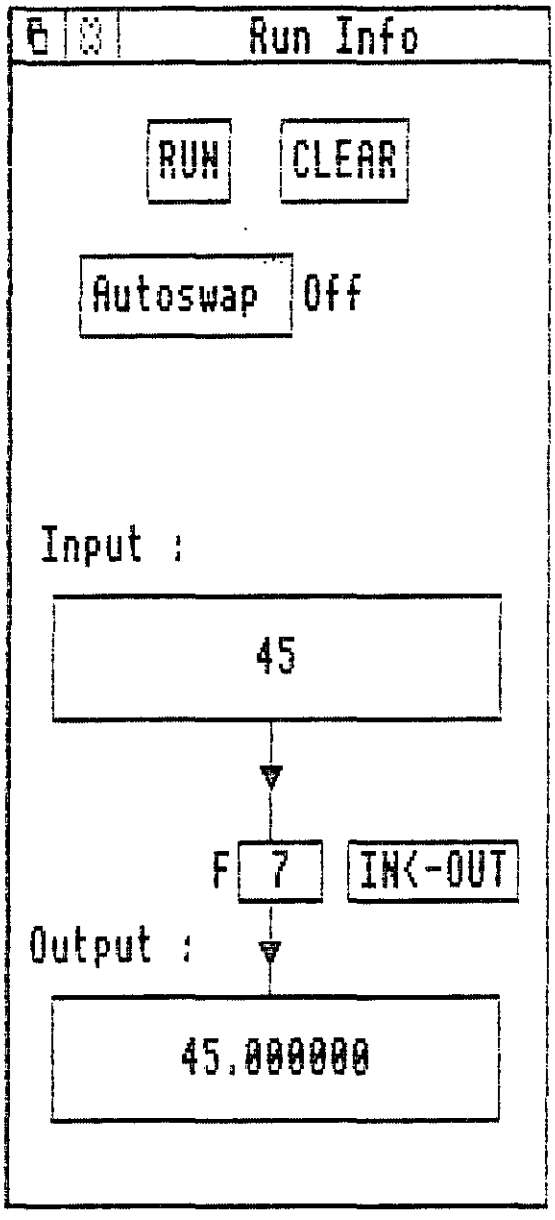


Fig (3)

The RUNINFO window

Position the mouse pointer anywhere within either of the two run windows, and type a number. The number should appear in the box labelled INPUT in the Run Info window. The input is error-trapped here, and the user may input only one *non-leading* decimal point, and may type a minus sign only in the initial position. Any attempt to input more than one decimal point, or a minus sign in other than the first place will be ignored. There is no need to press the return key when the number is complete. We may then run the flowchart by

pressing the red function key bearing the same number as the flowchart we wish to run. To run F1. tap the key F1! The machine responds by inserting the number of the flowchart just run in the Run info window in the F box. and the output in the output box. There are 30 flowcharts available. Flowcharts with numbers greater than 10 may be accessed with a combination of either the control or shift keys. Holding down the shift key and tapping a function key adds 10 to that keys number. holding control will add 20. Chart 11 can be accessed either by SHIFT-F1. or by F11. and similarly with chart 12.

In addition. after a run you should notice a record of that run appear in the Run Record window. along the lines of

5 ----)---- F1----)---- 9.150

Depending on the initial settings of the program. this may have been rounded to a certain number of significant figures or decimal places. This will be indicated after the result in the runrecord window. but not in the runentry window. The runrecord window can be slow to update. and the operations can be speeded up considerably by closing the runrecord window and working with the Runentry window only (runs will still be recorded and can be viewed by opening the window).

We may clear the input box for a second run in several different ways. Most obviously. typing a number immediately after a run will clear the input box automatically. Alternatively we may click ***SELECT*** on the icon labelled CLEAR towards the top of the run entry window. This clears all boxes in the run entry window (but not the run record). Equivalently. tapping the key labelled HOME will do the same job. Pressing the COPY key once transfers output to input.

and clears the output box. Depressing it a second time clears both boxes. but not the flowchart selected box (the small middle one). Thus pressing the COPY key twice has the same effect as HOME. Passing output to input can be useful if the user wants to explore convergence. or iterative methods (but see AUTOSWAP).

Clicking with ***SELECT*** on :

RUN will run the flowchart whose number appears in the small F box.

IN<-OUT has the same effect as pressing COPY once. transferring output to input

AUTOSWAP will toggle the AUTOSWAP state. It has two states. Clicking ***SELECT*** on the AUTOSWAP button turns it on if it was off. and vice versa. Turning AUTOSWAP ON re-routes the arrowing at the bottom of the window. Running with AUTOSWAP on will automatically put the output back into the input window every time. This mode of working is useful for testing convergence and so on. The output box is not used in this mode.

If the user does not modify the input immediately before a run with AUTOSWAP ON. the run record shows a series of 4 dashes where the input should be. This is intended to indicate that the input is taken from the previous output.

If the variables window is open it will be updated as the flowchart runs to show the contents of all the variables. You may.

of course. use the variables window to "set up" the input to a flowchart prior to a run. An empty input box is assumed to contain zero.

6.5 A more Detailed examination of FLOW

Note that in all the menus in FLOW, a red entry indicates that the user should type a name or value into the menu at that point. The caret will appear in the red box if the pointer is moved nearby.

6.5.1 The EDIT MENU

The first few items represent the "mode" of the boxeditor. One item will always be ticked (selected) and the others not. Select an item by clicking ***SELECT*** (the left hand mouse button with the cable emerging from the far side) on it.

APPEND always add the box to the end of the current flowchart, even if its contents have been picked up from an earlier box in the chart

OVERWRITE Overwrite the currently selected box. If none is currently selected, this option will overwrite the last box in the current flowchart, or flowchart 1 if none is selected

ADD AFTER Add the box after the current box. This is the same as append if the current box is the end. This is principally of use in adding extra boxes in the middle

ADD BEFORE Add the box before the current box. Some may prefer to do it this way! No additions to the chart will be made if the currently selected box is the header box.

You may set the mode by clicking ***SELECT*** on the appropriate menu item. The current mode is indicated by a tick in the menu.

DELETE BOX Will cause the currently selected box (coloured GREEN rather than RED, and with a heavy border) to be deleted from the flowchart. Succeeding boxes will move up, and the new currently selected box will be the one above the deleted one.

CLEAR Offers the user a submenu of options.

ALL CHARTS removes all charts from the flowchart window, and clears the graph window.

THIS CHART Clears ONLY the currently selected chart.

GRAPHS Clears the graph window, resets the scales and closes it if it was open.

RUN RECORD Clears the run record window of any previous results.

VARIABLES Clears all variables from memory. Note that changes and single deletions can be made from the variables menu.

RUN Pops up the runentry and runrecord windows.

GRAPH Pops up a window which allows the user to draw the cartesian plot of a function for a specified range. The y scale is automatically determined. Graphing is

documented in more detail below.

In order to graph successfully the user need only enter the chart number required, and the minimum and maximum x values required for the plot. These should be typed in on the bars provided in the graph sub-menu. Finally, click on Do it, and the graph will appear.

The autoscale options cope well with most functions, but functions containing singularities cannot be plotted successfully.

HELP

The help mechanism is initiated either by moving the pointer to the right over the arrow following help in the edit menu, or by clicking ***SELECT*** on the HELP icon on the control panel. Either method pops up a menu,"HELP.." which may consist of further sub-menus, or of final help items. It is up to the teacher to arrange these how she feels fit. The menu structure reflects the directory structure on the disk, and the files themselves are simply text files. Continue selecting by moving the pointer to the right until you come to an item with no right arrow. Click on this, and the help will be displayed in a resizable window. You may leave this on-screen, resize it, or whatever until you get rid of it by clicking on X. You can thus have help "on-screen" whilst you are working.

PREFERENCES

The only way to set preferences from within the program is via the preferences menu. from the edit menu. This has two sections. The upper section deals with angular measure. The ticked item is the currently selected one. Click ***SELECT*** on the angular measure you wish to use.

The lower section of the preferences menu is concerned with numeric accuracy. You may ask the program to round all output from flowcharts (in the runrecord window only) by setting a tick against *sig figs* or *dec places*. Click ***SELECT*** against whichever you prefer. You must then set the accuracy required by writing the number in the red section at the bottom of the menu. You do this by placing the pointer over the red section and using the caret, as before. The program will accept anything from 0 to 7 digits of accuracy.

FILING

Has four subsections:

RENAME allows a rename of any flow chart file (no others). Type in the new name first, and select the file you want to be renamed. You cannot rename any other files from within the program. This position can only be selected if flow$advanced is set on in the !run file, otherwise it is "shaded" and cannot be selected.

DELETE is self-evident. and again works only on flowchart files. Again. this facility can only be used if flow$advanced is set on in the !run file.

LOAD will allow you to load any FLOW chart file within the flowchart directory. You will be presented with a menu containing a list of flowchart items from which the user selects one by clicking ***SELECT*** on it. The entries in this menu are taken from the directory tree specified in the environment variable flow$flowdirname. The red box at the bottom will allow users to load via a pathname. and will scroll once the caret is placed in it by clicking ***SELECT*** on it.

SAVE saves all the flowcharts. Click ***SELECT*** on the red section to obtain the caret and type in the name at the bottom of the filing menu (the red section) and click ***SELECT*** on save. The user has no choice about where to save. Flowcharts can only be saved in the directory set in flow$flowdirname.

VARIABLES or MEMORY is considered on page 57. section 6.5.5

6.5.2 Flowchart Functions provided

The following list details the acceptable contents of flowchart boxes.

<num> represents a numeric parameter. floating point

<input> represents the input to the flowchart box

<output> represents the output from the flowchart box

<var> represents a named variable having a defined value. These can only be used if flow$namedvars is ON.

<mem> is a single unnamed variable. corresponding to the store on a calculator

¦ precedes any acceptable alternative

Box contents	Output is	Error conditions	MIN op Level
+<num>	<input>+<num>	none	1
-<num>	<input>-<num>	none	1
*<num> ¦ x <num>	<input>*<num>	none	1
/<num>	<input>/<num>	<num>=0	1
+<var>	<input>+<var>	<var> undefined namedvars not enabled	1
-<var>	<input>-<var>	<var> undefined namedvars not enabled	1
*<var> ¦ x<var>	<input>*<var>	<var> undefined namedvars not enabled	1
/<var>	<input>/<var>	<var>=0. <var> undefined	1
^<num>	<input>^<num>	<num> negative & non-integral <input>=0 & <num>=0	2
^<var>	<input>^<var>	<var> undefined <var> negative & non-integral <input>=0 &	2

		⟨var⟩=0	

(The next two functions must be typed in and are not available from the keyboard)

^1/⟨num⟩	('***') √⟨input⟩	⟨num⟩ = 0 & ⟨input⟩=0 ⟨num⟩ negative	2
^1/⟨var⟩	('***') √⟨input⟩	⟨var⟩ undefined ⟨var⟩ negative ⟨var⟩=0 & ⟨input⟩=0	2
√x ¦ SQRT	√⟨input⟩	⟨input⟩ < 0	2
1/x	1/⟨input⟩	⟨input⟩=0	2
+/- ¦ CHS	-⟨input⟩	none	1

Memory and variable related operations

+RCL	⟨input⟩+⟨mem⟩	⟨mem⟩ undefined	4
-RCL	⟨input⟩-⟨mem⟩	⟨mem⟩ undefined	4
*RCL ¦ xRCL	⟨input⟩*⟨mem⟩	⟨mem⟩ undefined	4
/RCL	⟨input⟩/⟨mem⟩	⟨mem⟩ undefined ⟨mem⟩ = 0	4
STO	⟨mem⟩=⟨input⟩, output=input		4
STO ⟨var⟩ ¦ = ⟨var⟩	⟨var⟩=⟨input⟩ output=input		4
RCL	output=⟨mem⟩ input irrelevant	⟨mem⟩ undefined	4
RCL ⟨var⟩ ¦ ⟨var⟩	output=⟨var⟩ input irrelevant	⟨var⟩ undefined	4

"Memory" operations which don't affect the running of the flowchart directly (not available from the boxeditor) :

These cannot be used on the anonymous memory (i.e. they MUST have a variable name after the operation), and the variable MUST exist before one of these operations is used :

STO+ ⟨var⟩	⟨var⟩=⟨var⟩+⟨input⟩ ⟨output⟩=⟨input⟩	⟨var⟩ undefined	4
STO* ⟨var⟩ ¦ STOx ⟨var⟩	⟨var⟩=⟨var⟩*⟨input⟩	⟨var⟩ undefined	4

	⟨output⟩=⟨input⟩		
STO/ ⟨var⟩	⟨var⟩=⟨var⟩/⟨input⟩ ⟨output⟩=⟨input⟩	⟨var⟩ undefined input=0	4
STO- ⟨var⟩	⟨var⟩=⟨var⟩-⟨input⟩ ⟨output⟩=⟨input⟩	⟨var⟩ undefined	4

Trigonometric functions

SIN	SIN(⟨input⟩) degrees/rads depends on preferences setting	Outside range which machine can cope with	3
COS	COS(⟨input⟩)	As SIN	3
TAN	TAN(⟨input⟩)	As SIN Also error "near" 90°	3
ASN	Arcsin(⟨input⟩)	⟨input⟩ not in range -1 to +1	3
ACS	Arccos(⟨input⟩)	⟨input⟩ not in range -1 to +1	3
ATN	Arctan(⟨input⟩)		3

Hyperbolic functions

(These functions are not available from the boxeditor keyboard and must be typed by the user)

SINH	sinh(⟨input⟩)		5
COSH	cosh(⟨input⟩)		5
TANH	tanh(⟨input⟩)		5

Other predefined functions

EXP	exp(⟨input⟩)		3
LOG	LOG_{10}(⟨input⟩)	⟨input⟩ ⟨= 0	3
LN	LOG_{e}(⟨input⟩)	⟨input⟩ ⟨= 0	3
10^{x}	Raise 10 to input power		3
F ⟨n⟩ ! F ⟨var⟩	Behaves like all of chart ⟨n⟩ Not on boxeditor integer part of ⟨var⟩ used if necessary.	⟨n⟩ = calling chartnumber	5

6.5.3 Working with named variables

In addition to the "anonymous" store, accessed by using STO, RCL, and the four operations +, -, x, / coupled with RCL, there are a number of named variables, or stores,which can be accessed by more advanced users, provided flow$namedvars is set to on. The syntax is an extension of the syntax used for anonymous stores. You may not use reserved words (i.e. the functions outlined above) as named variables, and this is trapped by the program. "x" is a slightly awkward case, since it is a synonym for multiplication when in the first character position. This affects only recall of x. This variable (and this one alone) cannot be recalled by simply inserting the variable name in a box because this will be interpreted as a times sign on its own. You MUST type "RCL x" to recall x. Variable names must begin with an alphabetic character, but after that, almost any character is legal. Leading spaces before variable names are ignored, and it is possible to type operations involving named variables without any spaces. FLOW recognises the end of the operation and the start of the variable name.

Assuming our variable is called "fred" (case IS significant) then we might choose to do the following :

=fred	stores the current value in variable fred, creating fred if necessary. (STO fred may be used if preferred)
fred	Destroys the output from the previous box, and replaces it with the value of fred. If fred is undefined then an error occurs. Any name other

than simply "x" is ok here..you cannot do this with "x", for reasons as noted above.

+fred Adds fred's value to the output from the previous box. Again, it is an error if fred does not exist.

-fred subtract fred's value from current value

xfred multiply current value by fred's value. "*fred" may be used if preferred

/fred divide current value by fred's value

6.5.4 Accessing named variables

Named variables can only be accessed by directly typing the name into the boxeditor window, but you may use a mixture of methods to construct the final desired box contents. For instance you can click on boxeditor buttons to begin with and finish off by typing variable names if required. You should bear in mind when choosing names that you cannot use the names of the built-in box operations.

Named variables are probably most useful in two cases. Firstly, where the algorithm expressed in this language requires the storage of intermediate results, and secondly when the user is attempting to build a flowchart to represent a function of more than one variable involved. Use another chart to initialise a value of a variable, and use the variable in the main flowchart. For instance, the following pair of charts will set up F2 to define a variable "slope", and F1 to multiply any input by slope :

F1 :	F2 :
x slope	= slope

Thus the user can investigate families of curves without redefining a flowchart for each one.

6.5.5 The Variables window

The variables window is opened by clicking on the variables entry in the Edit menu. Clicking on the entry will open the window until it is explicitly closed by the user by clicking ***SELECT*** on the cross. Moving the pointer over the right pointing arrow will open the window until the user moves the pointer off the window. Opening permanently is advisable for manipulating the variables, opening temporarily is acceptable for inspecting values at the end of a run. The variables window contains details of the variables currently known to the program, and also the contents of the anonymous store on the boxeditor calculator. Note that entering variable names into a flowchart will not create the variables. They will be created only when the flowchart containing the entries is run.

Clicking ***MENU*** whilst over the variables window will open a menu with four entries :

CLEAR ALL: Clears all the variables in the machine to uninitialised.

DELETE ONE : Moving the pointer over the right facing arrow will open a writeable box entitled "Name ?".

Enter the name of the variable you wish to delete here, and click ***SELECT*** or press the return key. The variable will be deleted from the list. It is an error to try to delete a variable which does not exist.

CREATE VAR : Allows the user to Create a variable and initialise its value. Move the pointer over the arrow, enter the variable name, move the pointer over the next arrow and enter a value. It is an error to create a variable which already exists.

ADJUST: Allows the user to change the value of an already existing variable. The method is exactly the same as CREATE VAR, except that it is an error to adjust a non-existent variable.

Note that none of the variable windows and menus are available if flow$namedvars environment variable is not set to "on". If this is set to "off" then the variables entry in the Menu will become shaded and unselectable. Also "variables" in the Clear menu changes to "memory", reflecting the fact that only the calculator "anonymous" memory is available in this case.

6.5.6 Memory Operations

The following operations do not affect the output at all, but complement the operations +RCL,-RCL,*RCL and /RCL.

For all of the memory operations the input is passed straight through, but there is a side-effect : the value of an already-existing variable is altered. This allows the summation of a series

or an infinite product to be investigated. and allows easier construction of a number of other algorithms.

STO+ fred	Adds the input to the box to fred. Output is not affected.
STO* fred	As previously. but multiplies.
STO- fred	Takes input from fred (note the order). output is unaffected.
STO/ fred	Divides fred by the input. Output is unaffected

6.5.7 The F <n> operation

This allows users to construct "subroutines" which can be called by other flowcharts. The box containing an F operation will behave exactly as if is was the entire flowchart F n inserted at that point. No error is produced if the flowchart is empty. since output=input is produced in these cases.

It is an error to call the flowchart currently running. since infinite recursion will result (there are no decision boxes in this version of the language) . and this mistake is thus trapped.

No local variables are provided. so the user should beware that side effects may result from using the calculator store and other named variables.

6.5.8 Graphing flowchart input and output

Selecting the graph item on the Edit menu will allow you to draw a graph of the output of a flowchart against its input. Begin by Moving the pointer to the right over the arrow in the GRAPH option in the Edit menu. Select the chart number (by default 1) : move the pointer on to the red section underneath "chart number" and type in the required chart number. Then set xmin and xmax similarly.

xmin and xmax are the limits for the range of input. If the user gets these the wrong way round the program will spot the error. The graph will be plotted in 100 steps from xmin to xmax. Finally, click anywhere on the menu with ***SELECT*** and the graph will be drawn automatically. Y axis scaling is computed by the program and there is no manual facility provided here, but note that this may cause difficulties if functions are drawn at, near or over discontinuities (beware of TAN, for example). Any errors encountered whilst evaluating the function to plot the curve will abort the graph plotting operation. The graph window may be left open. If you close the graph window then subsequently clicking on GRAPH, rather than moving the arrow over it to the right will reopen the graph window with its original contents intact. If you click on graph when no graphs have been plotted, an error , "No graph plotted" will be reported.

Note that FLOW remembers which graphs it has plotted and does not rework the tables of values unless the flowchart or ranges have been changed. This can cause some problems if a flowchart uses a named variable which is defined in another chart or set up externally by the user, since the change will not be noted by the graph plotting routines.If you use this method, clear the graphs yourself before plotting.

Clicking ***MENU*** whilst the pointer is over the graph window will open the graph window menu, which has 3 options:

INFO provides the user with details of the graphs plotted. It will tell the user how far apart the ticks are spaced, where the origin is, and how far apart the

dotted lines are. This will pop up and disappear if opened using the arrow. It will pop up and stay if the user clicks on INFO, and can then be moved around and left on - screen if desired. This window will also indicate which chart is associated with which line on the graph. The graphs are plotted in different colours on-screen, and the charts plotted are shown in the same colours within the info window.

CLEAR Will clear the internal record of any graphs plotted, and close the graph window, if it is open.

SUPERIMPOSE Will superimpose another curve over the already existing ones. Scales will not be altered, so the order in which the graphs are plotted makes a difference to the areas visible.

Plotting an empty flowchart will result in the line y=x. This is because the flowchart contains no operators, and therefore has no effect on any input. Evaluation, of course, produces the same result.

6.5.9 CONFIGURING FLOW

FLOW is configured upon loading by settings of various environment variables. This method is not particularly friendly, but it is relatively flexible. The environment variables are held conveniently in a file called !run which is how the operating system starts the application. The installed !run can be viewed as follows:

1) Open a directory viewer onto the FLOW disk, by clicking

SELECT on the disk icon on the control panel with the flow disk in the drive

2) Install !edit on the icon bar by double clicking *SELECT* on it quickly

3) Whilst holding down SHIFT, double click on !FLOW. This opens the application directory, allowing the user to examine the constituent files.

4) Drag the file called !run (with the *SELECT* button pressed) onto the edit icon on the icon bar and let go of *SELECT*.

It is probably convenient at this point to examine a model !run file :

----------START OF !RUN FILE ----------

```
|!run file for FLOW
|This version 0.2 16/7/89

rmensure fpemulator 2.70 rmload <Obey$dir>.modules.FPEmulator
rmensure fpemulator 2.70 error Cannot find FPEmulator
Iconsprites <Obey$Dir>.!sprites
wimpslot 400K

|              U S E R    A R E A
| User level configuration settings. All these optional
| and are set by the end user ...
| Modify only below this line to the note and not beyond


set flow$boxnumbers on
set flow$namedvars on
set flow$master on
set flow$advanced on
set flow$printing on
set flow$oplevel 4
set flow$helpdirname $.help
set flow$helpnamestyle *
set flow$flowdirname $.flows
set flow$degmode deg
set flow$editmode append
set flow$acctype sf
set flow$accuracy 4
set flow$autoload <flow$flowdirname>.trig
```

```
| E N D   OF   U S E R   A R E A
| End of user level configuration settings
| Do not modify below this line

run <Obey$Dir>.flow %*0
```

----------END OF !RUN FILE ----------

We may conveniently ignore any settings before the line labelled USER AREA. These lines merely ensure that all necessary resident software is installed before calling the program. The area in which appearance is considered is the user area itself. There are a number of environment variables which can be set. with effects as follows :

Variable	Value	Effect
flow$master	on	Allows "master level" ops
	<anything else ..or nothing>	(Has no effect in version 1.10) default is to disallow
flow$advanced	on	Allows advanced operations. e.g. file deletion, renaming
	<anything else>	disallow advanced ops default disallow
flow$printing	on	Allow user printing
	<anything else>	disallow user printing default disallow This has no effect in v1.10
flow$oplevel	1	Allows access to basic operators only, such as +,-,x and /
	2	
	3	
	4	
	5	Highest level. Allows access to all operators, including only those which can be hand typed and cannot be accessed from the boxeditor keyboard.

flow$oplevel has a direct effect on the appearance of the Box Editor window, as well as restricting legal operations. Here, for instance, is the appearance of a level 1 boxeditor window :

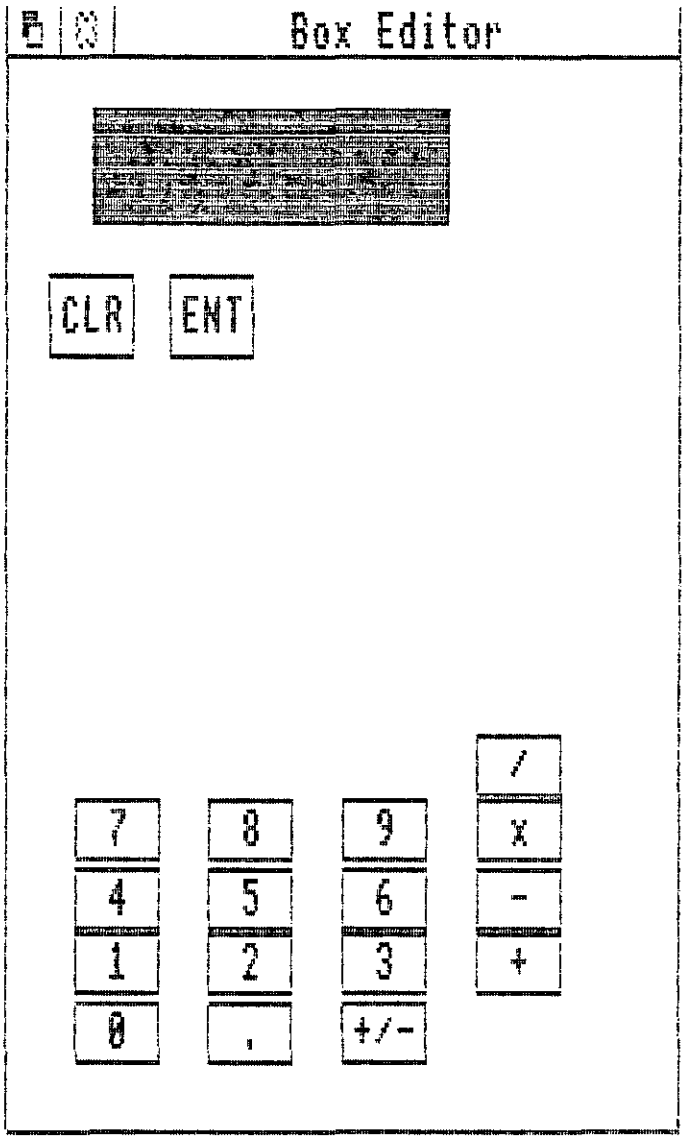


fig (4)
Level 1 Editor

(for comparison purposes, a level 4 boxeditor window was shown as figure 2 earlier)

flow$namedvars	on	Allows the user to use named variables in the flowcharts. Also allows access to the variables window and the variable editing facilities.
	off	Denies access to named variables, or the variable editing facilities.
flow$boxnumbers	on	Will print the box number of the box in the top left hand corner of the box. Can be useful if a chart does not

		work as expected since the error report will contain the box number at which the error was detected.
	off	Disables printing of box numbers. Makes the box look less cluttered. and less confusing for younger users.
flow$helpdirname	<a directory name>	The name of the directory where help files can be found. Default is "HELP". with no path
flow$helpnamestyle	<a dirname>	The style of the help files. This may be of use where other files are kept there. For instance. if all help files begin with "he" then set this variable to "he*" The default is "*". i.e. all files
flow$flowdirname	$.flows	The pathname of the flow chart files. Default is "FLOWS". with no path.
flow$degmode	deg rad	Degree mode is used Radian mode is used Default is degree mode
flow$editmode	append Overwrite before after	Edit mode is append Edit mode overwrite Default is append
flow$acctype	sf dp anything else	Accuracy is set to sf ..or dp ..or none
flow$accuracy	0 to 7	This number of places accuracy. Default 3. This is irrelevant if flow$acctype is not set to either sf or dp.
set flow$autoload	<a filename>	Load this upon entry to FLOW This should be a flow chart file. though not necessarily in the flow$flowdirname directory above. A full pathname should be used.

6.5.10 Editing the HELP files

As supplied these files are found "inside" the application directory. To access the files insert the disk into the disk drive and open the directory viewer. Hold down the SHIFT key and double click on the !FLOW application with ***SELECT***. This will open a second directory viewer of the inside of !FLOW. One entry (in the issue disk configuration) will be a blue folder entitled HELP. Double click on this with ***SELECT***. and the help area will become visible. You can load any help file by double clicking on its icon (blue pen over a page of text) with ***SELECT***. Using !EDIT to edit the file is documented in the standard ACORN documentation.

6.6 Error reporting

Errors are detected at two stages during processing a flowchart: the stage when the box is input. and then when a chart is run. When the box is input it is checked for syntax errors. and these will be reported to the user in a box which pops up in the middle of the screen. The user can do nothing until he has clicked on one of the options at the bottom of the screen. Some errors have only one option - "ok". which merely tell the user of a problem. The "ok"is meant to signify merely that the user acknowledges the problem. not that it *is* "ok". This use. though possibly eccentric. is quite standard on this machine and is not easily configurable. Others have two - "ok" and "cancel" which give the user an option. An example of where the user has a choice is obtained by clicking on the EXIT icon in the bottom left hand corner of the screen. Clicking on "ok" here will cause an exit. Clicking on "continue" will return the user to the program. Questions where options are involved are carefully phrased to make clear the effect of clicking on the various icons. Single option menu entries are more common.

6.6.1 Syntax errors

Syntax errors are detected when the user tries to enter a box which the editor does not recognise. such as an operator on its own. for example "+", or "x" : FLOW will tell the user that it should be followed by a number (or variable. if these are enabled). Entering a number with two decimal points will similarly cause an error. FLOW will do its best to point out what the problem is. and if it cannot will report that it does not recognise the string.

6.6.2 Run-time errors

Run-time errors are caused when the user runs a flowchart. The box syntax is ok, but the user has perhaps tried to divide by zero at some stage, or tried to find the angle whose sin is 5, or the square root of -4. All of these are run-time errors, and cannot be detected when the box is added to the chart.

When a run-time error is detected the box number in which it occurred is reported to the user, and the box is made the current box, so can be picked up from the flowchart window easily (green colour).

6.6.3 Operating system errors

Other errors can be caused by the operating system, but are reported to the user in exactly the same way, via the box in the middle of the screen. Most of these errors are to do with filing - perhaps a file is locked and cannot be overwritten, or perhaps the user has no read access to the file. A few errors may be encountered from other sources : for instance, opening a very large number of overlapping windows will cause an error. The computer may run out of memory given a very large flowchart, and this will be reported too.

6.6.4 Error messages

Error messages generated by FLOW, in alphabetical order.

Note that other errors can occur, but these are generated externally to FLOW from a variety of sources, and are consequently impossible to document.

0 to the power 0 not defined!
You tried to evaluate this, and is is not defined!

ASN or ACS can only have inputs in range -1 to 1
ASN or ACS cannot be used where the input was outside the range -1 to +1,because the functions are not defined outside this range

Accuracy <=7 please! setting to 7
Accuracy can only be set within these limits.

Accuracy >=1 please! setting to 1
Accuracy can only be set within these limits.

Box needs a number (or variable) after the operation
User tried an operation on its own, which isn't allowed.

Cannot delete header box
The user attempted to delete the box at the top of a flowchart, which is not allowed.

Cannot divide by 0
During the execution of a flowchart, a division by zero was attempted. The program will normally highlight the box, and will tell you the box number at which the error occurred.

Cannot open help file
Generally speaking the teacher has not told the program where to find the help files, or has not set public access.

Cannot take Square root of negative numbers
During the execution of the program, an attempt to find the square root of a negative number was attempted.The program will normally highlight the box, and will tell you the box number at which the error occurred.

Cannot take log of negative numbers
During the execution of the program, an attempt to find the log of a negative number was attempted.The program will normally highlight the box, and will tell you the box number at which the error occurred.

Cannot take reciprocal of 0
...The program will normally highlight the box, and will tell you the box number at which the error occurred.

Cannot call same flowchart
The user has a box in a flowchart which refers to the same flowchart.. Because there is no decision-taking machinery this would result in infinite recursion and is trapped by the program.

Cannot evaluate TAN near 90 degrees
The TAN function has a singularity near 90 degrees.

Cannot open file : insufficient access??
The access string of the file the user tried to open did not allow read access.

Cannot open file to save it : disk full, or exists and is locked??
Could not write over an existing file. Reasons are given

Floating point error : was FPE loaded
The program will not work correctly if the Floating Point Emulator is not loaded. It should be loaded in the supplied boot sequence.

FLOW doesn't recognise...
FLOW cannot make sense of the operation you have entered. You may have made a typing mistake, or you may have misunderstood the syntax

Illegal character in filename
The filename as typed contains an illegal character for the filing system in use.

Illegal instruction : program may have crashed
This error should not occur, but may be found if the program has not been set up or loaded via the supplied boot sequences, or cannot find the resources it requires.

Illegal value for environment variable
A configuration setting (i.e. a flow$*** variable) has been set to contain a value which FLOW cannot use.

Move pointer over the => symbol from left to right
You selected a menu item which has a right facing arrow after it..You should move the mouse pointer to the right over the arrow to open a sub-menu.

Must be a name,not a number
User probably typed a variable name when a number is required.

No legal current box
The user is in overtype mode but has no current box selected, or has the header box selected.

No filename given!
You MUST give a filename for this operation (probably save)

No graphs plotted
You cannot open the graph window by clicking on GRAPH in the main menu when no charts have been selected for plotting. To select a chart for plotting move the pointer over the right facing arrow to the right of GRAPH.

No more room for new variables!
The variable space is full, and no more may be created.

No such variable
An access of a non-existent variable was attempted.

Not a FLOWCHART file
The file the user tried to load is not a flowchart file, or has been corrupted such that FLOW does not recognise it.

Number must have an operation before it
..in a box definition, the number must have an operator before.

Only one +,-,x, / or ^ allowed
You Cannot enter an expression in a flowchart box, only one operation

(in the first column) is allowed.

Outside the range which this function can evaluate

In any computer the range of values for which functions can be calculated accurately is limited. The range has been exceeded in the given box, and an error has occurred as a result.

System ABORT : may not be able to recover
The system has been closed down in an unexpected way, for an undisclosed reason.

TAN is not defined for this input
You may not take the TAN of the input given. The program will normally highlight the box, and will tell you the box number at which the error occurred.

There must be something in the box
User tried to insert an empty box into the flowchart

The number is invalid
It probably contains an illegal character.

This memory is not defined
An attempt was made to alter a memory which does not exist.

This variable exists already
An attempt was made to create a variable which already exists.

Too many graphs
You may plot a maximum of 4 graphs simultaneously in this implementation

Two decimal points!
The user typed two decimal points in one number.

Unusual system error - may not recover!
This should not occur, but MAY indicate that you are running short of memory (for instance, by attempting to install a number of other programs at the same time as FLOW)

Unusual termination request from user
The program has terminated in an odd way.

Variable name too long- truncated
The variable name is too long for the program, and has been shortened.

7 Educational approach

FLOW is intended to be used as an exploratory tool for investigation of functions of one variable. It is intended to be used by the student initially as an aid to working through problems. The intention is to get the student to relate pressing keys on his calculator to operators in a FLOW chart diagram of a function. To this end several test worksheets have been devised to test operation at various levels of learning. I consider "A" level work first. Here the program is a convenient way of producing sequences and investigating convergence and oscillation. The Newton-Raphson method of producing an approximate value for a square root may, for instance, be simulated on FLOW without any difficulty, and this is presented after an investigation into chaotic behaviour, which may profitably be used with able mathematicians in the sixth form.

7.1 Investigation 1 : Chaos (Vivaldi (1989))

Chaotic behaviour is currently in vogue in mathematics. We shall use FLOW to investigate one function which gives some surprising results and exhibits chaotic behaviour.

Begin by setting FLOW up to iterate the function

$$x_{n+1} = \alpha x_n (1-x_n)$$ (the so-called "logistic map")

using a named variable, "alpha", for α.

Set α to 2, and iterate (set AUTOSWAP to ON) from $x_0=0.4$. The result should be a comfortable convergence to 0.5. Can you predict this behaviour by constructing an equation and solving it ?

Now iterate with the same starting value but with $\alpha = 1+\sqrt{5}$ (which of course needs to be evaluated before use, ≈ 3.236). Can you predict the obtained behaviour?

Now try $\alpha = 4$ with the same x_0. Can this behaviour be predicted? Can you even begin to appreciate what is happening?

The mathematics here is non-trivial.

If convergence occurs then the iteration should converge to a root of

$$x = \alpha x(1-x)$$

$$\Rightarrow \quad x(\alpha x - (\alpha-1)) = 0$$

$$\Rightarrow \quad x=0 \text{ or } \quad x = \frac{\alpha-1}{\alpha}$$

Which of course produces the correct result of 0.5 for $\alpha=2$, but there is no hint of problems to come at $\alpha=1+\sqrt{5}$, and a more sophisticated analysis of stability is required.

7.2 A look at Newton-Raphson

The standard Newton-Raphson method provides us with the broad hint that the sequence

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

converges to a root of the equation $f(x)=0$

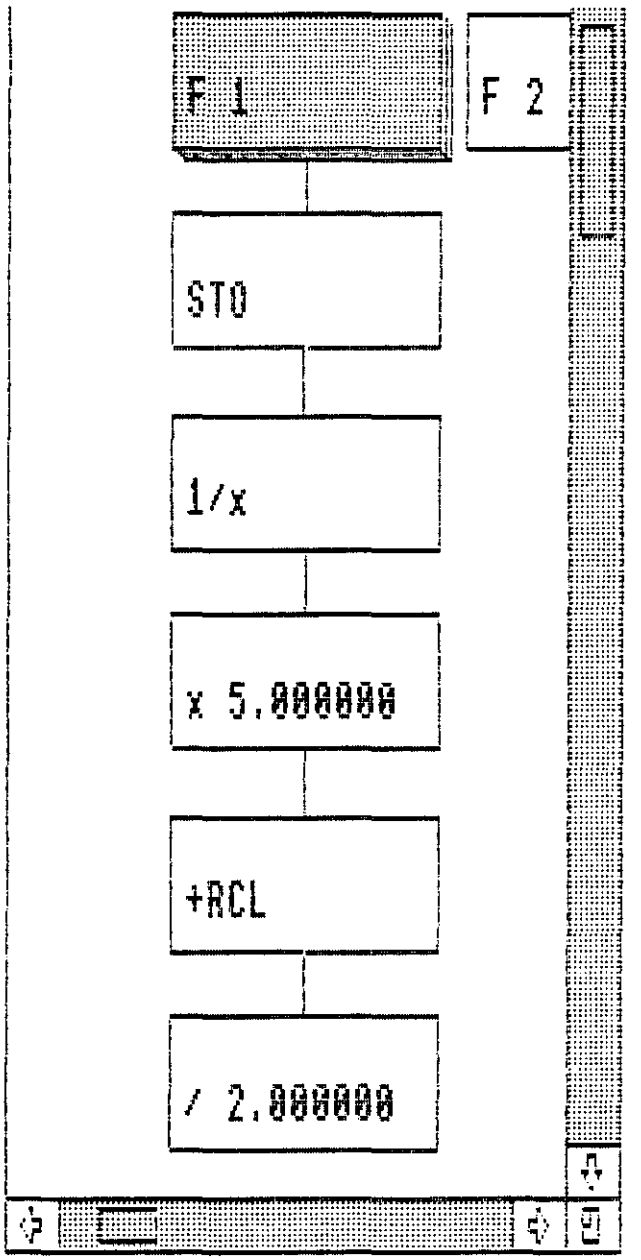
The student may then be encouraged to produce the right hand side for an equation of the form $x^2 - 5 = 0$. The result is, of course

$$x_{n+1} = \frac{(x_n + 5/x_n)}{2}$$

suggesting that the more generai result for $x^2 - a = 0$ is

$$x_{n+1} = \frac{(x_n + a/x_n)}{2}.$$

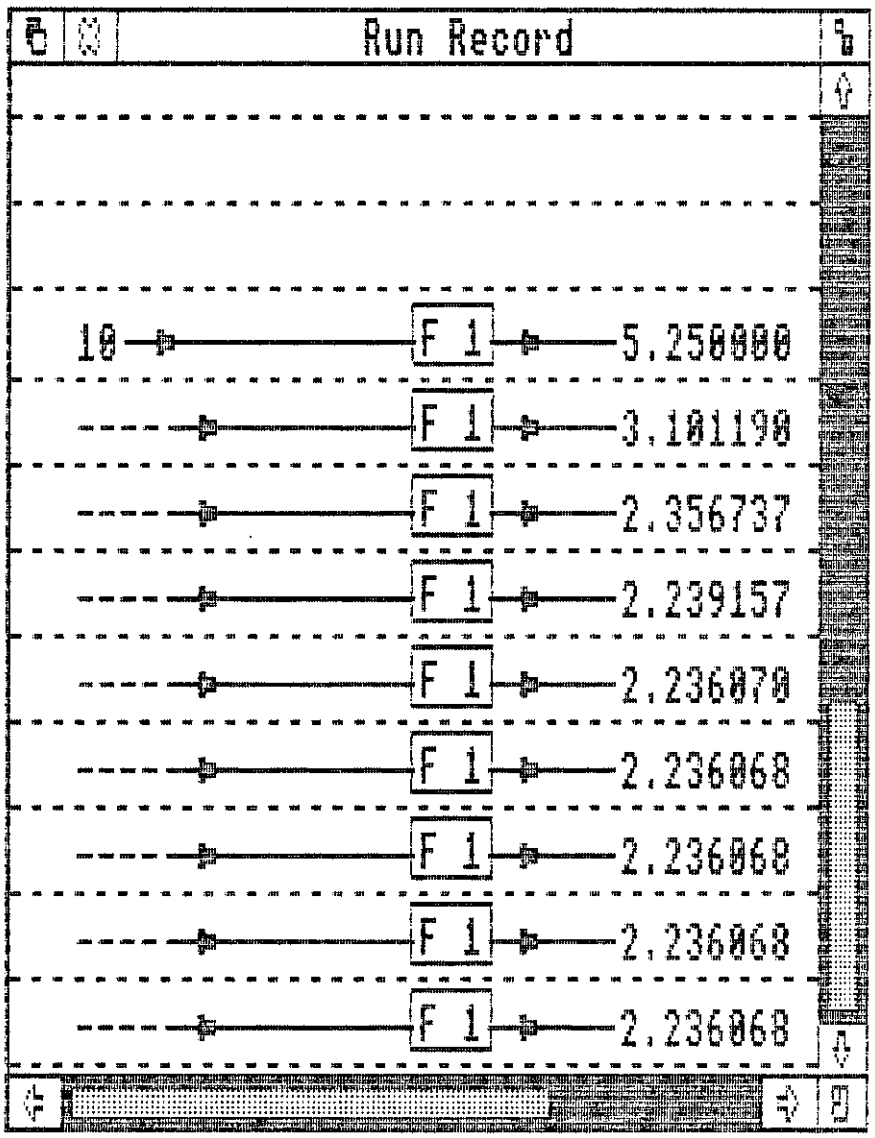
The student then meets the problem of evaluating this expression repeatedly. Here, FLOW can be introduced rapidly, with the eventual production of a chart along the lines of



Which directly reflects the key presses the student makes on the calculator in order to evaluate the expression (there is an implied "=" at the conclusion of each box in the chart, in accordance with SMP practice). He may then use this flowchart, with AUTOSWAP on, if necessary, to produce a sequence which clearly converges to the required result. The procedure is quick, suggestive

of the right links, and has been shown to be helpful in small informal classroom tests.

The effect of running this flowchart several times using the AUTOSWAP facility is illustrated below :



Note that the convergence is clearly illustrated, and that the starting point for the iteration is visible. The student can count quickly the number of iterations necessary to reach a given level of approximation.

By using a second flowchart and named variables the student can also simulate the general solution, involving a although FLOW does not cope quite as naturally with 2 variables.

7.3 Investigation 2 : Newton-Raphson and square roots

You know now that in general, if x_n is a root of the equation $f(x)=0$ then

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

is a "better" one.

Construct a Newton-Raphson iteration to find a solution to the equation

$$x^2 = 5$$

[Beware : this is *not* in the form f(x)=0 !!!]

When you have developed an iteration, turn the computer on and start up FLOW. Build a flowchart using the calculator/boxeditor which will perform this operation. Try it out on an initial guess of 5.

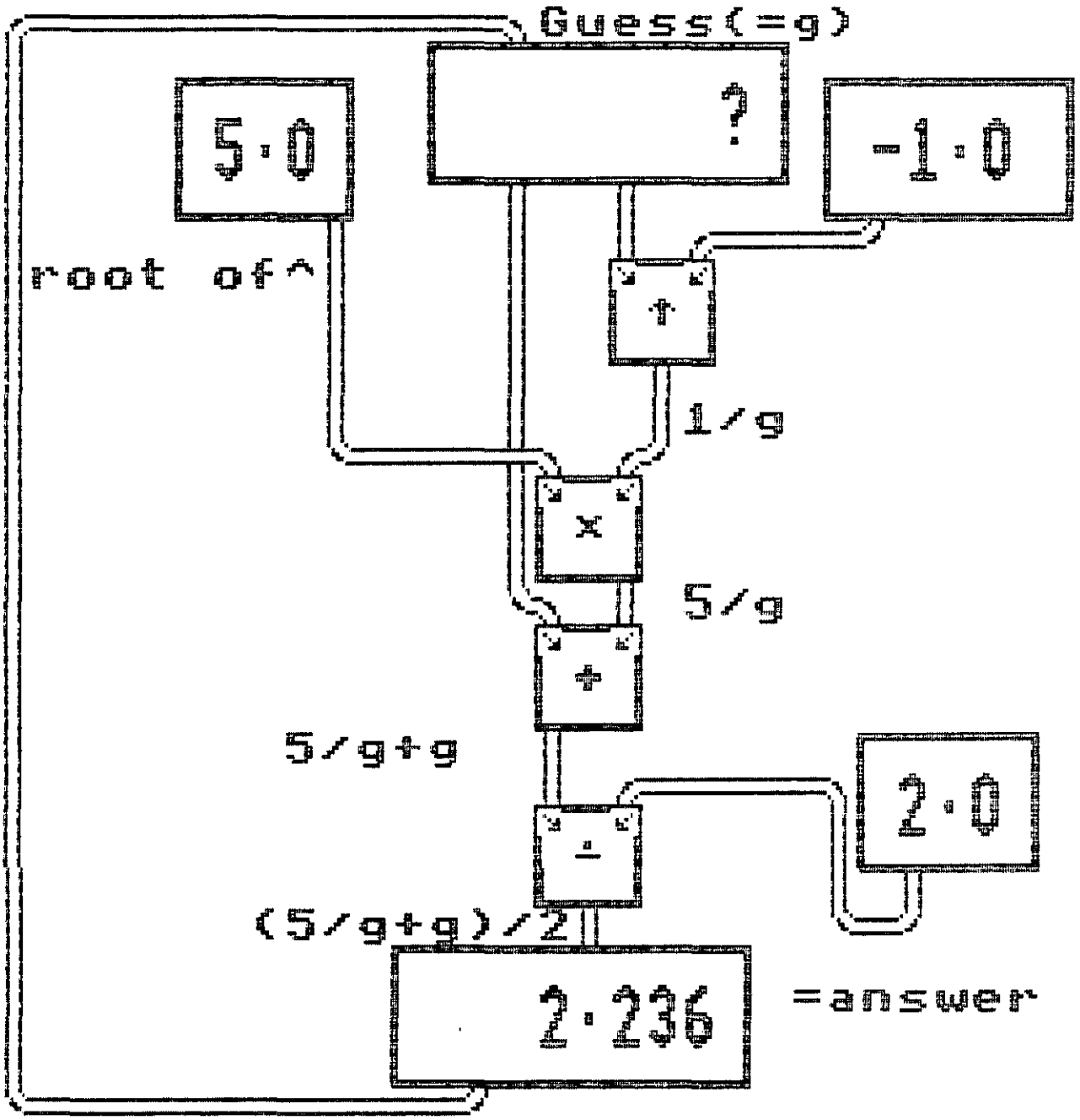
Try other initial guesses.

What effect does the initial guess have on convergence of the iteration? Does it affect the *rate* of convergence ?

Can you find an initial guess for which the chart will not converge?

Now modify your working to solve the equation $x^2 = a$ where a is a known number. Can you modify the flowchart in the computer to produce the iteration required ?

At this point a comparison with the NUMERATOR notation may be worth making. It is possible to simulate a Newton-Raphson iteration using the NUMERATOR package, but the limitations of the NUMERATOR notation become apparent. Here is a possible solution using Numerator :



It can be seen that the notation is more difficult to follow, although it is possible to construct problems which are much easier

to solve or depict using the tank notation (for instance, any problem which involves a function of more than one variable is tricky in FLOW notation, but may be easier to follow in NUMERATOR

There follows one investigation suitable for use with a competent GCSE year set :

7.4 Investigation 3 : Box volumes

An open box is made from a sheet of A4 card 297 x 210 mm by cutting squares of side x mm from its corners and then folding up the sides.

a) Explain why the volume V mm^3 of the box is given by the formula

$$V = x(210-x)(297-x)$$

b) Using FLOW, build a flowchart of this function. Why is it not easy to invert the flowchart ?

c) What are the largest and smallest values of x ?

d) Get FLOW to draw the function. Find the largest possible value of V. For what value of x does this volume occur? Would you have guessed this value ?

7.5 An investigation with 11 year olds

Functional ideas cannot be introduced early enough into the 11-16 curriculum. Here it is envisaged that the students would have been shown how to use FLOW, and will be familiar with the ideas involved in constructing flowcharts,either from sequences or from informal english descriptions. Here, the use envisaged is "emancipatory", in the sense that the program is merely carrying out computations for the student, although the use suggested in the worksheet is intended to formalise ideas in the student's mind.

For this exercise FLOW would be configured to run at level 1 in order that just a basic four-function calculator is available for use. A number of investigations can be constructed using only these simple operations, and a worksheet is included below to illustrate one idea. The iterative flowcharts are included on the accompanying disk, Filed as SEQINVS.

7.5.1 Investigation 4 : Sequences and series

You have spent some time discussing flowcharts and calculators. Now try to guess what flowcharts will produce these sequences :

1) 2,4,8,16,32,...

2) 5,7,9,11,13,...

3) 1,-1,1,-1,1,-1,1.....

4) 32,16,8,4,2,...

5) 3,7,15,31,63,...

Use FLOW to work out what the twentieth number in each sequence would be.

Can you work out what key presses on a calculator would take you from one number in the sequence to the next?

7.6 Investigation 5 : formula construction

(SMP New Book 4 part 2 page 26 q 7)

On a dry road the shortest braking distance d metres of my car is given approximately by dividing the square of the speed in km/h by 200.

1) Build, using FLOW, a flowchart to allow the user to input the speed in km/h and to output the braking distance in metres.

2) Find the shortest braking distances from speeds of 30 km/h and 60 km/h

3) Write down a formula connecting d and v.

4) On a nasty evening visibility is down to 45 m. What speeds would be safe according to the formula? Why is the formula unlikely to be useable here ?

5) Compare this formula and the one given in the highway code.

$$d = v + v^2/2$$

Using the graphing facilities of FLOW.

8 References

Alexander K. & Blanchard D (1985)
Educational Software : a creator's handbook — CET/MEP, 1985

Chandler D. (1984) Young Learners and the Microcomputer — OUP, 1984

Cronbach L.J. & Snow R.E.(1977)
Aptitudes & instructional methods, a handbook for research on interaction — Irvington,1977

De Greene K.B. (1970)
Systems and Psychology, in OSG, Systems Behaviour, qv

Evertsz R.,Hennessey S. and Devi R. (Sept 1988)
GADL : a Graphical Interface for Mental Arithmetic Algorithms, Institute of Educational Technology, Open University, September 1988 — CITE report no. 49,

Fox M.E. (1988) Theory and design For a Visual Calculator For Arithmetic, Institute of Educational Technology, Open University, March 1988 — CITE report no. 32,

Hartley J.R. (1981) Learner Initiatives in Computer Assisted Learning — in Howe & Ross,(qv)

Howe & Ross (1981)(ed) Microcomputers in Secondary Education : Issues and techniques — Kogan Page

Lansdell J (1988) "The Visual Mac", Review, page 39, Program Now vol 2 number 9, November 1988 — Intra Press

Open Systems Group (1972) (ed)
Systems Behaviour — Harper & Rowe

Pask G & Pangaro P. (1962) Entailment Meshes as Representations of Knowledge and Learning. — in Howe & Ross (qv)

School Mathematics Project (1982)
New Book 1 part 1
New book 1 part 2
New book 4 part 1 — Cambridge University Press

Shackel B.(1981)(ed) Man-computer interaction: Human factors Aspects of computers and people — Sijthoff & Noorhoff

Vivaldi F. (1989) An Experiment with Mathematics — New Scientist, 28 Oct 1989 p.46

Watson, Deryn (1987) Developing CAL : Computers in the Curriculum — Harper Ed series

Weber J.(1989) Simul & Stella — Personal Computer World, Nov 1989 p.186

9 Listings

9.1 Preliminary notes

The bulk of the program is written in C. Two small segments are written in Archimedes assembler language. There are several sections to the source code :

f_setup Is a setup module. Most of this is concerned with setting up window definitions, icons, sprites and so on. All of this module is executed early on.

f_help Provides the help system

f_main This module contains the function "main", which is the starting point for the suite. It contains general code for menus, window manipulations and so on. f_main also contains the poll loop which is the interface between the program and the controlling WIMP routine - all input and output passes through this route.

f_ops handles parsing of input, evaluation of flowcharts given input and so on

f_plot handles the graph plotting.

f_asm Is a small file containing a few lines of assembler, providing a few miscellaneous routines which would have been tiresome to write in C

In addition , there are a number of files which are used as an interface to the WIMP system :

w_shell Provides a means of controlling the WIMP calls to

user functions.

hourglass Allows the program to use the hourglass facility when busy.

Header files :

h_flow Contains constants, global declarations, macros and so on for the f_ series of files

h_winds Header for w_shell

9.2 Software used in program construction

The Acorn ANSI C compiler, release 2.01, latterly release 3 was used for all sections written in C.

RASM RISC Macro Assembler, various versions , written by myself, for the assembler code

Acorn Linker to tie all object modules together.

Acorn !paint utility to process screen images for this document

9.3 Program structure

There follows the results of a standard analysis of the program including all FLOW modules. The WIMP system takes control at poll(), line 64, followed by a complete listing of all the modules.

9.3.1 Structural analysis of FLOW

(including c.f_main,c.f_ops,c.f_setup and c.f_plot but not the w_* routines)

```
   1      MAIN [static in c.f_main]
   2              Hg_On
   3              initialise [static in c.f_main]
   4                      malloc
   5                      load_sprites [static in c.f_main]
   6                              make_sprite_area [static in c.f_main]
   7                                      malloc
   8                                      do_error
   9                                      exit
  10                              fextent
  11                              swix
  12                              wimp_error
  13                      set_taskname
  14                      strcpy
  15                      signal
```

```
16                      getenv
17                      atoi
18                      set_from_env [static in c.f_main]
19                              getenv
20                              strcmp
21                              do_error
22                      strcmp
23                      prefs_menu [static in c.f_main]
24                              unset_tick_item [static in c.f_main]
25                              set_tick_item [c.f_main]
26                      set_tick_item
27              mode
28              redefine_characters [static in c.f_main]
29                      vdu
30              set_palette [static in c.f_main]
31                      fopen
32                      vdu
33                      getc
34                      fclose
35              open_wimp
36              set_point_shape
37              wimp_error
38              setup_winds [c.f_setup]
39                      malloc
40                      usr_create_wind
41                      setup_file_menu [static in c.f_setup]
42                              malloc
43                              strcpy
44                      setup_prefs_menu [static in c.f_setup]
45                      setup_graph_menu [static in c.f_setup]
46                              malloc
47                              strcpy
48                      setup_super_menu [static in c.f_setup]
49                              malloc
50                              strcpy
51                      setup_vars_menu [static in c.f_setup]
52                              malloc
53                      setup_dir_menu_r [c.f_help]
54                              setup_dir_menu2
55                      setup_dir_menu [c.f_help]
56                              setup_dir_menu1
57                      strcpy
58                      setup_rename_menu [c.f_help]
59                              setup_dir_menu1
60                      define_calc_icons [static in c.f_setup]
61                              make_text_icon
62                              make_writeable_icon
63                      define_panel_icons [static in c.f_setup]
64                              make_text_icon
65                              allow_for
66                      define_runentry_icons [static in c.f_setup]
67                              malloc
68                              strcpy
69                              make_text_icon
70              initialise_functions [static in c.f_main]
71                      background_mouse_function
72                      mse_pressed_function
73                      window_redraw_function
```

```
74                      wind_enter_function
75                      wind_leave_function
76                      close_window_function
77                      key_pressed_function
78              initialise_screen [static in c.f_main]
79                      usr_open_wind
80                      move_mouse_to
81              clear_graphs [c.f_main]
82                      clear_graph_index [static in c.f_main]
83                      usr_close_wind
84              load_flowcharts [c.f_ops]
85                      fopen
86                      do_error
87                      fgetc
88                      fclose
89                      fgetword [static in c.f_ops]
90                              fgetc
91                      fgetdouble [static in c.f_ops]
92                              fgetc
93                      malloc
94                      free
95                      set_window_extent
96                      is_window_open
97                      usr_close_wind
98                      usr_open_wind
99              getenv
100             Hg_Off
101             system
102             poll

103     Edit_menu_decode [c.f_main]
104             item
105             unset_all_ticks [static in c.f_main]
106                     unset_tick_item [see line 24]
107             set_tick_item
108             update_boxrecs [c.f_main]
109                     force_update
110             delete_box [c.f_main]
111                     do_error
112                     free
113                     redraw_current_flowchart [static in c.f_main]
114                             redraw_flowchart_number [c.f_main]
115                                     xco_flowchart
116                                     force_update
117                     undo_plot_record [c.f_main]
118                             find_index
119                             is_window_open
120                             usr_close_wind
121             clear_menu [static in c.f_main]
122                     wipe_all_charts [c.f_main]
123                             wipe_chart [c.f_main]
124                                     free
125                                     undo_plot_record [see line 117]
126                                     redraw_flowchart_number [see line 114]
127                     set_window_extent
128                     usr_close_wind
129                     usr_open_wind
130                     clear_graphs [see line 61]
```

```
131                     wipe_chart (see line 123)
132                     input_focus_off
133                     clearrunrecord (c.f_ops)
134                             usr_update_wind
135                     clear_variables (c.f_ops)
136                             usr_update_wind
137             prefs_menu (see line 23)
138             file_menu (static in c.f_main)
139                     load_menu (c.f_main)
140                             strcpy
141                             sprintf
142                             strlen
143                             decode_menu
144                             wimp_error
145                             is_pathname (c.f_main)
146                                     isainum
147                                     do_error
148                             load_flowcharts (see line 84)
149                             clear_graphs (see line 81)
150                     rename_menu (c.f_main)
151                             strcpy
152                             sprintf
153                             decode_menu
154                             strlen
155                             is_pathname (see line 145)
156                             rename_file
157                             wimp_error
158                             do_error
159                     delete_menu (c.f_main)
160                             strcpy
161                             sprintf
162                             decode_menu
163                             strlen
164                             delete_file
165                             wimp_error
166                     save_flowcharts (c.f_ops)
167                             strlen
168                             do_error
169                             is_pathname (see line 145)
170                             strcpy
171                             sprintf
172                             exists (static in c.f_ops)
173                                     osfile
174                                     wimp_error
175                             wimp_decision (c.f_ops)
176                                     strcpy
177                                     wimp_error
178                             fopen
179                             fputc
180                             fputword (static in c.f_ops)
181                                     fputc
182                             length_of_flowchart (c.f_ops)
183                             fputdouble (static in c.f_ops)
184                                     fputc
185                             fclose
186                             settype
187                     setup_dir_menu (see line 55)
188                     setup_rename_menu (see line 58)
```

```
189             usr_open_wind
190             graph_menu [static in c.f_main]
191                     do_graph [c.f_main]
192                             clear_graphs [see line 81]
193                             do_error
194                             find_index [static in c.f_main]
195                             evaluate [c.f_ops]
196                                     run_time_error [static in c.f_ops]
197                                             sprintf
198                                             do_error
199                                             usr_update_scroll_wind
200                                             xco_flowchart
201                                             yco_boxnum
202                                     op_num_params
203                                     varval [static in c.f_ops]
204                                             var_exists [static in c.f_ops]
205                                                     strcmp
206                                             run_time_error [see line 196]
207                                     check_bounds [static in c.f_ops]
208                                             run_time_error [see line 196]
209                                             fabs
210                                     sqrt
211                                     pow
212                                     d_to_r [static in c.f_ops]
213                                     check_range [static in c.f_ops]
214                                             run_time_error [see line 196]
215                                     sin
216                                     cos
217                                     modf
218                                     fabs
219                                     tan
220                                     r_to_d [static in c.f_ops]
221                                     asin
222                                     acos
223                                     atan
224                                     log10
225                                     log
226                                     exp
227                                     store [c.f_ops]
228                                             strlen
229                                             do_error
230                                             var_exists [see line 204]
231                                             strncpy
232                                     sinh
233                                     cosh
234                                     tanh
235                                     evaluate <<< recursive >>>
236                             is_window_open
237                             usr_update_wind
238                             usr_open_wind
239                     atoi
240                     atof
241             do_error
242             help_menu [c.f_help]
243                     usr_close_wind
244                     sprintf
245                     decode_menu
246                     strlen
```

```
247                     strcmp
248                     is_a_dir
249                     free
250                     malloc
251                     fextent
252                     fopen
253                     do_error
254                     fread
255                     fclose
256                     set_window_extent
257                     usr_open_wind_scroll

258         fgetname [c.f_ops]
259                 fgetc

260         graph_window_closing [c.f_main]

261         key_on_runrecord [c.f_ops]
262                 key_on_runentry [c.f_ops]
263                     is_function_key [static in c.f_ops]
264                     sprintf
265                     update_ikey [static in c.f_ops]
266                             force_update
267                     strlen
268                     strcpy
269                     run_flowchart [static in c.f_ops]
270                             read_dbl [c.f_ops]
271                                     strlen
272                                     do_error
273                                     sscanf
274                             evaluate [see line 195]
275                             sscanf
276                             do_error
277                             strcpy
278                             strncpy
279                             print_sf
280                             strlen
281                             print_dp
282                             sprintf
283                             usr_update_wind
284                     strpbrk
285                     update_record [static in c.f_ops]
286                             sscanf
287                             strcpy
288                             sprintf
289                     usr_update_wind
290                     update_input [static in c.f_ops]
291                             force_update
292                     update_output [static in c.f_ops]
293                             force_update
294                     vdu
295                     atoi

296         mouse_elsewhere [c.f_main]
297                 usr_create_menu

298         mouse_enters_runentry [c.f_ops]
299                 input_focus_to
```

```
300     mouse_enters_runrecord [c.f_ops]
301             is_window_open
302             input_focus_to

303     mouse_leaves_runentry [c.f_ops]
304             input_focus_off

305     mouse_leaves_runrecord [c.f_ops]
306             input_focus_off

307     mouse_on_flowcharts [c.f_main]
308             move_current_box [static in c.f_main]
309                     adjust_to_origin
310                     flow_num
311                     min
312                     max
313                     box_num
314                     force_update
315                     xco_flowchart
316                     yco_boxnum
317                     setup_calc_screen [c.f_main]
318                             op_num_params
319                             strcpy
320                             text_of
321                             sprintf
322                     update_calc_screen [c.f_main]
323                             force_update
324                     update_boxrecs [see line 106]
325             usr_create_menu

326     mouse_on_help [c.f_help]
327             usr_create_menu

328     mouse_on_variable [c.f_ops]
329             usr_create_menu

330     mse_on_runentry [c.f_ops]
331             atoi
332             strlen
333             strcpy
334             run_flowchart [see line 269]
335             update_fkey [see line 265]
336             update_input [see line 290]
337             update_output [see line 292]
338             strpbrk
339             update_record [see line 265]
340             usr_update_wind
341             usr_create_menu

342     mse_on_runrecord [c.f_ops]
343             usr_create_menu

344     redraw_help [c.f_help]
345             move
346             put_sprite
347             gcol
348             intersect
```

```
349             puts
350             strlen

351     redraw_runentry [c.f_ops]
352             intersect
353             move
354             gcol
355             puts
356             drawby
357             arrowhead [static in c.f_ops]
358                     moveby
359                     plot
360             draw

361     redraw_runrecord [c.f_ops]
362             gcol
363             move
364             charsize [c.f_main]
365                     vdu
366             puts
367             strlen
368             moveby
369             printf
370             drawby
371             arrowhead [see line 357]
372             draw
373             gpos_y [static in c.f_ops]
374                     swix
375                     wimp_error
376             strncmp
377             plot

378     redraw_vartable [c.f_ops]
379             move
380             gcol
381             puts
382             printf

383     stripspaces [c.f_ops]

384     clearup [static in c.f_main]
385             do_error

386     dbp [static in c.f_ops]
387             floor
388             log10
389             fabs

390     ensure_box_is_visible [static in c.f_main]
391             get_wind_info
392             wimp_error
393             xco_flowchart
394             yco_boxnum
395             usr_update_scroll_wind

396     entering_boxeditor [static in c.f_main]
397             input_focus_to
```

```
398     entering_flows [static in c.f_main]
399             is_window_open
400             input_focus_to

401     frac [static in c.f_ops]
402             floor
403             ceil
404             fabs

405     graph_window_menu [static in c.f_main]
406             usr_open_wind
407             clear_graphs [see line 81]
408             graph_super_menu [c.f_main]
409                     do_graph [see line 191]
410                     atoi
411                     atof
412                     usr_update_wind

413     key_on_boxeditor [static in c.f_main]
414             strlen
415             syntax_ok [c.f_ops]
416                     malloc
417                     tokenise [static in c.f_ops]
418                             is_token [static in c.f_ops]
419                                     strncmp
420                                     strlen
421                             strlen
422                             until
423                             isdigit
424                             sscanf
425                             isalpha
426                             malloc
427                             do_error
428                             strcpy
429                     strcpy
430                     check_syntax [static in c.f_ops]
431                             op_num_params
432                             op_level
433                     construct_error_message [static in c.f_ops]
434                             strlen
435                             strcpy
436                             op_num_params
437                             lotsa_ops [static in c.f_ops]
438                                     is_op [static in c.f_ops]
439                             strcat
440                     sprintf
441                     do_error
442             update_current_flowchart [static in c.f_main]
443                     append_box [static in c.f_main]
444                             malloc
445                     overwrite_box [static in c.f_main]
446                             do_error
447                     insert_after_box [static in c.f_main]
448                             malloc
449                     insert_before_box [static in c.f_main]
450                             insert_after_box [see line 447]
451                     ensure_box_is_visible
452                     redraw_current_flowchart [see line 113]
```

```
453                     undo_plot_record [see line 117]
454                     update_max [static in c.f_main]
455                             set_window_extent
456                             usr_update_scroll_wind
457                             xco_flowchart
458                             yco_boxnum
459                     length_of_flowchart
460                     update_boxrecs [see line 106]
461                     setup_calc_screen [see line 317]
462                     reset_caret [static in c.f_main]
463                             swix
464                             wimp_error
465                     update_calc_screen [see line 322]
466             update_calc_screen [see line 322]

467     leaving_boxeditor [static in c.f_main]
468             input_focus_off

469     leaving_flows [static in c.f_main]
470             input_focus_off

471     [see line 1]

472     mouse_on_boxeditor [static in c.f_main]
473             oficon [static in c.f_main]
474             reset_caret [see line 462]
475             strlen
476             strcat
477             syntax_ok [see line 415]
478             update_current_flowchart [see line 442]
479             update_calc_screen [see line 322]
480             usr_create_menu

481     mouse_on_panel [static in c.f_main]
482             usr_create_menu
483             usr_open_wind
484             leave_program [c.f_main]
485                     wimp_decision [see line 175]
486                     close_wimp
487                     mode
488                     puts
489                     exit
490             system

491     mouse_on_plotwindow [static in c.f_main]
492             usr_create_menu

493     print_dp [static in c.f_ops]
494             sprintf

495     print_sf [static in c.f_ops]
496             sprintf
497             fabs
498             floor
499             log10
500             pow
501             atof
502             strcspn
```

```
503             strncpy

504     redraw_boxeditor [static in c.f_main]
505             move
506             gcol
507             printf
508             puts

509     redraw_flowcharts [static in c.f_main]
510             xco_flowchart
511             yco_boxnum
512             move
513             gcol
514             charsize [see line 364]
515             puts
516             intersect
517             drawbox [static in c.f_main]
518                     move
519                     gcol
520                     drawby
521                     moveby
522                     plot
523                     printf
524                     strcpy
525                     text_of
526                     op_num_params
527                     sprintf
528                     strlen
529                     strncat

530     redraw_plotbox [static in c.f_main]
531             graph_plot
532             super_plot

533     setup_dir_menu [static in c.f_help]
534             malloc
535             add_to_pathname [static in c.f_help]
536                     strcat
537             strncpy
538             menutexticon
539             strcpy
540             read_directory [static in c.f_help]
541                     swix
542                     wimp_error
543             wimp_error
544             sprintf
545             is_a_dir
546             strlen
547             menuwriticon
548             free
549             remove_last_from_path [static in c.f_help]
550                     strlen

551     vars_menu_decode [static in c.f_ops]
552             item
553             clear_variables [see line 135]
554             remove_variable [static in c.f_ops]
555                     var_exists [see line 204]
```

```
556                     usr_update_wind
557                     do_error
558             new_vars_menu [static in c.f_ops]
559                     item
560                     create_variable_menu [static in c.f_ops]
561                             sscanf
562                             strlen
563                             do_error
564                             var_exists [see line 204]
565                             item
566                             store [see line 227]
567                             usr_update_wind
```

9.3.2 Index to structural analysis:

Function name	Defined in...	Line in Above list
Edit_menu_decode	c.f_main	103
add_to_pathname	static in c.f_help	535
append_box	static in c.f_main	443
arrowhead	static in c.f_ops	357
charsize	c.f_main	364
check_bounds	static in c.f_ops	207
check_range	static in c.f_ops	213
check_syntax	static in c.f_ops	430
clear_graph_index	static in c.f_main	82
clear_graphs	c.f_main	81
clear_menu	static in c.f_main	121
clear_variables	c.f_ops	135
clearrunrecord	c.f_ops	133
clearup	static in c.f_main	384
construct_error_message	static in c.f_ops	433
create_variable_menu	static in c.f_ops	560
d_to_r	static in c.f_ops	212
dbp	static in c.f_ops	386
define_calc_icons	static in c.f_setup	60
define_panel_icons	static in c.f_setup	63
define_runentry_icons	static in c.f_setup	66
delete_box	c.f_main	110
delete_menu	c.f_main	159
do_graph	c.f_main	191
drawbox	static in c.f_main	517
ensure_box_is_visible	static in c.f_main	390
entering_boxeditor	static in c.f_main	396
entering_flows	static in c.f_main	398
evaluate	c.f_ops	195
exists	static in c.f_ops	172
fgetdouble	static in c.f_ops	91
fgetname	c.f_ops	258
fgetword	static in c.f_ops	69
file_menu	static in c.f_main	138
find_index	static in c.f_main	194
fputdouble	static in c.f_ops	183
fputword	static in c.f_ops	180
frac	static in c.f_ops	401
gpos_y	static in c.f_ops	373
graph_menu	static in c.f_main	190
graph_super_menu	c.f_main	408
graph_window_closing	c.f_main	260
graph_window_menu	static in c.f_main	405
help_menu	c.f_help	242
initialise	static in c.f_main	3
initialise_functions	static in c.f_main	70
initialise_screen	static in c.f_main	78
insert_after_box	static in c.f_main	447
insert_before_box	static in c.f_main	449
is_function_key	static in c.f_ops	263
is_op	static in c.f_ops	438
is_pathname	c.f_main	145
is_token	static in c.f_ops	416
key_on_boxeditor	static in c.f_main	413

key_on_runentry	c.f_ops	262
key_on_runrecord	c.f_ops	261
leave_program	c.f_main	484
leaving_boxeditor	static in c.f_main	467
leaving_flows	static in c.f_main	469
length_of_flowchart	c.f_ops	182
load_flowcharts	c.f_ops	64
load_menu	c.f_main	139
load_sprites	static in c.f_main	5
lotsa_ops	static in c.f_ops	437
main	static in c.f_main	1
make_sprite_area	static in c.f_main	6
mouse_elsewhere	c.f_main	296
mouse_enters_runentry	c.f_ops	298
mouse_enters_runrecord	c.f_ops	300
mouse_leaves_runentry	c.f_ops	303
mouse_leaves_runrecord	c.f_ops	305
mouse_on_boxeditor	static in c.f_main	472
mouse_on_flowcharts	c.f_main	307
mouse_on_help	c.f_help	326
mouse_on_panel	static in c.f_main	481
mouse_on_plotwindow	static in c.f_main	491
mouse_on_vartable	c.f_ops	328
move_current_box	static in c.f_main	308
mse_on_runentry	c.f_ops	330
mse_on_runrecord	c.f_ops	342
new_vars_menu	static in c.f_ops	556
oficon	static in c.f_main	473
overwrite_box	static in c.f_main	445
prefs_menu	static in c.f_main	23
print_dp	static in c.f_ops	493
print_sf	static in c.f_ops	495
r_to_d	static in c.f_ops	220
read_dbl	c.f_ops	270
read_directory	static in c.f_help	540
redefine_characters	static in c.f_main	28
redraw_boxeditor	static in c.f_main	504
redraw_current_flowchart	static in c.f_main	113
redraw_flowchart_number	c.f_main	114
redraw_flowcharts	static in c.f_main	509
redraw_help	c.f_help	344
redraw_plotbox	static in c.f_main	530
redraw_runentry	c.f_ops	351
redraw_runrecord	c.f_ops	361
redraw_vartable	c.f_ops	378
remove_last_from_path	static in c.f_help	549
remove_variable	static in c.f_ops	554
rename_menu	c.f_main	150
reset_caret	static in c.f_main	462
run_flowchart	static in c.f_ops	269
run_time_error	static in c.f_ops	196
save_flowcharts	c.f_ops	166
set_from_env	static in c.f_main	18
set_palette	static in c.f_main	30
set_tick_item	c.f_main	25
setup_calc_screen	c.f_main	317
setup_dir_menu	c.f_help	55
setup_dir_menu1	static in c.f_help	533

setup_dir_menu2	static in c.f_help	
setup_dir_menu_r	c.f_help	53
setup_file_menu	static in c.f_setup	41
setup_graph_menu	static in c.f_setup	45
setup_prefs_menu	static in c.f_setup	44
setup_rename_menu	c.f_help	56
setup_super_menu	static in c.f_setup	46
setup_vars_menu	static in c.f_setup	51
setup_winds	c.f_setup	38
store	c.f_ops	227
stripspaces	c.f_ops	383
syntax_ok	c.f_ops	415
tokenise	static in c.f_ops	417
undo_plot_record	c.f_main	117
unset_all_ticks	static in c.f_main	105
unset_tick_item	static in c.f_main	24
update_boxrecs	c.f_main	108
update_calc_screen	c.f_main	322
update_current_flowchart	static in c.f_main	442
update_fkey	static in c.f_ops	265
update_input	static in c.f_ops	290
update_max	static in c.f_main	454
update_output	static in c.f_ops	292
update_record	static in c.f_ops	285
var_exists	static in c.f_ops	204
vars_menu_decode	static in c.f_ops	551
varval	static in c.f_ops	203
wimp_decision	c.f_ops	175
wipe_all_charts	c.f_main	122
wipe_chart	c.f_main	123

9.4.1 H.FLOW

```
/* >h.flow


 @    @          @@@@@@  @        @@@@   @    @
 @    @          @       @       @    @  @    @
 @@@@@@          @@@@@   @       @    @  @    @
 @    @    @@@   @       @       @    @  @ @@ @
 @    @    @@@   @       @       @    @  @@  @@
 @    @    @@@   @       @@@@@@   @@@@   @    @

*/

/*********************************************************************
   This File contains global constant and variable declarations for the
   collection of f_ files comprising the FLOW source.
   Note that h.flow itself includes h.winds, which is the global definitions
   for the winds (WIMP interface) file

        This verion 1.00 24/8/89

*********************************************************************/
#define ARTHUR_OLD_NAMES
#include <stdlib.h>
#include <math.h>
#include <ctype.h>
#include <string.h>
#include <stdio.h>
#include <arthur.h>
#include <limits.h>
#include <float.h>
#include "winds.h"
#include "windasm.h"
#include "h.hourglass"

/* Variables.... */

#define NUM_VARS 20

/* defines for variable status ... */
/* not initialised : */

#define V_UNINIT 0

/* initialised to some value by  user : */

#define V_OK     1



/*******General key definitions ********/

#define SCREENLENGTH 15
#define DELETE_KEY 0x7F
#define RETURN_KEY 0x0D
#define COPY_KEY   0x18B
#define HOME_KEY   0x1E
```

```
#define END_OF_STRING (char)0

#define D_EPSILON 1E-9                          /* kludge for floating point equality */


/****** Ensure that things which ought to be defined actually are *****/

#ifndef NULL
#define NULL (void*)0
#endif

#ifndef TRUE
#define TRUE 1
#endif

#ifndef FALSE
#define FALSE 0
#endif

#define until(x) while(!(x))
#define forever while(TRUE)

/* Constants for flow  sub-programs....
   ...anything prefaced with D_ is a default value, stored in variable of
   same name with the D_ missing. This transfer is effected in f_main.
*/
/* default contents of runInfo boxes...*/

#define DFLT_NUMINPUT  "                    "
#define DFLT_NUMOUTPUT "                    "
#define DFLT_RUNFLOWNUM " 1 "


#define D_HELPDIRNAME "HELP"                                    /* Path to help files */
#define D_HELPNAMESTYLE "*"                                  /* Style for help file names */
#define HELP_WIDTH   1280                                                /* pixels */
#define HELP_HEIGHT  800

#define D_FLOWDIRNAME "FLOWS"                                /* path to flowcharts saved */
#define OS_GBPB_READENTRIES 8                      /* code to read entries from current dir */
#define MENU_GAP 5                                           /* gap between menu entries */
#define MAX_FLOWS 30                              /* Maximum number of flowcharts available */
#define NUM_OPS 33                                      /*number of operations allowed, */
#define NUM_TOKENS 12                                   /*Number of distinct tokens */
#define FILE_VERS_NUM 1                                /* version number of file type */
#define NUM_GRAPH_POINTS 100                     /* use 100 points in graph plot tables */

/* boxeditor geometry */

#define CALC_ROWS 10
#define CALC_COLS 4
#define W_BOXEDIT_HEIGHT 800
#define W_BOXEDIT_WIDTH 500

/* variable table, size ... */

#define W_VARBOX_HEIGHT   (NUM_VARS*CHAR_HEIGHT+50)
#define W_VARBOX_WIDTH    (30*CHAR_WIDTH)
```

```
/* flowchar windows, initial size... */

#define W_FLOWBOX_HEIGHT  300                    /*initial height pixels of flowchart window*/
#define W_FLOWBOX_WIDTH   300*(MAX_FLOWS+1)                              /* width in pixels, same */

#define W_RUNENTRY_HEIGHT  850
#define W_RUNENTRY_WIDTH   400

/* plotwindow sizes */

#define W_PLOTBOX_WIDTH 1100
#define W_PLOTBOX_HEIGHT 900

/* system geometry */

#define CHAR_HEIGHT     32
#define CHAR_WIDTH      16

/* Runentry box geometry */

#define IN_OUT_X 30
#define IN_Y 360
#define OUT_Y 60
#define IN_OUT_HT 96
#define F_X (IN_OUT_X+9*CHAR_WIDTH)-3
#define F_Y ((IN_Y+OUT_Y+IN_OUT_HT)/2-32)

/* geometry of the graphical boxes for flowcharts ... */

#define LINE_LENGTH       50
#define BOX_HEIGHT        100                   /* Height of one flowchart box in pixels */
#define BOX_WIDTH         200

#define TOT_BOX_HT  (LINE_LENGTH+BOX_HEIGHT)

/* when recording output from runs... */


#define MAX_REC_COLS 40                                  /* maximum size of rec_line,used to */
#define MAX_REC_LINES (30)                                                    /* record output */
#define W_RUNRECORD_HEIGHT  (MAX_REC_LINES*CHAR_HEIGHT*2+200)
                                                       /* Pixel height of run record window */
#define W_RUNRECORD_WIDTH  850


/* flags returned by syntax checking routines... */


#define NO_GOOD    0
#define NOT_PERMITTED -1


/**** TOKENS, produced when input line is crunched *****/


#define TOK_PLUS      0x80
#define TOK_MINUS     0x81
```

```
#define TOK_TIMES   0x82
#define TOK_DIVIDE    0x83

#define TOK_RECIP     0x84
#define TOK_SQRT      0x85
#define TOK_POW       0x86

#define TOK_SIN        0x87
#define TOK_COS        0x88
#define TOK_TAN        0x89
#define TOK_ASN        0x8A
#define TOK_ACS        0x8B
#define TOK_ATN        0x8C

#define TOK_LOG        0x8D
#define TOK_LN         0x8E
#define TOK_EXP        0x8F
#define TOK_TENP       0x90


#define TOK_STO        0x91
#define TOK_RCL        0x92

#define TOK_PLUSRCL    0x93
#define TOK_MINUSRCL   0x94
#define TOK_TIMESRCL   0x95
#define TOK_DIVIDERCL  0x96
#define TOK_CHS        0x97

#define TOK_SINH       0x98
#define TOK_COSH       0x99
#define TOK_TANH       0x9A

#define TOK_SUBFLOW    0x9B

#define TOK_ROOT       0x9C
#define TOK_PSTO       0x9D
#define TOK_TSTO       0x9E
#define TOK_MSTO       0x9F
#define TOK_DSTO       0xA0


#define TOK_NUMBER 0xB0
#define TOK_VAR    0xB1
#define TOK_ENTER  0xB2



/****WIMP  keycodes for function keys...they always return these
       values from within the WIMP
*/

#define FN_0     0x180
#define FN_1     0x181
#define FN_2     0x182
#define FN_3     0x183
#define FN_4     0x184
```

```
#define FN_5      0x185
#define FN_6      0x186
#define FN_7      0x187
#define FN_8      0x188
#define FN_9      0x189

#define FN_10     0x1CA
#define FN_11     0x1CB
#define FN_12     0x1CC

#define S_FN_0      0x190
#define S_FN_1      0x191
#define S_FN_2      0x192
#define S_FN_3      0x193
#define S_FN_4      0x194
#define S_FN_5      0x195
#define S_FN_6      0x196
#define S_FN_7      0x197
#define S_FN_8      0x198
#define S_FN_9      0x199

#define S_FN_10     0x1DA
#define S_FN_11     0x1DB
#define S_FN_12     0x1DC

#define C_FN_0      0x1A0
#define C_FN_1      0x1A1
#define C_FN_2      0x1A2
#define C_FN_3      0x1A3
#define C_FN_4      0x1A4
#define C_FN_5      0x1A5
#define C_FN_6      0x1A6
#define C_FN_7      0x1A7
#define C_FN_8      0x1A8
#define C_FN_9      0x1A9

#define C_FN_10     0x1EA
#define C_FN_11     0x1EB
#define C_FN_12     0x1EC


/* and add to these ... */

#define CONTROL  0x20
#define SHIFT    0x10

/* plotting ... */
/* maximum number of graphs allowed */

#define MAXGRAPHS 4



/*************************************************************************/
/* MENUs :  Miscellany                                                   */
/*************************************************************************/
```

```
#define NO_SUB_MENU (menu_block*)-1

#define menutexticon(col1,col2)         TEXT+(col1<<24)+(col2<<28)+FILLED

#define menuwriticon(col1,col2)         TEXT+INDIRECT+\
                                        (col1<<24)+(col2<<28)+FILLED


/*********************************************************************/
/* MENU decoding functions : positions in the menu  of entries...  */
/*********************************************************************/

/* Menu items                    */

/* 1) Filing menu                */


#define FM_RENAME 0
#define FM_DELETE 1
#define FM_LOAD 2
#define FM_SAVE 3
#define FM_SAVE_FILENAME 4


/* 2) Edit menu                  */

#define EM_APPEND 0
#define EM_OVER 1
#define EM_AFTER 2
#define EM_BEFORE 3
#define EM_DELETEBOX 4
#define EM_CLEAR 5

#define EM_RUN 6
#define EM_GRAPH 7
#define EM_HELP 8
#define EM_PREFS 9
#define EM_FILE 10
#define EM_VARS 11


/* 3) graph menu                 */

#define GM_FNUM 1
#define GM_XMIN 3
#define GM_XMAX 5
#define GM_GO   6

/* 4) graph window menu          */

#define GWM_INFO 0
#define GWM_CLEAR 1
#define GWM_SUPERIMP 2

/* 5) Graph superimpose menu */

#define GSM_NUM 1
```

```
#define GSM_GO  2

/* 6) Prefs menu              */

#define PM_DEGS 0
#define PM_RADS 1

#define PM_SIG_FIGS 2
#define PM_DEC_PLACES 3
#define PM_NO_ACCURACY_SET 4

#define PM_ACCURACY 5

/* 7) Clear menu entries   */

#define CM_ALL 0
#define CM_CURRENT 1
#define CM_GRAPHS 2
#define CM_RUNRECORD 3
#define CM_VARS 4

/* 8) vars menu */

#define VM_CLEAR 0
#define VM_DELETE 1
#define VM_CREATE 2
#define VM_ADJUST 3


/* possible values for accuracy_type ...*/


#define NONE 0
#define DEC_PLACES 1
#define SIG_FIGS 2


#define sprintfNULL(buf,fmt,string) buffer[sprintf(buf,fmt,string)]=NULL


/* x coordinate of flowchart number num */

#define xco_flowchart(num)  (BOX_WIDTH+(BOX_WIDTH+20)*(num))

/* y coordinate of box number num, top line of the box */

#define yco_boxnum(num) ((FLOW_HEIGHT-100)-(BOX_HEIGHT+LINE_LENGTH)*num)

/* input pixel position relative to window origin, get box output : */

#define flow_num(x) (int)floor((float)(x-BOX_WIDTH)/(float)(BOX_WIDTH+20)+0.5)
#define box_num(y)  \
   (int)floor((float)(FLOW_HEIGHT-100-y)/(float)(BOX_HEIGHT+LINE_LENGTH))

#define max(a,b) ((a>b)?a:b)
#define min(a,b) ((a<b)?a:b)
```

```
/*********************************/
/* UNIV BLOCK Allocations for various
   routines.
   UNIV_BLOCK is the block used by POLL
   to pass information to various functions
   (The functions always declare the univ block as *block)
       univ_block is defined in the ARTHUR system library,
        and is used for passing data from WIMP to routines
*/
/*********************************/



/* for mouse button presses ....  */

#define mousex    (block->a[0])
#define mousey    (block->a[1])
#define mouseb    (block->a[2])
#define mouseicon (block->a[4])


/*for key presses ...... */

#define keyicon    (block->a[1])
#define keycrtxoff (block->a[2])
#define keycrtyoff (block->a[3])
#define caretht    (block->a[4])
#define crtindex   (block->a[5])
#define keycode    (block->a[6])


/* for menu select ............. */

#define item(i)         (block->a[i])



/* redraw block bits */
/* this block is a redraw_block */

#define WA_XMIN  (block->x0)
#define WA_YMIN  (block->y0)
#define WA_XMAX  (block->x1)
#define WA_YMAX  (block->y1)

/*********************************************************************/
/*** Error codes encountered during the program  *******************/
/*********************************************************************/

#define NOT_FOUND 0x108D6                                   /* file not found error (GBPB)*/




typedef struct op_descrip {                                 /* Internal description of operator */
```

```
                int level,
                    num_params;
                char *description;
                } op_descrip;

extern op_descrip operation[NUM_OPS];

#define text_of(box)  operation[box->op_num-0x80].description
#define op_level(token) operation[token-0x80].level
#define op_num_params(token) operation[token-0x80].num_params



typedef struct op_tokens {                                  /* Marry token and text for token */
                char *text;
                int token;
                } op_tokens;


/* below, op_num is operation number, as given above, indexing operation[]
          box_num is number of box in sequence.
          First box is 0 = header box
          so first box which does anything is #1
*/

/* data types.... */

#define NO_PARM 0
#define NUMBER 1
#define VARIABLE 2

typedef struct flow_box_type {
              int op_num,                                       /* operation token, TOK_xxxx */
                  box_num,                                  /* in sequence from start of chart */
                  dtype;                                     /* var or number,defined above */

              union data {
                    double param1;                                  /* data is either number */
                    char *title;                                          /* or name string */
                    char *varname;                                      /* or variable name */
                   } data;

              struct flow_box_type *prev,                             /* double linked list */
                                   *next;
              } flow_box_type;


/* FLOWCHART VARIABLES : */


typedef struct variable {                                  /* user variables stored in these */
                int status;                                      /* as in V_ constants above */
                double value;
                char name[15];
                } variable;
```

```
extern variable var[NUM_VARS];


typedef struct rrec  {                                  /* run record store of previous results */
                       char inpt[60];
                       char output[60];
                       int flownum;
                     } rrec;


extern rrec rec_line[MAX_REC_LINES+1];



/* global variables ... */


extern double EPSILON;

extern char *HELPDIRNAME,
            *HELPNAMESTYLE,
            *FLOWDIRNAME,
            *sp_area;                                   /* user sprite area for odd bits */


extern int w_run,                                       /* window handles begin w_*/
           w_flowcharts,
           w_runentry,
           w_runrecord,
           w_help,
           w_error,
           w_graphinfo,
           w_panel,
           w_plotwindow,
           w_vartable,
           w_boxedit,

/* icon handles ..... */

           calcscreen,
           clearbutton,                                 /* in box editor */
           enterbutton,                                 /* ENTER, in box editor */
           calcbutton[40],
           set_op_level,                                /* teacher defined op level set */

           runbutton,
           clearinbutton,
           swapbutton,
           autoswapbutton,                              /* icon to toggle autoswap */
           autoswap,

           err_ok_icon,                                 /* in error window */

           calcicon,                                    /* on control panel */
           runicon,
           flowicon,
           helpicon,
           screensaveicon,
```

```
          exiticon,


/* plotting items */

          curve_colour[MAXGRAPHS],
          index[MAXGRAPHS],
          nextfree,


/* "config" variables, affecting way routines work , and access to
       routines ....
*/

          accuracy_type,                                  /* SIG_FIGS, DEC_PLACES, or NONE */
          degrees,                                        /* boolean, degs or rads switch */
          master_status,                                          /* privilege settings */
          advanced_status,
          vars_allowed,
          paths_allowed,
          printing_status,
          oldfilename_pos,
          max_box_number,                                 /* highest box number on chart */
          current_box_num,                                /* box currently highlighted */
          current_flowchart,
          FLOW_HEIGHT;                                /* Height of flowcharts window, pixels */

extern flow_box_type
               flowchart[MAX_FLOWS+1],                    /* Header boxes for each chart */
               *current_box;

extern error *my_error;                                   /* For machine SWIs, in f_main */

extern char *screen,                                            /*boxeditor screen text */
           *cbtext[CALC_ROWS*CALC_COLS],                /* Text for boxeditor buttons */
           *numinput,                                         /* runentry input string */
           *numoutput,                                       /* runentry output string */
           *runflownum,                                         /* runentry box icons */

/* writeable menu items..... */

           *loadfilename,                                    /*filename to load, string */
           *savefilename,                                        /* 12 characters long */
           *newfilename,
           *oldfilename,
           *Xminnum,                                          /* graph minimum x value */
           *Xmaxnum,                                            /* ditto, max x value */
           *Fgraphnum,                                   /* first chart number on graph */
           *Super_num,                                  /* second and subsequent numbers */
           *del_var_name,                                    /* variable name to delete */
           *crtvar_name,                                      /*created variable name */
           *crtvar_val,                                      /*created variable value */
           *Accnumber;                                      /* String integer accuracy */


extern menu_block Edit_menu,
                  Prefs_menu,
```

```
                File_menu,
                Clear_menu,
                Graph_window_menu,
                Vars_menu,
                *Help_menu,
                *Load_menu,
                *Delete_menu,
                *Rename_menu,
                newname_menu;


/************************************************/
/****E X T E R N A L   P R O T O T Y P E S *****/
/************************************************/




extern void save_flowcharts(char *filename),
            load_flowcharts(char *filename),
            clear_variables(void),
            charsize(int, int),
            read_dbl(char *buf,double *number),                          /* allows commas in number */
            help_menu(univ_block *block),                                          /* in f_help */
            load_menu(univ_block *block),                                           /* f_ops */
            redraw_flowchart_number(int num),
            redraw_graphinfo(redraw_block *block),                                 /* f_plot */
            redraw_vartable(redraw_block *block),
            redraw_help(redraw_block *block),                                      /* f_help */
            mouse_on_help(univ_block *),                                           /* f_help */
            mouse_on_vartable(univ_block *),
            Edit_menu_decode(univ_block * block),
            clearrunrecord(void),                                                   /* f.ops */
            mse_on_runentry(univ_block*),
            mse_on_runrecord(univ_block*),
            graph_plot(int nump,double *x, double *y,int horiz,int vert,int colour),
            super_plot(int numpoints,double *x,double *y,int colour),
            adjust_to_origin(int w_handle,univ_block *block),
            stripspaces(char*),                                                    /* in f_ops */
            key_on_runrecord(univ_block*),                                         /* in f_ops */
            key_on_runentry(univ_block *),
            mouse_enters_runrecord(univ_block*),
            mouse_leaves_runrecord(univ_block *),
            mouse_enters_runentry(univ_block*),
            redraw_runrecord(redraw_block*),
            redraw_runentry(redraw_block*),
            mouse_enters_runentry(univ_block*),
            mouse_leaves_runentry(univ_block*);



extern menu_block *setup_dir_menu(char *header, char *directory),
                  *setup_dir_menu_r(char *header, char *directory),
                  *setup_rename_menu(char *header, char *directory);


extern double evaluate(double input,int fnum,flow_box_type *box, int *err);
```

```
                                                                    /*pass through chart*/

extern int syntax_ok(char* buffer, flow_box_type *box),                          /* in f_ops */
           length_of_flowchart(int number),
           wimp_decision(char *text),
           is_pathname(char *filename),                                /* true if full pathname */
           gr_origin_x(void),                                  /* returns x pos of graphics origin */
           gr_origin_y(void);                                          /* ditto, y pos'n */
```

9.4.2 H.HOURGLASS

```
/*>h.hourglass
        header for hourglass module
        v0.01 MT 11/9/89
*/

extern  void Hg_On(void),
             Hg_Off(void),
             Hg_Smash(void),
             Hg_Start(int delay),
             Hg_Percent(int percent),
             Hg_LEDs(int eor,int and);
```

9.4.3 H.WINDASM

```
/* h.windasm */
extern void setup_winds(void),
            settype(char* file,int type),
            a_open_wimp(void),
            close_wimp(void);

extern int  fextent(char* pathname),
            is_a_dir(char* pathname);

extern error  *rename_file(char * old, char *new),
              *delete_file(char* name);
```

9.4.4 H.WINDS

```
/*>h.winds


 @    @         @    @     @    @    @ @@@@@    @@@@
 @    @         @    @     @    @@   @ @    @ @
 @@@@@@         @    @     @    @ @  @ @    @   @@@@
 @    @   @@@   @ @@ @     @    @  @ @ @    @       @
 @    @   @@@   @@  @@     @    @   @@ @    @ @    @
 @    @   @@@   @    @     @    @    @ @@@@@    @@@@

*/

/*********************************/
/*******Default palette**********/
/*********************************/

#define white 0
#define grey1 1
#define grey2 2
#define grey3 3
#define grey4 4
#define grey5 5
#define grey6 6
#define black 7
#define dark_blue 8
#define yellow 9
#define green 10
#define red 11
#define cream 12
#define dark_green 13
#define orange 14
#define blue 15


#define max_windows 30
#define WRITEABLE (15<<12)                              /* in window work area flags */
#define NOTIFY_APPLIC (3<<12)
#define CAN_SELECT (9<<12)
#define SELECT_NOTIFY (4<<12)
#define CLICK_SELECT (5<<12)
#define CLICK_DRAG_DBL (10<<12)

/* wimp_error flags */

#define WE_OK 1
#define WE_CANCEL 2
#define WE_HILITE_CANCEL 4
#define WE_RETURN_IMMED 8
#define WE_NOERROR 16
#define WE_NOCLICKRET 32
#define WE_SELONE 64


#ifndef TRUE
#define TRUE 1
#define FALSE 0
```

```
#endif

#define ON_TOP -1

#define TITLE_FGCOL   black
#define TITLE_BGCOL   grey2
#define WA_FGCOL      black
#define WA_BGCOL      grey4
#define SB_FGCOL      grey1
#define SB_BGCOL      grey2
#define HL_COL        cream

#define MENU_TIT_FGCOL black
#define MENU_TIT_BGCOL grey3
#define MENU_WA_FGCOL  black
#define MENU_WA_BGCOL  white

#define MENUTI_FGCOL  black
#define MENUTI_BGCOL  white

#define MENUWI_FGCOL  white
#define MENUWI_BGCOL  red


/* Standard colours for windows... */

#define stdcols()         (char) TITLE_FGCOL,                          /*titfg*/     \
                          (char) TITLE_BGCOL,                          /*titbg*/     \
                          (char) WA_FGCOL,                             /*wafg*/      \
                          (char) WA_BGCOL,                             /*wabg */     \
                          (char) SB_BGCOL,                           /*scroll outer*/\
                          (char) SB_FGCOL,                          /*scroll inner */\
                          (char) HL_COL,                            /* highlight */\
                          (char) 0,                                    /* always */

#define tit_bar_cols      (TITLE_FGCOL<<24)+(TITLE_BGCOL<<28)


/* standard colours for menus ... */

#define stdmencols()        (char) MENU_TIT_FGCOL,\
                            (char) MENU_TIT_BGCOL,\
                            (char) MENU_WA_FGCOL, \
                            (char) MENU_WA_BGCOL,




#define input_focus_off()  input_focus_to(-1)


typedef struct wind_info_block
{
        int wind_handle;                                          /* window handle */
        int x0, y0, x1, y1;                                  /* work area coordinates */
```

```
        int scx, scy;                                                   /* scroll bar positions */
        int wind_behind;                                               /* handle of window in front*/
        int flags;                                                          /* word of flag bits  */
        char colours[7];        /* colours: foreground, background,
                                 * work area foreground, background,
                                 * scroll bar outer/inner, title background
                                 * if window has highlight colour */
        char reservedb;                                                          /* reserved byte */
        int exx0, exy0, exx1, exy1;                                           /* extent coordinates */
        int title_icon, work_icon;                                 /* icon flags for title bar, work area */
        int reservedw[2];                                                       /* reserved space */
        char title[12];                                                          /* title string */
        int initial_icons;                                              /* no. of icons in definition */
        int iconspace[20*32];

} wind_info_block;



extern void usr_close_wind(int handle),
            usr_open_wind(int handle,int wind_behind),
            usr_open_wind_scroll(int handle,int wind_behind,int sx,int sy),
            usr_update_wind(int handle),
            usr_update_scroll_wind(int handle,int x,int y),
            force_update(int handle,int xl,int yl,int xh,int yh),
            usr_create_menu(menu_block*,int,int,void(*func)(univ_block*)),
            input_focus_to(int handle),
            open_wimp(int bgcol),
            scroll_window(int handle,int xd,int yd),
            set_window_extent(int handle,int x0,int y0,int x1,int y1),
            wputs(int handle,int xpos,int ypos,char* text),

/* set up functions to respond to user actions ... */

            key_pressed_function(int,void(*func)(univ_block*)),
            mse_pressed_function(int,void(*func)(univ_block*)),
            window_redraw_function(int,void(*func)(redraw_block*)),
            wind_leave_function(int,void(*func)(univ_block*)),
            wind_enter_function(int,void(*func)(univ_block*)),
            close_window_function(int,void(*func)(void)),
            open_window_function(int,void(*func)(void)),
            background_mouse_function(void(*func)(univ_block*)),
            make_unselectable(int icon,int wind_handle),
            make_selectable(int icon,int wind_handle),
            put_caret(int handle,int x,int y),
            do_error(char *text),
            do_error_num(int errnum,char* errmess),
            set_taskname(char *name),
            poll(int mask),
            get_wind_info(wind_info_block*,error*),

/* miscellany...  */
          move_mouse_to(int x,int y),
          mouse_bound_box(int x0,int y0,int x1,int y1),
          mouse_unbound(void),
          confine_pointer_to(int whandle),
          put_sprite(char* name,int action, char* userarea);
```

```
extern int usr_create_wind(wind_block*),
          make_text_icon(int window,int x,int y,char*,
                         int buttype,int fg,int bg,int height),
          make_sprite_icon(int window,int x,int y,int wid,int ht,char *name,
                           char *sblock,int type),
          is_icon_selected(int,int),
          wimp_error(error * e,int flags),
          is_window_open(int handle),
          is_window_fullsize(int handle),
          is_window_top(int handle),
          wimp_decision(char *message),
          intersect(redraw_block*,int xmi,int ymi ,int xma ,int yma),
          make_writeable_icon(int window,int x,int y,char*,int length,
                              int height,int fg,int bg);

                                                                    /*error handling */


                                                                   /* Dialog boxes ... */
extern char* sp_area;
```

9.4.5 C.F MAIN

```
/*> c.f_main


@@@@@@@        @     @    @      @@@   @     @
@              @@   @@   @ @       @    @@    @
@              @ @ @ @  @   @      @    @ @   @
@@@@@          @  @  @ @     @     @    @  @  @
@              @     @ @@@@@@@     @    @   @ @
@              @     @ @     @     @    @    @@
@       @@@@@@@ @     @ @     @    @@@   @     @


    Collection of main top-level routines for Flowchart
    This section contains main() which is the first routine called

*/

/* Local "include" files :  */

#include "flow.h"                                                    /* Program global manifests */
#include <signal.h>
#include <arthur.h>
/* Edit modes */

#define APPEND 1
#define OVERWRITE 2
#define INSERT_AFTER 3
#define INSERT_BEFORE 4

/* file containing sprite definitions*/
#define SPRITE_FILE_NAME "<Obey$dir>.Flowspr"

#define PARAM (box->data.param1)

/* MAXGRAPHS graphs allowed on-screen at once */

int    curve_colour[MAXGRAPHS] = {
                red,yellow,green,dark_blue
       },                                                             /* current colour of curve */

       set_op_level;                                 /* operation level at which we are currently working */


static int printing_boxnumbers=TRUE,
           edit_mode=APPEND;

int        index[MAXGRAPHS],                                          /* index of graphs plotted */
           nextfree;                                                 /* number of graphs plotted */

char *boxinfo="BXINF";                                      /* name of sprite in user_area for boxeditor*/

static double x[MAXGRAPHS] [NUM_GRAPH_POINTS],                                /* x coordinates */
              y[MAXGRAPHS] [NUM_GRAPH_POINTS];                                /* y coordinates */

/*********************************/
/* PROTOTYPES                    */
```

```
/*********************************/
/*
       These are documented in more detail as they occur in the program
*/

void leave_program(void),                                          /* quit from program */
     wipe_chart(int chartnum),                                      /* clear this chart */
     wipe_all_charts(void),                                         /* clear all charts */
     do_graph(int chartnum,double xmin,double xmax),                   /* plot a graph */
     delete_box(flow_box_type * this_one),                        /* rmove box from chart */
     update_calc_screen(void),                                  /* replot calculator screen */
     graph_window_closing(void),                               /* cope with window closing.. */
     clear_graphs(void),                          /* clear the graph windows of all bits, reset scales */

     update_boxrecs(void),                                    /* change indication of edit state */
     setup_calc_screen(flow_box_type *box),                                /* initialise screen */
     undo_plot_record(int num);                             /* remove record that has been plotted */

static int find_index(int),
       ensure_box_is_visible(int cur_flow,int cur_box);

/*********************************/
/* Data defs, this program       */
/*********************************/

flow_box_type   flowchart[MAX_FLOWS+1],                          /* initial boxes for each chart */
/*..they continue as a linked list */
               *current_box=&flowchart[1];                                          /* initially */

static char *errmsg,                                    /* used to provide error reporting facilities */
           *moveoverarrow = "Move pointer over the \x89 to select new name";

char
     *HELPDIRNAME,                                          /* the name of the help directory */
     *HELPNAMESTYLE,                                       /* the style of the help file names */
     *FLOWDIRNAME,                              /* the name of the directory containing flowcharts */
     *newfilename,                                                        /* filenames...*/
     *oldfilename,
     *prog_sprites,                           /* program sprites area (sprites NOT defined in file) */
     *sp_area;                                     /* sprite area block (sprites defined in file) */


double EPSILON;

int current_flowchart=1,                                        /* start with this flowchart */
    max_box_number=0,                                  /* highest box number present on chart */
    FLOW_HEIGHT=W_FLOWBOX_HEIGHT,                    /* current height, pixels, flowcharts window*/
    current_box_num=0,                                                /* start on this box */
    accuracy_type=NONE,                                  /* sig, dp or none, default none */
    degrees=TRUE,                                                     /* True or false... */
    master_status=FALSE,                                /* allows "Master level" operations */
    advanced_status=FALSE,                            /* allows "advanced" ops e.g. file delete*/
    printing_status=FALSE,                                    /* printing allowed by user */
    vars_allowed=FALSE,                                       /* User has named variables */
    paths_allowed=FALSE;                                   /* full paths allowed in filenames */
                                                          /* means user can move about freely */
```

```
/*********************************************************************/
/* MENU  functions                                                   */
/* Including decoding menus , and miscellaneous functions to do      */
/* Setting and unsetting ticks in menus                              */
/*********************************************************************/


static void unset_tick_item(int num, menu_block *menu)
{

/* remove the tick from item num in menu menu */

        menu->m[num].menu_flags=(menu->m[num].menu_flags)&~LEFT_TICK;
}



static void unset_all_ticks(menu_block *menu)
{
/* remove ticks from all items in menu menu */
        int i;
        for(i=0;i<MAXITEMS;i++) unset_tick_item(i,menu);
}


void set_tick_item(int num, menu_block *menu)
{

/* set tick item num in menu menu */

        menu->m[num].menu_flags=(menu->m[num].menu_flags)|LEFT_TICK;
}

int is_pathname(char *p)
{

/* TRUE is p is a pathname, else FALSE */

        int i=0,is_path=FALSE;
        while(p[i]!=NULL && isalnum(p[i])) i++;
        if(p[i]!=NULL) is_path=TRUE;                                    /* non alpha => pathname */
        if(is_path&&(!paths_allowed)) {
                is_path=FALSE;
                do_error("Illegal character in filename");
        }
        return(is_path);
}


void load_menu(univ_block *block)
{
/* User is loading a flowchart file...
   block[0] is the item number in the load_menu which the user has selected to load
*/
        int i=0;
        char filename[168];
        textbuf *p;
        if(block->a[0]==oldfilename_pos) {
```

```
                strcpy(filename,oldfilename);
                my_error=NULL;
                p=(textbuf*)filename;
        } else {
                sprintf(filename,"%s.",FLOWDIRNAME);
                p=(textbuf*)(filename+strlen(FLOWDIRNAME)+1);
                decode_menu(Load_menu,block,p,my_error);
                while(filename[i]!=0x0D) i++;
                filename[i]=NULL;
        }
        if(my_error!=NULL) {
                wimp_error(my_error,WE_OK);
        } else  {
                if(is_pathname((char*)p)) load_flowcharts((char*)p);
                else load_flowcharts(filename);
                strcpy(savefilename,(char*)p);
                clear_graphs();                                  /* all plot tables now invalid */
       }
}


void delete_menu(univ_block *block)
{

/* User has  opened and clicked the delete menu.. find the filename required and delete it */

        char filename[168];
        if(block->a[0]==oldfilename_pos) {
             strcpy(filename,oldfilename);
             my_error=NULL;
        } else {
             int i=0;
             sprintf(filename,"%s.",FLOWDIRNAME);
             decode_menu(Delete_menu,block,
                    (textbuf*)(filename+strlen(FLOWDIRNAME)+1),my_error);
             while(filename[i]!=0x0D) i++;
             filename[i]=NULL;
        }
        if(my_error==NULL) my_error=delete_file(filename);
        if(my_error!=NULL) wimp_error(my_error,WE_OK);
}

void rename_menu(univ_block *block)
{

/* User wishes to rename a file. block[0] is the item number to be renamed */

        int i=0;
        textbuf filename,newname;
        if(block->a[1]!=-1) {                            /* only respond if user clicked on new name*/
          block->a[1]=-1;
          if(block->a[0]==oldfilename_pos) {
               strcpy(filename.a,oldfilename);
               my_error=NULL;
          } else {
               sprintf(filename.a,"%s.",FLOWDIRNAME);
               decode_menu(Rename_menu,block,
                    (textbuf*)(filename.a+strlen(FLOWDIRNAME)+1),my_error);
```

```
            while(filename.a[i]!=0x0D) i++;
            filename.a[i]=NULL;
          }
          if(is_pathname(newfilename)) strcpy(newname.a,newfilename);
          else sprintf(newname.a,"%s.%s",FLOWDIRNAME,newfilename);
          if(my_error==NULL)
                my_error=rename_file((char*)filename.a,(char*)newname.a);
          if(my_error) wimp_error(my_error,WE_OK);
        } else do_error(moveoverarrow);
}


void graph_super_menu(univ_block *block)
{

/* User has elected to superimpose a graph on existing axes */
/* determine graph number, and redraw... */

        switch(block->a[0]) {
        default :
        case (GSM_GO) :
                        do_graph(atoi(Super_num),atof(Xminnum),atof(Xmaxnum));
                        usr_update_wind(w_plotwindow);
                        break;
        }
}

static void clear_graph_index(void)
{

/* clear the list of flowcharts plotted */

        int i;
        nextfree=0;
        for(i=0;i<MAXGRAPHS;i++) index[i]=0;
}

static void graph_window_menu(univ_block *block)
{

/* User clicked on graph window menu, which has three options...*/

        switch(block->a[0]) {
           case (GWM_INFO) :
                                usr_open_wind(w_graphinfo,ON_TOP);
                                break;

           case (GWM_CLEAR) :
                                clear_graphs();
                                break;

           case (GWM_SUPERIMP) :
                                graph_super_menu((univ_block*)&block->a[1]);
                                break;

           default :            break;
        }
}
```

```
static void graph_menu(univ_block *block)
{

/* Graph menu consists largely of writeable bits, e.g.
   flow number to plot, range, and only one selectable item,
   which will cause plot to occur. "default" ensures that user
   can click anywhere on menu to plot. "Go" isn't necessary, but is reassuring
*/
        switch(block->a[0]) {

          default:
          case GM_GO :
                      do_graph(atoi(Fgraphnum),atof(Xminnum),atof(Xmaxnum));
                      break;


        }
}



static void file_menu(univ_block *block)
{

/* The MAIN file menu parsing routine..loads and saves, mainly */

        switch(block->a[0]) {

        case FM_LOAD :
                       load_menu((univ_block*)&block->a[1]);
                       break;

        case FM_RENAME :
                       rename_menu((univ_block*)&block->a[1]);
                       break;

        case FM_DELETE :
                       delete_menu((univ_block*)&block->a[1]);
                       break;

        case FM_SAVE :
        case FM_SAVE_FILENAME:
                       save_flowcharts(savefilename);
                       break;
        }

/* ensure menus are up to date... */

        if (advanced_status) {

          Delete_menu=File_menu.m[FM_DELETE].sub_menu=
                              setup_dir_menu("Delete..",FLOWDIRNAME);

          Rename_menu=File_menu.m[FM_RENAME].sub_menu=
```

```
                        setup_rename_menu("Rename..",FLOWDIRNAME);

        }

        Load_menu=File_menu.m[FM_LOAD].sub_menu=
                        setup_dir_menu("Load chart",FLOWDIRNAME);

}


static void clear_menu(int choice)
{

/* User selected CLEAR from Edit menu. He has 4 options here...*/

        switch(choice) {

        case CM_ALL :                                                   /* clear all flowcharts */
                        wipe_all_charts();
                        max_box_number=0;
/* reset window dimensions ..*/
                        FLOW_HEIGHT=W_FLOWBOX_HEIGHT;
                        set_window_extent(w_flowcharts,0,0,W_FLOWBOX_WIDTH,FLOW_HEIGHT);
                        usr_close_wind(w_flowcharts);
                        usr_open_wind(w_flowcharts,ON_TOP);

/* and drop through to ..... */

        case CM_GRAPHS :

                        clear_graphs();
                        break;


        case CM_CURRENT :                                       /* clear only the current chart */

                        wipe_chart(current_flowchart);
                        break;


        case CM_RUNRECORD :                                             /* clear the runrecord */
                        input_focus_off();                      /* remove the caret or highlight */
                        clearrunrecord();                               /* and remove all records */
                        break;

        case CM_VARS  :                                         /* clear the variables space */
                        clear_variables();
                        break;
        }
}



static void prefs_menu(int choice)
{

/* User selected preferences... from edit menu */
```

```
      switch(choice) {
/**** Degrees bit ****************************/

        case PM_DEGS :  unset_tick_item(PM_RADS,&Prefs_menu);                    /* undo rads */
                        set_tick_item(PM_DEGS,&Prefs_menu);                    /* set degrees */
                        degrees=TRUE;                                        /* and tell program */
                        break;

        case PM_RADS :  unset_tick_item(PM_DEGS,&Prefs_menu);                     /* ditto here */
                        set_tick_item(PM_RADS,&Prefs_menu);
                        degrees=FALSE;
                        break;

/**** Accuracy bit ****************************/

        case PM_SIG_FIGS : unset_tick_item(PM_DEC_PLACES,&Prefs_menu);
                        unset_tick_item(PM_NO_ACCURACY_SET,&Prefs_menu);
                        set_tick_item(PM_SIG_FIGS,&Prefs_menu);
                        accuracy_type=SIG_FIGS;
                        break;
        case PM_DEC_PLACES :
                        unset_tick_item(PM_SIG_FIGS,&Prefs_menu);
                        unset_tick_item(PM_NO_ACCURACY_SET,&Prefs_menu);
                        set_tick_item(PM_DEC_PLACES,&Prefs_menu);
                        accuracy_type=DEC_PLACES;
                        break;

        case PM_NO_ACCURACY_SET :
                        unset_tick_item(PM_SIG_FIGS,&Prefs_menu);
                        unset_tick_item(PM_NO_ACCURACY_SET,&Prefs_menu);
                        set_tick_item(PM_NO_ACCURACY_SET,&Prefs_menu);
                        accuracy_type=NONE;
                        break;

        default :       break;                              /* Clicking on writeable icon does nothing*/

        }

}


void Edit_menu_decode(univ_block *block)
{

/* Top level decode of edit menu.
   This will in a number of cases, pass the decoding on to the preceding
   routines for further work before calling a routine that does something
   The order here follows the order in the menu itself
*/
        switch(item(0)) {

/******** Edit mode section **********/

        case EM_APPEND : edit_mode=APPEND;
                         unset_all_ticks(&Edit_menu);
                         set_tick_item(EM_APPEND,&Edit_menu);
                         update_boxrecs();
```

```
                        break;

        case EM_OVER    : edit_mode=OVERWRITE;
                          unset_all_ticks(&Edit_menu);
                          set_tick_item(EM_OVER,&Edit_menu);
                          update_boxrecs();
                          break;

        case EM_AFTER : edit_mode=INSERT_AFTER;
                        unset_all_ticks(&Edit_menu);
                        set_tick_item(EM_AFTER,&Edit_menu);
                        update_boxrecs();
                        break;

        case EM_BEFORE : edit_mode=INSERT_BEFORE;
                         unset_all_ticks(&Edit_menu);
                         set_tick_item(EM_BEFORE,&Edit_menu);
                         update_boxrecs();
                         break;

/******** Delete and CLEAR section **********/

        case EM_DELETEBOX :
                        delete_box(current_box);
                        break;

        case EM_CLEAR : clear_menu(item(1));
                        break;

/******** Miscellany **********/
/* nearly all of these pass on for further work... */

        case EM_PREFS :
                        prefs_menu(item(1));
                        break;


        case EM_FILE:   file_menu((univ_block*)&block->a[1]);                    /* more work*/
                        break;

        case EM_RUN :
                        usr_open_wind(w_runrecord,ON_TOP);
                        usr_open_wind(w_runentry,ON_TOP);
                        break;

        case EM_GRAPH:  if(block->a[1]!=-1) graph_menu((univ_block*)&block->a[1]);   /*more work */
                        else {
/* user clicked on graphs rather than moved over arrow */
                            if(nextfree!=0) usr_open_wind(w_plotwindow,ON_TOP);
                            else do_error("No graphs to plot");
                        }
                        break;

        case EM_HELP :  help_menu((univ_block*)&block->a[1]);                    /* more work */
                        break;

        case EM_VARS :  usr_open_wind(w_vartable,ON_TOP);
                        break;
```

```
        default :       break;

        }
}


/*********************************************************************/
/* FLOWCHART functions                                               */
/* These functions are concerned with manipulation of the            */
/* Flowcharts themselves, their internal representations, and        */
/* Their on-screen representations, but NOT evaluating flowcharts    */
/* OR checking syntax of boxes                                       */
/*********************************************************************/

void charsize(int x, int y)
{
/* blow up characters to x and y pixel sizes */
/* default is 8 x 8 */
/* Bug in multiple VDU call in this compiler... have to call singly.....*/

        vdu(23);
        vdu(17);
        vdu(7);
        vdu(6);
        vdu(x);
        vdu(0);
        vdu(y);
        vdu(0);
        vdu(0);
        vdu(0);
}


void update_boxrecs(void)
{
      force_update(w_boxedit,220,500,500,700);
}


void undo_plot_record(int num)
{

/* if the graph has been plotted then mark plot table invalid ( chart has been changed in some way) */

        int i=find_index(num);                                  /* check if graph has been plotted */
        if(i!=-1) {                                               /* -1 indicates not been plotted */
               while((i++)<nextfree) index[i]=index[i+1];
               index[nextfree--]=0;
               if(is_window_open(w_plotwindow)) usr_close_wind(w_plotwindow);
        }
}


void wipe_chart(int num)
{
```

```
/* clear the given flowchart number num */

        flow_box_type *nxt,*box=flowchart[num].next;
        while(box!=NULL)  {
                nxt=box->next;
                free(box);
                box=nxt;
        }
        flowchart[num].next=NULL;
        undo_plot_record(num);                                  /* modified version hasn't been plotted */
        redraw_flowchart_number(num);
}


void wipe_all_charts(void)
{

/* clear ALL flowcharts */

        int i;
        for(i=1;i<=MAX_FLOWS;wipe_chart(i++));
}


static void move_current_box(univ_block *block)

{

/* User has selected a different box to work with. Adjust all... */

      int old_box,old_flow;
      adjust_to_origin(w_flowcharts,block);
      old_flow=current_flowchart;
      old_box=current_box_num;
      current_flowchart=flow_num(mousex);

/* ensure in range.... */

      current_flowchart=min(MAX_FLOWS,max(1,current_flowchart));
      current_box=&flowchart[current_flowchart];

      current_box_num=box_num(mousey);

      while((current_box->box_num!=current_box_num)&&(current_box->next!=NULL))
            current_box=current_box->next;

      if(current_box==NULL) {
            current_box=&flowchart[current_flowchart];
            current_box_num=0;
      }



      if(old_flow!=current_flowchart||old_box!=current_box_num) {

          force_update(w_flowcharts,
                       0,FLOW_HEIGHT-90,
                       W_FLOWBOX_WIDTH,FLOW_HEIGHT);
```

```
          force_update(w_flowcharts,
                       xco_flowchart(old_flow)-300,yco_boxnum(old_box)-300,
                       xco_flowchart(old_flow)+300,yco_boxnum(old_box)+300);

          force_update(w_flowcharts,
                      xco_flowchart(current_flowchart)-300,
                      yco_boxnum(current_box_num)-300,
                      xco_flowchart(current_flowchart)+300,
                      yco_boxnum(current_box_num)+300);
      }
      setup_calc_screen(current_box);                                    /* Setup editor screen */
      update_calc_screen();                                                 /* if it's open... */
      update_boxrecs();
}


void mouse_on_flowcharts(univ_block *block)
{

/* A mouse button has been clicked whilst on the flowchart window */

        switch (mouseb) {

        case L_PRESSED :
                move_current_box(block);                                 /* SELECT = new current box */
                break;

        case M_PRESSED :                                                  /* MENU - pop up edit menu */
                usr_create_menu(&Edit_menu,mousex,mousey,Edit_menu_decode);
                break;

        default:                                                         /* ADJUST does nothing here */
                break;
        }
}


static void drawbox(int flownum, flow_box_type *this_box, int xp,int yp)
{

/* Main routine for drawing a box on-screen at xp,yp */

#define cbox  (this_box->box_num)
#define box_is_current (cbox==current_box_num&&flownum==current_flowchart)
        char contents[30];
        *contents=NULL;
        move(xp,yp);

        if(cbox==0) {
           if (box_is_current) gcol(0,grey3);                       /* now correct if box is header */
           else gcol(0,grey5);
        } else {
           if (box_is_current) gcol(0,green);                            /* non-header colours */
           else gcol(0,red);
        }

        drawby(0,-LINE_LENGTH);
```

```
        moveby(-BOX_WIDTH/2,-BOX_HEIGHT);
        plot(97,BOX_WIDTH,BOX_HEIGHT);                                  /* rectangle, filled */
        gcol(0,black);
        drawby(-BOX_WIDTH,0);
        drawby(0,-BOX_HEIGHT);
        drawby(BOX_WIDTH,0);
        drawby(0,BOX_HEIGHT);
        if(box_is_current) {
           moveby(4,-4);
           drawby(0,-BOX_HEIGHT);
           drawby(-BOX_WIDTH,0);
           moveby(4,-4);
           drawby(BOX_WIDTH,0);
           drawby(0,BOX_HEIGHT);
           moveby(-8,8);
        }
        if(printing_boxnumbers && (cbox>0)) {
           moveby(-BOX_WIDTH+10,-3);
           gcol(0,white);
           moveby(-printf("%d",cbox)*CHAR_WIDTH,3-(BOX_HEIGHT/2));
           gcol(0,black);
        } else moveby(-BOX_WIDTH+10,-BOX_HEIGHT/2);

/* Skeleton now drawn, now determine what goes inside the box */

        strcpy(contents,text_of(this_box));
        if(cbox!=0) switch(op_num_params(this_box->op_num)) {

              case 0 :
                      break;

              case 1 :
                      switch(this_box->dtype) {

                      case(NUMBER) :                                    /* param is a number */
                          if(this_box->op_num==TOK_SUBFLOW)
                                                                        /* call other flowchart */
                               sprintf(contents+strlen(contents),"%d",(int)(this_box->data.param1));
                                                                        /* else "normal" call */
                          else sprintf(contents+strlen(contents),"%.4f",(this_box->data.param1));
                          break;

                      case(VARIABLE) :                                  /* param is a variable (named) */
                          strncat(contents,this_box->data.varname,10);
                          break;

                      case(NO_PARM) :
                          break;
                      }
                      break;
        }
        else sprintf(contents,"F %d",flownum);                          /* Must be the header box */
        if(strlen(contents)<=10) printf(contents);                      /* Won't overflow the box */
        else {
             contents[10]=NULL;                                         /* truncate the string */
             printf("%s\x89",contents);                                 /* and indicate to user this has occured */
        }
#undef cbox
```

```
}


static void redraw_flowcharts(redraw_block *block)
{


/* redraw all the flowcharts. This routine is called by the WIMP, and not by the program "manually" */

        int flownum=1,xco,yco;
        flow_box_type *this_box;

        while(flownum<=MAX_FLOWS) {
            this_box=&flowchart[flownum];
            while(this_box!=NULL) {
                xco=xco_flowchart(flownum);
                yco=yco_boxnum(this_box->box_num);
                if(flownum==current_flowchart&&this_box->box_num==0) {
                        move(xco-50,yco+50);
                        gcol(0,white);
                        charsize(8,16);
                        puts("CURRENT");
                        charsize(8,8);
                }

/* Only re-do it if it is on-screen.... */

                if(intersect(block,xco-300,yco-400,xco+300,yco+400))
                        drawbox(flownum,this_box,xco,yco);

                this_box=this_box->next;
           }
           flownum++;
       }
}




/*********************************************************************/
/* BOXEDITOR  functions                                              */
/*********************************************************************/



void redraw_flowchart_number(int num)
{

/* Just update one flowchart, generally the current one */

        int xco=xco_flowchart(num);
        force_update(w_flowcharts,xco-200,0,xco+200,FLOW_HEIGHT);
}


static void redraw_boxeditor(redraw_block *block)

{
```

```
/* redraw the boxeditor window. Most is done by the WIMP */

        move(220,660);
        gcol(0,white);
        printf("Chart :%d",current_flowchart);
        move(220,620);
        printf("box   :%d",current_box_num);
        move(220,580);
        printf("Mode  :");
        switch(edit_mode) {
          case APPEND       : puts("Append");
                              break;
          case OVERWRITE    : puts("Over");
                              break;
          case INSERT_AFTER : puts("Add after");
                              break;
          case INSERT_BEFORE: puts("Add before");
       }
}



static void redraw_current_flowchart(void)
{

/* redraw the current flowchart */

        redraw_flowchart_number(current_flowchart);
}

void setup_calc_screen(flow_box_type *box)
{

/* Fill the boxeditor "screen" with appropriate contents */

        switch(op_num_params(box->op_num)) {

              case 0 : strcpy(screen,text_of(box));
                       break;

              case 1 : switch (box->dtype) {

                       case VARIABLE :

                        sprintf(screen,"%s%s",text_of(box),box->data.varname);
                        break;

                       case NUMBER :
                        if(box->op_num==TOK_SUBFLOW)
                              sprintf(screen,"F %d",(int)box->data.param1);
                        else  sprintf(screen,"%s%f",text_of(box),box->data.param1);
                        break;

                       case NO_PARM :

                         strcpy(screen,text_of(box));
```

```
                        break;

                    }
                    break;
        }
}

void update_calc_screen(void)
{

/* redraw just the boxeditor screen, generally because the user has
   done something to it, other than typing characters */

        force_update(w_boxedit,
                80,W_BOXEDIT_HEIGHT-4*CHAR_HEIGHT,
                80+(SCREENLENGTH+5)*CHAR_WIDTH,W_BOXEDIT_HEIGHT-4*CHAR_HEIGHT+96);

}

static void overwrite_box(flow_box_type *current, flow_box_type *box)
{

/* Overwrite the current box, *current, with what is in *box
   ..note that current is a value parameter, not global variable
*/
        if((current!=NULL)&&(current->box_num!=0)) {
          current->op_num=box->op_num;
          current->dtype=box->dtype;
          current->data.param1=PARAM;
        } else do_error("No legal current box");
}

void delete_box(flow_box_type *current)
{

/* delete the box *current (local variable, not necessarily the global
   "current", but must be in current flowchart.
*/

        flow_box_type *previous=&flowchart[current_flowchart];
        if(current==previous) do_error("Can't delete header box");
        else {                                                          /* find the box */
           while(previous->next!=current) previous=previous->next;
           previous->next=current->next;                                /* delete it */
           if (current->next)  current->next->prev=previous;
           current_box_num=previous->box_num;                           /* renumber the boxes */
           while(previous->next!=NULL) {                    /* Just using previous as convenience here */
               previous->next->box_num=previous->box_num+1;
               previous=previous->next;
          }
          free(current);
          current_box=previous;
          redraw_current_flowchart();
          undo_plot_record(current_flowchart);                          /* any existing plot invalid */
       }
}
```

```
static void update_max(int boxnum)
{

/* check if flowcharts window needs resizing as a result of users activity */

    if (max_box_number<boxnum) {
            max_box_number=boxnum;
            FLOW_HEIGHT=W_FLOWBOX_HEIGHT+max_box_number*TOT_BOX_HT;
            set_window_extent(w_flowcharts,0,0,W_FLOWBOX_WIDTH,FLOW_HEIGHT);
            usr_update_scroll_wind(w_flowcharts,xco_flowchart(current_flowchart),yco_boxnum(current_box_num));

       }
}

static void append_box(flow_box_type *current, flow_box_type *box)
{

/* append the box at the end of the current chain */

        while(current->next!=NULL) current=current->next;                                   /* find end */
        current->next=malloc(sizeof(flow_box_type));
        if(current->next) {
           current->next->box_num=(current->box_num)+1;
           current->next->prev=current;

/* set global pointer... */

           current_box=current->next;
           current=current->next;
           current_box_num=current->box_num;                                /* make new box current */
           current->op_num=box->op_num;
           current->dtype=box->dtype;
           current->data.param1=PARAM;
           current->next=NULL;
        }
}


static void insert_after_box(flow_box_type *current,flow_box_type *box)
{

/* insert the box box after the box current */

        flow_box_type *link=current->next;
        current->next=malloc(sizeof(flow_box_type));
        current->next->prev=current;

/* now play with the contents of the new box */

        current=current->next;
        current->box_num=current->prev->box_num+1;
        current->next=link;
        current->op_num=box->op_num;
        current->dtype=box->dtype;
        current->data.param1=PARAM;

/* set global pointer */
```

```
        current_box=current;

/* make inserted box current */

        current_box_num=current->box_num;

/* relink the box numbers in the chain... */

        while(current->next!=NULL) {
                current->next->box_num=(current->box_num)+1;
                current=current->next;
        }
}


static void insert_before_box(flow_box_type *current,flow_box_type *box)
{

/* inserting before is equivalent of inserting after the previous! */

    if(current->prev!=NULL) insert_after_box(current->prev,box);
}

static void reset_caret(void)
{

/* move caret to start of writeable icon if it is inside one */

        reg_set regs;
        int block[30];
        regs.r[1]=(int)block;
        my_error=swix(Wimp_GetCaretPosition,&regs);
        if(my_error) wimp_error(my_error,WE_OK);
        block[5]=0;                                                    /* index into string */
        regs.r[1]=(int)block;
        if((block[0]!=-1)&&(block[1]!=-1)) {
/* if inside window and writeable icon then... */
           regs.r[0]=block[0];                                           /* window handle */
           regs.r[1]=block[1];                                            /* icon handle */
           regs.r[4]=-1;                                  /* work out pixel position from string index */
           regs.r[5]=0;                                              /* string index is at start */
           my_error=swix(Wimp_SetCaretPosition,&regs);
           if(my_error) wimp_error(my_error,WE_OK);
        }                                                                /* else do nothing */
}


static void update_current_flowchart(flow_box_type *box)
{

/* high level routine, controlling the previous routines */

        switch(edit_mode) {
        case APPEND :
                        current_box=&flowchart[current_flowchart];
                        append_box(current_box,box);
                        break;
```

```
        case OVERWRITE :
                    overwrite_box(current_box,box);
                    break;

        case INSERT_AFTER :
                    insert_after_box(current_box,box);
                    break;

        case INSERT_BEFORE :
                    insert_before_box(current_box,box);
                    break;
        }

        if(!ensure_box_is_visible(current_flowchart,current_box_num))
              redraw_current_flowchart();                              /* redraw the flowchart */
        undo_plot_record(current_flowchart);                         /*plotted points no longer valid*/
        update_max(length_of_flowchart(current_flowchart));                    /* check if window size
        update_boxrecs();

/* Update the text in the BOX Editor... */

        if(&flowchart[current_flowchart]!=NULL) setup_calc_screen(current_box);

        *screen=END_OF_STRING;
        reset_caret();                                    /* put caret at start , if it is in use */
        update_calc_screen();
}


static int ensure_box_is_visible(int cur_flow, int cur_box)
{
        wind_info_block wib;
        int xco,yco;
        wib.wind_handle=w_flowcharts;
        get_wind_info(&wib,my_error);
        if(my_error) wimp_error(my_error,WE_OK);
        xco=xco_flowchart(cur_flow);
        yco=yco_boxnum(cur_box);
        if(!(xco>wib.scx && xco<(wib.scx+wib.x1-wib.x0) &&
            yco-100>(wib.scy-wib.y1+wib.y0) && yco<wib.scy)) {
               usr_update_scroll_wind(w_flowcharts,
                                      xco_flowchart(current_flowchart),
                                      yco_boxnum(current_box_num));
            return(TRUE);
        }else return(FALSE);
}


static int oficon(int ihandle)
{

/* returns the item number in cbtext (text of buttons on calculator)
of which ihandle is the icon */

register int i=0;
   while((calcbutton[i]!=ihandle)&&(i<CALC_ROWS*CALC_COLS)) i++;
   if(i<CALC_ROWS*CALC_COLS)return(i);
```

```
   else return(0);
}


static void key_on_boxeditor(univ_block *block)
{

/* The user has pressed a key whilst the pointer is over the  boxeditor */

        flow_box_type this_one;                                         /*post syntax check contents */
        int len,finished=FALSE;
          switch(keycode) {

          case DELETE_KEY :
               if((len=strlen(screen))!=0) screen[len-1]=NULL;
               break;

          case RETURN_KEY :
                finished=TRUE;
                break;

          default :
               if ((keycode<0x80)&&(strlen(screen)<SCREENLENGTH)) {
                         screen[len=strlen(screen)]=keycode;
                         screen[len+1]=NULL;
               }
               break;

          }
          if ((finished)&&(syntax_ok(screen,&this_one)))
                   update_current_flowchart(&this_one);
          else     update_calc_screen();
}


static void mouse_on_boxeditor(univ_block *block)
{

/* User has clicked a mouse button over the boxeditor */

        int finished=FALSE;
        flow_box_type this_one;                                         /*post syntax check contents */
        char *text=cbtext[oficon(mouseicon)];
        switch(mouseb) {

       case L_PRESSED :if(mouseicon!=-1)  {                                          /* SELECT */

/* determine icon, and act appropriately */

                         if(mouseicon==clearbutton) {
                                   *screen=NULL;
                                   reset_caret();
                         }
                         else if(mouseicon==enterbutton) finished=TRUE;
                         else if(strlen(screen)+strlen(text)<SCREENLENGTH)
                         strcat(screen,text);

                         if ((finished)&&(syntax_ok(screen,&this_one)))
```

```
                        update_current_flowchart(&this_one);
                    else update_calc_screen();
                }
                break;

        case M_PRESSED :
/* MENU -- pop up EDIT menu */
                usr_create_menu(&Edit_menu,mousex,mousey,Edit_menu_decode);
                break;

        default :
                break;
        }
}


static void entering_boxeditor(univ_block *block)
{
/* The pointer is about to enter the boxeditor. Set input focus in there,
   or we won't be able to detect any key presses
*/
        input_focus_to(w_boxedit);
}


static void leaving_boxeditor(univ_block *block)
{
/* Pointer leaving the boxeditor. Turn off input focus, so keys no longer go there */
        input_focus_off();
}


static void entering_flows(univ_block *block)
{

/* the pointer is entering the flowcharts window */

        if(is_window_open(w_boxedit)) input_focus_to(w_boxedit);
}


static void leaving_flows(univ_block *block)
{

/* the pointer is leaving the flowcharts window */

        input_focus_off();
}


/*********************************************************************/
/* Graph functions                                                   */
/* High-level routines connected with graph-plotting. The low -      */
/* level ones which actually do the work are in f_plot               */
/*********************************************************************/

void clear_graphs(void)
```

```
{
/* clear all graphs out and close windows */

                clear_graph_index();
                usr_close_wind(w_plotwindow);
                usr_close_wind(w_graphinfo);
}


void graph_window_closing(void)
{
/* unused,this version */
}

static void mouse_on_plotwindow(univ_block *block)
{

/* Button pressed on graph..only menu does anything here */

        switch (mouseb) {

        case L_PRESSED :
                    break;

        case M_PRESSED :
                    usr_create_menu(&Graph_window_menu,
                                    mousex,mousey,
                                    graph_window_menu);

                    break;
        case R_PRESSED :
                    break;
        }
}



static int find_index(int num)
{

/* find if this flowchart is already graphed. If not, return -1 */

        int i=0;
        while(i<nextfree && index[i]!=num) i++;
        return(((index[i]==num)&&(i<nextfree))?i:-1);
}



void do_graph(int chartnum, double xmin, double xmax)
{

/* This routine sets up arrays to be plotted by plotting routines.*/

        static double lastxmin=0,lastxmax=0;
        double xstep ;
        int i,j,error=FALSE;
```

```
        xstep=(xmax-xmin)/NUM_GRAPH_POINTS;          /* number of steps to plot (always NUM_GRAPH_POINTS) */
        if(!(xmin==lastxmin&&xmax==lastxmax)) {

/* user has changed ranges, so replot everything */

            clear_graphs();
            lastxmin=xmin;
            lastxmax=xmax;
        }

        if(nextfree>=MAXGRAPHS) {
            error=TRUE;
            do_error("Too many graphs");
        }

        j=find_index(chartnum);
        if((j==-1)&&(nextfree<MAXGRAPHS)) {
          j=nextfree;
          index[nextfree++]=chartnum;
          i=0;
          while((i<NUM_GRAPH_POINTS)&&(!error)) {
                x[j][i]=xmin+xstep*i;
                y[j][i]=evaluate(x[j][i],chartnum,&flowchart[chartnum],&error);
                i++;
          }
        }
        if (error==FALSE) {
          if(is_window_open(w_plotwindow))
                    usr_update_wind(w_plotwindow);
          else
                    usr_open_wind(w_plotwindow,ON_TOP);
        } else {
          index[--nextfree]=0;
        }
}


static void redraw_plotbox(redraw_block *block)
{

/* This is called by the WIMP when opening plotwindow and when redraws are required */

        int i;

/* assert nextfree>0 at this stage */

        graph_plot(NUM_GRAPH_POINTS,&x[0][0],&y[0][0],W_PLOTBOX_WIDTH,W_PLOTBOX_HEIGHT,curve_colour[0]);
        for(i=1;i<nextfree;i++)  super_plot(NUM_GRAPH_POINTS,&x[i][0],&y[i][0],curve_colour[i]);

}

/*********************************************************************/
/* Panel functions                                                   */
/* The panel is the bar across the bottom with the main program      */
/* icons on it                                                       */
/*********************************************************************/

static void mouse_on_panel(univ_block *block)
```

```
{

/* User has pressed a mouse button on the panel */

/* We want HELP to respond to any button press..user might not know! */

     if(mouseicon==helpicon)
          usr_create_menu(Help_menu,mousex,mousey,help_menu);
     else  switch(mouseb) {
        case L_PRESSED :

/* NB can't use Switch here because RHSs are not constants */

               if (mouseicon==calcicon) {
                        usr_open_wind(w_boxedit,ON_TOP);
                        usr_open_wind(w_flowcharts,ON_TOP);
               }

               else if(mouseicon==runicon) {
                        usr_open_wind(w_runrecord,ON_TOP);
                        usr_open_wind(w_runentry,ON_TOP);
               }

               else if(mouseicon==flowicon) usr_open_wind(w_flowcharts,ON_TOP);
               else if(mouseicon==exiticon) leave_program();
               else if(mouseicon==screensaveicon) system("SCREENSAVE screen");
               break;

        case M_PRESSED :
               usr_create_menu(&Edit_menu,mousex,mousey,Edit_menu_decode);
               break;
        default :
                break;
        }
}

void mouse_elsewhere(univ_block *block)
{
     usr_create_menu(&Edit_menu,mousex,mousey,Edit_menu_decode);
}

/*************************************************************************/
/* INITIALISATION phase   */
/*************************************************************************/

static char* make_sprite_area(int size)
{

/* manufacture a sprite area of size size */

        int *area;
        sp_area=malloc(size+16);
        if(sp_area==NULL) {
              do_error("Not enough room for sprites");
              exit(0);
        }
        area=(int*)sp_area;
        area[0]=size+16;                                    /* total size of sprite area */
```

```
        area[1]=0;                                              /* number of sprites in area */
        area[2]=16;                                               /* offset to first sprite */
        area[3]=16;                                                 /* ...& first free word */
        return((char*)area);
}

static void load_sprites(char *name)
{

/* Load sprites requied by the program ... program expects the sprite file to contain :
   2 x sprites for HELP window, and one for dialog box
*/
        reg_set regs;

        sp_area=make_sprite_area(fextent(name));
        regs.r[0]=256+10;                                   /* load (user) sprite area with sprites*/
        regs.r[1]=(int)sp_area;
        regs.r[2]=(int)name;
        my_error=swix(OS_SpriteOp,&regs);
        if(my_error) wimp_error(my_error,WE_OK);

}


static void clearup(int code)
{

/* Routine used in development..shouldn't be called from now on */

    switch(code) {
    case SIGABRT : do_error("System ABORT : may not be able to recover");
                   break;
    case SIGFPE  : do_error("Floating point error : was FPE loaded ?");
                   break;
    case SIGILL  : do_error("Illegal instruction : program may have crashed");
                    break;
    case SIGSEGV :
    case SIGSTAK : do_error("Unusual system error - may not recover!");
                   break;

    case SIGTERM : do_error("Unusual termination request from user");
                   break;

    }
}

static void set_palette(char *file)
{

/* Grab palette definitions, and incorporate in this display */

        int i;
        FILE *f=fopen(file,"rb");
        if (f)    for(i=0;i<120;i++)  vdu(getc(f));
        fclose(f);
}
```

```
static void set_from_env(int *var, int dflt, char *envvar)
{

/* Set the variable var from envvar using default dflt if not set */

        char *tmp;
        *var=dflt;
        if((tmp=getenv(envvar))!=NULL) {
                if((!strcmp(tmp,"on"))||
                   (!strcmp(tmp,"ON"))) *var=TRUE;
                else if((!strcmp(tmp,"off"))||
                        (!strcmp(tmp,"OFF")))*var=FALSE;
                else do_error("Illegal value for environment variable");
        }
}


static void initialise(void)
{

/* set up most things */

        int i;
        char *screen=malloc(SCREENLENGTH);                    /* calculator screen icon text area*/
        char *tmp;

        *screen=NULL;                                          /* and make C think it's empty */
        newfilename=malloc(60);
        oldfilename=malloc(60);
        *newfilename=NULL;
        *oldfilename=NULL;

#define bit newname_menu.m[0].data.writeable
        bit.text_buff=newfilename;
        bit.val_string=(char*)-1;
        bit.bufflen=60;
#undef bit
        load_sprites(SPRITE_FILE_NAME);                              /* get program sprites */
        errmsg=malloc(100);

        for(i=0;i<=MAX_FLOWS;i++)  {                           /* ensure flowcharts all empty */
               flowchart[i].prev=NULL;
               flowchart[i].next=NULL;
               flowchart[i].op_num=NO_GOOD;
               flowchart[i].box_num=0;
               flowchart[i].dtype=NO_PARM;
        }

        set_taskname("Flow");

        strcpy(runflownum,"1");                                  /* default flowchart number */

/* preference defaults....*/

        signal(SIGABRT,&clearup);
        signal(SIGFPE,&clearup);
        signal(SIGILL,&clearup);
        signal(SIGINT,&clearup);
```

```
        signal(SIGSEGV,&clearup);
        signal(SIGTERM,&clearup);
        signal(SIGSTAK,&clearup);

/* Now set up program from environment variables */

        HELPDIRNAME=malloc(80);

/* where do we find the help files ... */

        if((tmp=getenv("flow$helpdirname"))!=NULL) strcpy(HELPDIRNAME,tmp);
        else strcpy(HELPDIRNAME,D_HELPDIRNAME);

        HELPNAMESTYLE=malloc(80);

/* What do help file names look like ? */

        if((tmp=getenv("flow$helpnamestyle"))!=NULL) strcpy(HELPNAMESTYLE,tmp);
        else strcpy(HELPNAMESTYLE,D_HELPNAMESTYLE);

/* What operations are permitted ? */

        set_op_level=5;
        if((tmp=getenv("flow$oplevel"))!=NULL) set_op_level=atoi(tmp);

        FLOWDIRNAME=malloc(80);
        if((tmp=getenv("flow$flowdirname"))!=NULL) strcpy(FLOWDIRNAME,tmp);
        else strcpy(FLOWDIRNAME,D_FLOWDIRNAME);


        set_from_env(&master_status,FALSE,"flow$master");
        set_from_env (&vars_allowed,FALSE,"flow$namedvars");
        set_from_env(&printing_boxnumbers,TRUE,"flow$boxnumbers");
        set_from_env(&paths_allowed,FALSE,"flow$pathsallowed");
        set_from_env(&advanced_status,FALSE,"flow$advanced");
        set_from_env(&printing_status,FALSE,"flow$printing");

/* angular measure .... */

        if((tmp=getenv("flow$degmode"))!=NULL) {
                if(!strcmp(tmp,"rad")) {
                    degrees=FALSE;
                    prefs_menu(PM_RADS);
                } else  if(!strcmp(tmp,"deg")) {
                    degrees=TRUE;
                    prefs_menu(PM_DEGS);
                }
        } else {
              degrees=TRUE;
              prefs_menu(PM_DEGS);
        }



/* accuracy settings  */

        if((tmp=getenv("flow$acctype"))!=NULL) {
                if(!strcmp(tmp,"sf")) {
```

```
                        accuracy_type=SIG_FIGS;
                        prefs_menu(PM_SIG_FIGS);
                } else if(!strcmp(tmp,"dp")) {
                        accuracy_type=DEC_PLACES;
                        prefs_menu(PM_DEC_PLACES);
                }
        } else {
               accuracy_type=NONE;
               prefs_menu(PM_NO_ACCURACY_SET);
        }

        Accnumber=(char*)malloc(12);

        if(accuracy_type != NONE) {
                if((tmp=getenv("flow$accuracy"))!=NULL) {
                        strcpy(Accnumber,tmp);
                } else strcpy(Accnumber,"3");
        }

        if((tmp=getenv("flow$editmode"))!=NULL) {
                if(!strcmp(tmp,"append")) {
                        edit_mode=APPEND;
                        set_tick_item(EM_APPEND,&Edit_menu);
                }
                else if(!strcmp(tmp,"overwrite")) {
                        edit_mode=OVERWRITE;
                        set_tick_item(EM_OVER,&Edit_menu);
                }
                else if(!strcmp(tmp,"before")) {
                        edit_mode=INSERT_BEFORE;
                        set_tick_item(EM_BEFORE,&Edit_menu);
                }
                else if(!strcmp(tmp,"after")) {
                        edit_mode=INSERT_AFTER;
                        set_tick_item(EM_AFTER,&Edit_menu);

                }
        }  else {
                edit_mode=APPEND;
                set_tick_item(EM_APPEND,&Edit_menu);
        }


        EPSILON=D_EPSILON;
        for(i=0;i<MAX_REC_LINES;i++) {
                *rec_line[i].inpt=NULL;
                *rec_line[i].output=NULL;
                rec_line[i].flownum=0;
        }
}


static void initialise_screen(void)
{

/* Initialise screen appearance  by  opening bottom control panel */

        usr_open_wind(w_panel,ON_TOP);
```

```
        move_mouse_to(250,50);
}



static void initialise_functions(void)
{

/* tell the WIMP shell about which function does which job ...
   This routine sets up the functions which are called in response to
   the window manager's requests for jobs to be done
*/

/* background */

        background_mouse_function(mouse_elsewhere);

/* 1) Flowcharts window : */

        mse_pressed_function(w_flowcharts,mouse_on_flowcharts);
        window_redraw_function(w_flowcharts,redraw_flowcharts);
        wind_enter_function(w_flowcharts,entering_flows);
        wind_leave_function(w_flowcharts,leaving_flows);

/* 2) plot window...*/

        mse_pressed_function(w_plotwindow,mouse_on_plotwindow);
        window_redraw_function(w_plotwindow,redraw_plotbox);
        close_window_function(w_plotwindow,graph_window_closing);

/* 3) Box Editor window : */

        mse_pressed_function(w_boxedit,mouse_on_boxeditor);
        key_pressed_function(w_boxedit,key_on_boxeditor);
        wind_enter_function(w_boxedit,entering_boxeditor);
        wind_leave_function(w_boxedit,leaving_boxeditor);
        window_redraw_function(w_boxedit,redraw_boxeditor);

/* 4) control panel : */

        mse_pressed_function(w_panel,mouse_on_panel);


/* 5) help window : */

        window_redraw_function(w_help,redraw_help);
        mse_pressed_function(w_help,mouse_on_help);


/* 6) Run Entry window : */

         window_redraw_function(w_runentry,redraw_runentry);
         wind_enter_function(w_runentry,mouse_enters_runentry);
         wind_leave_function(w_runentry,mouse_leaves_runentry);
         key_pressed_function(w_runentry,key_on_runentry);
         mse_pressed_function(w_runentry,mse_on_runentry);

/* 7) Run record window : */
```

```
        wind_enter_function(w_runrecord,mouse_enters_runrecord);
        wind_leave_function(w_runrecord,mouse_leaves_runrecord);
        window_redraw_function(w_runrecord,redraw_runrecord);
        key_pressed_function(w_runrecord,key_on_runrecord);
        mse_pressed_function(w_runrecord,mse_on_runrecord);

/* 8) Graph window */

        window_redraw_function(w_graphinfo,redraw_graphinfo);

/* Make graph window option INFO open onto info window ...*/

        Graph_window_menu.m[GWM_INFO].sub_menu=(menu_block*)w_graphinfo;

/* 9) Varbox window */

        window_redraw_function(w_vartable,redraw_vartable);
        mse_pressed_function(w_vartable,mouse_on_vartable);

/* temporary pop-up of variables */

        Edit_menu.m[EM_VARS].sub_menu=(menu_block*)w_vartable;
}

/*********************************************************************/
/* EXIT  functions  */
/*********************************************************************/


void leave_program(void)
{

/* Come here if user requested to go */

        int reply=wimp_decision("Are you sure you want to leave?");
        if(reply==1)  {
           close_wimp();
           mode(12);
           puts("Completed ...");
           exit(0);
        }
}


static void redefine_characters(void)
{


/* The square root and superscript x are not part of the
   standard Archimedes character set. Redefine two unused ones
   to do the job.

     Also, set up line style for dotted lines

*/

        static int chardefs[30] = {
```

```
                23,0xBE,3,3,6,6,118,28,12,0,                                    /* sq rt */
                23,0xBF,144,96,96,144,0,0,0,0,                                  /* x superscr */
                23,6,0xF0,0,0,0,0,0,0,0                                         /* dot-dash line style */
            );
        int i;
        for(i=0;i<30;i++) vdu(chardefs[i]);

}

static pshape_block dflt = {                                                    /* Pointer shape ... */
                1,(char*)-1,12,12,0,12
                };


/*********************************************************/
/************** M A I N    R O U T I N E ****************/
/*********************************************************/

int main(int argc, char *argv[])
{

        char *tmp;
        Hg_On();                                                        /* turn the hourglass on */
        initialise();                                                     /* set up most things */
        mode(12);                                                       /* Set up graphics mode */
        redefine_characters();                                               /* ...characters ! */
        set_palette("<obey$dir>.!palette");                                       /* ...palette */
        open_wimp(grey4);                                                        /* ..more wimp */
        set_point_shape(&dflt,my_error);                                        /*..pointer shape */
        if(my_error) wimp_error(my_error,WE_OK);
        setup_winds();                                                          /*...FLOW windows */
        initialise_functions();                                                    /*...functions */
        initialise_screen();                                          /* ...screen appearance initially */
        clear_graphs();

/* START UP file load :
   clicking on a flowchart icon, or a command line spec takes precedence,
else use the file spec'd in the !run file, if any. If none there, just load
up with nothing in
*/
        if(argc>1) load_flowcharts(argv[1]);
        else if((tmp=getenv("flow$autoload"))!=NULL) load_flowcharts(tmp);

/* The main program now drops into the WIMP polling loop */
/* ..and in fact will never return from it */

        Hg_Off();                                                               /* turn hourglass off */
        system("POINTER 1");                                                 /* ensure the pointer is ON */
        poll(0);
        return(0);                                                          /*..... a programming "niceity" */
}
```

9.4.6 C.F_OPS

```
/* >c.f_ops


@@@@@@@          @@@@@@@ @@@@@@   @@@@@
@                @     @ @     @ @     @
@                @     @ @     @ @
@@@@@            @     @ @@@@@@   @@@@@
@                @     @ @              @
@                @     @ @       @      @
@       @@@@@@@ @@@@@@@ @        @@@@@

   operations, definition and parsing routines for operation boxes,

*/
#include <errno.h>
#include "flow.h"                                          /* in which all other includes defined */


/* tokens used by tokenise to tokenise the input buffer */

/* for the moment... */

#define NOT_TOKEN -1

#define PI 3.1415926535

/* some error messages which crop up frequently...*/

static char
     *undefmem="This memory is not defined",
     *nosuchvar="No such variable",
     *invalidnum="The number is invalid",
     *varexists="This variable exists already",
     *divbyzero="Can't divide by 0",
     *reciperror="Can't take reciprocal of 0",
     *logrange="Can't take log of numbers <=0",
     *sqrtrange="Can't take Square root of negative numbers",
     *atrigrange="ASN or ACS can only have inputs in range -1 to 1",
     *tanrange="Cannot evaluate TAN near 90 degrees",
     *namenotnumerr="Must be a name,not a number",
     *recurserr="Cannot call the same flowchart",
     *udzpz="0 to the power 0 is undefined",
     *dom="Outside the range which this function can evaluate",
     *erange="The result is too large to stored by FLOW";

char outbuffer[50];

variable var[NUM_VARS];                                    /* named variables stored here */
rrec rec_line[MAX_REC_LINES+1];                             /* run record saved in these */

static double memory;                                       /* anonymous memory store */

static int memory_set=FALSE,                               /* the anonymous memory is defined */
           dp_gone=FALSE,                                 /* decimal point is in numinput string */
           last_was_fkey=FALSE,                           /* last key on run areas was a function key */
           next_var_free=0;                               /* next variable free in variable array */
```

```
int        autoswap=FALSE;                    /* Boolean switch, either swap in-out automatically, or not */

/********* Prototype functions, this block *********/

static char *print_dp(double number,int precision),
            *print_sf(double number,int precision);

void       key_on_runentry(univ_block *),
           read_dbl(char *buf,double *number),
           store (double val,char *ident),
           clearrunrecord(void);

op_descrip operation[NUM_OPS] = {
                                                                        /* level, number of parameters */

/* operation[TOK_xxx-0x80].level gives level,
 * operation[TOK_xxx-0x80].num_params gives number of parms
 * operation[TOK_xxx-0x80].label gives output description
 * ..Must be kept in same order as TOK_ defines in order to preserve above
 */
                {1,1,"+ "},                                                          /* plus */
                {1,1,"- "},                                                         /* minus */
                {1,1,"x "},                                                         /* times */
                {1,1,"/ "},                                                        /* divide */

                {2,0,"1/x"},                                                   /* reciprocal */
                {2,0,"√x"},                                                          /*sqrt */
                {2,1,"^ "},                                                         /* power */

                {3,0,"SIN"},                                                         /* sin */
                {3,0,"COS"},                                                         /* cos */
                {3,0,"TAN"},                                                         /* tan */
                {3,0,"ASN"},                                                         /* asn */
                {3,0,"ACS"},                                                         /* acs */
                {3,0,"ATN"},                                                         /* atn */

                {3,0,"LOG"},                                                         /* LOG */
                {3,0,"LN"},                                                           /* LN */
                {3,0,"EXP"},                                                        /* exp */
                {3,0,"10↑"},                                                       /* 10^x */

                {4,1,"STO "},                                              /* TOK_STO Store in memory */
                {4,1,"RCL "},                                             /* TOK_RCL Recall from memory */

                {4,0,"+RCL "},
                {4,0,"-RCL "},
                {4,0,"xRCL "},
                {4,0,"/RCL "},

                {1,0,"+/- "},

                {5,0,"SINH "},
                {5,0,"COSH "},
                {5,0,"TANH "},
                {5,1,"F "},

                {2,1,"^1/ "},                                                /* root, late addition! */
```

```
                {4,1,"STO+ "},                                  /* add to variable */
                {4,1,"STO- "},                                /* subtract from variable */
                {4,1,"STO* "},                                 /* multiply by variable */
                {4,1,"STO/ "},                               /* divide variable by number */
               };


#define NUM_POSS 44

/* NUM_OPS plus duplicates , e.g. STO & =, * and x */

static op_tokens  token[NUM_POSS] = {


               {"+RCL",TOK_PLUSRCL},
               {"-RCL",TOK_MINUSRCL},
               {"xRCL",TOK_TIMESRCL},
               {"*RCL",TOK_TIMESRCL},                                  /* alias for xRCL */
               {"/RCL",TOK_DIVIDERCL},

               {"+/-",TOK_CHS},
               {"CHS",TOK_CHS},                                        /* alias for +/- */

               {"+",TOK_PLUS},
               {"-",TOK_MINUS},
               {"*",TOK_TIMES},
               {"x",TOK_TIMES},
               {"/",TOK_DIVIDE},


               {"1/x",TOK_RECIP},

               {"√x",TOK_SQRT},
               {"SQRT",TOK_SQRT},                                  /* typing alias for √x */

               {"^1/",TOK_ROOT},

               {"POW ",TOK_POW},
               {"y^x",TOK_POW},
               {"^",TOK_POW},

               {"SINH",TOK_SINH},
               {"COSH",TOK_COSH},
               {"TANH",TOK_TANH},

               {"SIN",TOK_SIN},
               {"COS",TOK_COS},
               {"TAN",TOK_TAN},
               {"ASN",TOK_ASN},
               {"ACS",TOK_ACS},
               {"ATN",TOK_ATN},

               {"LOG",TOK_LOG},
               {"LN",TOK_LN},

               {"EXP",TOK_EXP},
               {"e^x",TOK_EXP},                                        /* alias */
```

```
            {"10^x",TOK_TENP},                                          /* alias */
            {"10↑",TOK_TENP},

            {"STO+",TOK_PSTO},
            {"STO-",TOK_TSTO},
            {"STO*",TOK_MSTO},
            {"STOx",TOK_MSTO},
            {"STO/",TOK_DSTO},

            {"STO",TOK_STO},
            {"=",TOK_STO},
            {"RCL",TOK_RCL},
                                                    /* +RCL collected through +, etc */
            {"F",TOK_SUBFLOW},
            {"ENTER",TOK_ENTER},

        };


void stripspaces(char *buf)
{
/* remove spaces from the string in buf, and terminate with END_OF_STRING
*/

        int i=0,j=0;
        do {
           if(buf[i]!=' ') buf[j++]=buf[i++];
           else i++;
        } while(buf[i]);                                                /* is not NULL */
        buf[j]=END_OF_STRING;
}


static int is_token(char *buf)
{

/* return token number, or NOT_TOKEN */

     int k=0,reply=NOT_TOKEN;
     while((k<NUM_POSS)&&(reply==NOT_TOKEN)) {
       if(strncmp(buf,token[k].text,strlen(token[k].text))==0) reply=k;
       k++;
     }
     return(reply);
}


static int tokenise(union data *dta, char *buffer, char *out)
{
        /* tokenise the input string, returning any non-unit found
           to the caller */

        int i=0,j=0,numread,reply;
        char *retvarname;
        int parm_type=NO_PARM;

        while(buffer[i]==' ') i++;                                      /* skip leading spaces */
```

```
if((reply=is_token(buffer+i))!=NOT_TOKEN) {  /* check first place, which may be a token from table */
                out[j++]=token[reply].token;
                i+=strlen(token[reply].text);
}

while(buffer[i]==' ') i++;

until(buffer[i]==END_OF_STRING) {

     if(isdigit(buffer[i])||buffer[i]=='.') {

                out[j++]=TOK_NUMBER;
                sscanf(buffer+i,"%le%n",&(dta->param1),&numread);
                i+=numread;
                parm_type=NUMBER;
                continue;

    }

    reply=is_token(buffer+i);

    if((reply!=NOT_TOKEN)&&(token[reply].token!=TOK_TIMES)) {
                out[j++]=token[reply].token;
                i+=strlen(token[reply].text);
                continue;
    }

    if(isalpha(buffer[i])) {
                if(j==0) {                                          /* variable on own=RCL */
                  out[j++]=TOK_RCL;
                }
                out[j++]=TOK_VAR;
                retvarname=buffer+i;
                dta->varname=malloc(strlen(retvarname)+1);
                if(dta->varname==NULL) {
                       do_error("No room left!");
                } else {
                       strcpy(dta->varname,retvarname);
                       i+=strlen(retvarname);
                       parm_type=VARIABLE;
                }
                buffer[i]=END_OF_STRING;
                continue;
     }
     if(buffer[i]==' ') i++;
     out[j++]=buffer[i++];
}

out[j]=END_OF_STRING;

                                                    /* deal with RCL with no variable name */

if(out[1]==TOK_RCL) {
     out[0]+=TOK_PLUSRCL-TOK_PLUS;
     out[1]=END_OF_STRING;
     parm_type=NO_PARM;
}
```

```
                                                    /* deal with leading minus to numeric parameter */

        if(out[1]==TOK_MINUS&&out[2]==TOK_NUMBER) {
            int i=1;
            while(out[i]!=END_OF_STRING) {
                 out[i]=out[i+1];
                 i++;
            }
            dta->param1=-dta->param1;
        }

        return(parm_type);
}


static int check_syntax(char *buf)
{                                       /* buf is a tokenised rep'n of the input with the number removed */

/* Return either the token of the operator involved, OR
   NO_GOOD if the string does not conform to the syntax

    This is hard-coded and hand-coded because it is a very
  simple syntax, and not difficult to code by hand
*/

        int checkout,token=(int)*buf;
        if (((op_num_params(token)==0)&&
            (buf[1]==END_OF_STRING))
             ||
            (((op_num_params(token)==1)&&
             ((int)buf[1]==TOK_NUMBER)&&
             (buf[2]==END_OF_STRING)))
             ||
            (((op_num_params(token)==1)&&
             (((int)buf[1]==TOK_VAR)||((int)buf[1]==TOK_RCL))&&
             (buf[2]==END_OF_STRING)))
             ||
            (token==TOK_STO)                                        /* these may have 0 params */
             ||
            (token==TOK_RCL))   {
                    int i=0;
                    checkout=TRUE;
                                                                    /* check for named variables */
                    while((buf[i]!=END_OF_STRING)&&(buf[i]!=TOK_VAR)) i++;
                       if(buf[i]==TOK_VAR && !vars_allowed) checkout=FALSE;
                                                                    /* check for reserved words as variables */
                   if(((token==TOK_STO)||(token==TOK_RCL))&&
                     (buf[1]!=TOK_VAR)&&
                     (buf[1]!=END_OF_STRING))  checkout=FALSE;
        } else      checkout=FALSE;

        if((checkout)&&(op_level(token)>set_op_level)) {
                checkout=FALSE;
                return(NOT_PERMITTED);
        } else  return(checkout?token:NO_GOOD);
}
```

```
static int is_op(char ch)
{

/* determine whether the given token is an operation */

        return(  (ch==TOK_PLUS)||(ch==TOK_MINUS)
               ||(ch==TOK_DIVIDE)||(ch==TOK_TIMES)
               ||(ch==TOK_POW) );
}


static int lotsa_ops(char *b)
{
                                /* check if any operations found in b , true if yes, false if none found */

        int ret=FALSE,i=0;
        do
            if(is_op(b[i++])) ret=TRUE;
        while((!ret)&&(b[i]!=END_OF_STRING));
        return(ret);
}


static void construct_error_message(char *message, char *tb, char *b)
{

/* tb is the tokenised buffer, look for certain obvious problems
   and report those if found, else just report that there is a problem
   b is the original text buffer
*/

        if(strlen(b)==0) {
                strcpy(message,"There must be something in the box");
        }

        else if(tb[0]==TOK_NUMBER) {
                strcpy(message,"Number must have an operation before it");
        }

        else if(((tb[0]==TOK_STO)||(tb[0]==TOK_RCL))&&
                     (tb[1]!=TOK_VAR)&&
                     (tb[1]!=END_OF_STRING)) {
                strcpy(message,"Reserved word used as variable name");
        }

        else if((op_num_params(tb[0])>0) && tb[1]==END_OF_STRING) {
                strcpy(message,"Box needs a number (or variable) after the operation");
        }
        else if(lotsa_ops(tb+1)) {
                strcpy(message,"Only one +,-,x, / or ^ allowed");
        }

        else    {
                strcpy(message,"FLOW doesn't recognise \"");
                strcat(message,b);
                strcat(message,"\"");
        }
}
```

```
int syntax_ok(char *buffer, flow_box_type *box)
{

/* returns TRUE if syntax checks out as a valid buffer, else returns FALSE,giving the user some guide as to
   the probable source of error in err_report
*/

        int retval;
        char *err_report=malloc(50);
        box->dtype=tokenise(&box->data,buffer,outbuffer);
        if((outbuffer[0]<=TOK_DIVIDE)&&(outbuffer[1]==TOK_RCL)) {
                                                        /* deal with RCL with no variable name */
                box->data.varname=malloc(4);
                strcpy(box->data.varname,"RCL");
        }
        box->next=NULL;
        box->op_num=check_syntax(outbuffer);
        if( (box->op_num) > NO_GOOD) retval=TRUE;

        else {
             retval=FALSE;
             switch (box->op_num) {
             case NO_GOOD :
                   construct_error_message(err_report,outbuffer,buffer);
                   break;
             case NOT_PERMITTED :
                  sprintf(err_report,"FLOW doesn't recognise \"%s\"",buffer);
                  break;
             }
             do_error(err_report);
        }
        return(retval);
}

static double d_to_r(double x)
{

/* Convert x degrees to radians */

        return(x*PI/180);
}

static double r_to_d(double x)
{

/* convert x radians to degrees */

        return(x*180/PI);
}

/*****************************************************/
/*  VARIABLE    routines                             */
/*****************************************************/

static int var_exists(char *varname, int *index)
{
```

```
/* return TRUE if variable exists, index in i, else return FALSE, index undefined */
        int i=0;
        while((i<next_var_free) && (strcmp(var[i].name,varname)!=0)) i++;
        *index=i;
        return((strcmp(var[i].name,varname)==0)&&(var[i].status==V_OK));
}

static void remove_variable(univ_block *block)
{
        int i,j;
        if(var_exists(del_var_name,&i)) {

                        /*this line needed in case last variable in list, and not then  overwritten! ...*/

              var[i].status=V_UNINIT;
              for(j=i;j<next_var_free-1;j++)  var[j]=var[j+1];
              next_var_free--;
              usr_update_wind(w_vartable);
        } else do_error(nosuchvar);
}


static void create_variable_menu(univ_block *block, int newvar)
{

/* newvar==TRUE => we want to create new variable, newvar==FALSE=> We are modifying an existing one */

        int d,exists,eos;
        double value;
        sscanf(crtvar_val,"%Lf%n",&value,&eos);
        if(eos!=strlen(crtvar_val)) {
               do_error(invalidnum);
               return;
        }
        exists=var_exists(crtvar_name,&d);                                        /* d is dummy */
        if (exists && newvar) {
             do_error(varexists);
        } else if(!exists && !newvar) {
             do_error(nosuchvar);
        } else switch(item(0)) {
        case 0 :
                  store(value,crtvar_name);
                  usr_update_wind(w_vartable);
                  *crtvar_val=NULL;
                  *crtvar_name=NULL;
        default : break;
        }
}

static void new_vars_menu(univ_block *block, int newvar)
{
        switch(item(0)) {
        case 0 :
                  create_variable_menu((univ_block*)&block->a[1],newvar);
        default : break;
        }
}
```

```
static void vars_menu_decode(univ_block *block)
{

/* Mouse click on vars menu */

        switch(item(0)) {
        case VM_CLEAR :
                        clear_variables();
                        break;
        case VM_DELETE:
                        remove_variable((univ_block*)&block->a[1]);
                        break;

        case VM_CREATE:
                        new_vars_menu((univ_block*)&block->a[1],TRUE);
                        break;
        case VM_ADJUST:
                        new_vars_menu((univ_block*)&block->a[1],FALSE);
                        break;
        }
}

void mouse_on_vartable(univ_block *block)
{

/* The mouse pointer is on the variable table window */
/* react only to menu button */

        switch(mouseb) {

        case R_PRESSED :
        case L_PRESSED : break;

        case M_PRESSED :
                        usr_create_menu(&Vars_menu,mousex,mousey,vars_menu_decode);
                        break;

        }
}

void redraw_vartable(redraw_block *block)
{

/* Redraw the new version of the variable table */

        int i,line=0;
        move(0,W_VARBOX_HEIGHT-10);
        gcol(0,grey2);
        if(next_var_free==0) {
               puts("No named variables");
               line++;
        }
        else for(i=0;i<next_var_free;i++) if (var[i].status==V_OK) {
               gcol(0,grey2);
               move(0,W_VARBOX_HEIGHT-10-(line++)*CHAR_HEIGHT);
               printf("%15s  = ",var[i].name);
               gcol(0,white);
               printf("%f",var[i].value);
```

```
        }
        gcol(0,grey2);
        move(0,W_VARBOX_HEIGHT-10-line*CHAR_HEIGHT);
        if(memory_set) printf("Calculator store = %f\n",memory);
        else puts("Calculator store unset");
}

void clear_variables(void)
{

/* remove all trace of all variables */

        int i;
        for(i=0;i<NUM_VARS;i++) var[i].status=V_UNINIT;
        next_var_free=0;
        memory_set=FALSE;
        usr_update_wind(w_vartable);
}


void store (double val, char *ident)
{
/* store value val in the variable name ident*/
        int i=0;

/* find the variable name... */

        if(strlen(ident)>14) do_error("Variable name too long- truncated");
        if (var_exists(ident,&i)) {
                var[i].value=val;
                var[i].status=V_OK;
        } else  {                                                       /* new variable */
                if(next_var_free<NUM_VARS) {
                   var[i].value=val;
                   var[i].status=V_OK;
                   strncpy(var[i].name,ident,14);
                   var[i].name[14]=(char)0;
                   next_var_free++;
                } else {
                   do_error("No more room for new variables!");
                }
        }
}


static void run_time_error(char *errmess, int fnum, int boxnum,int *errflag)
{

/* an error has occured whilst the chart was running */

        char mine[120];
        sprintf(mine,"%s at box %d",errmess,boxnum);
        do_error(mine);
        current_box_num=boxnum;
        current_flowchart=fnum;
        usr_update_scroll_wind(w_flowcharts,xco_flowchart(current_flowchart),yco_boxnum(current_box_num)-100);
        *errflag=TRUE;
}
```

```
static double varval(flow_box_type *box, int fnum, int *evalerror)
{

/* Return the value attached to varname, or error if not found */

        int i;
        double ret=1;                                   /* Must be non-zero for divide call in evaluate */
        if(var_exists(box->data.varname,&i)) ret=var[i].value;
        else run_time_error(nosuchvar,fnum,box->box_num,evalerror);
        return(ret);
}

static double check_range(double n, double lower, double upper,int fnum, int bnum, int *evalerror)
{
        if(n<=lower||n>=upper) {
              run_time_error(dom,fnum,bnum,evalerror);
              n=0;
        }
        return(n);
}

static double check_bounds(double n, int fnum, int bnum,int *evalerror)
{
        double res=n;
        if(errno!=0) switch(errno)  {
            case(EDOM) : run_time_error(dom,fnum,bnum,evalerror);
                         res=0;
                         break;
            case(ERANGE):run_time_error(erange,fnum,bnum,evalerror);
                         res=0;
                         break;
        }
        else if (fabs(n) > HUGE_VAL) {
            run_time_error(erange,fnum,bnum,evalerror);
            res=0;
        }
        return(res);
}


double evaluate(double input, int fnum, flow_box_type *box,
                int *evalerror)
{
/*returns the result of passing input through the boxes
  chained to box, until an end is reached. This is the core routine
  of the whole program!
*/

#define bnum box->box_num
double result=input,param;

     *evalerror=FALSE;
     if(fnum<0 || fnum>MAX_FLOWS) run_time_error("No such chart",fnum,bnum,evalerror);
     while((box!=NULL)&&(!(*evalerror))) {

        if((op_num_params(box->op_num)>0)&&(box->op_num!=TOK_STO))
```

```
        switch(box->dtype) {
                        case NUMBER :
                              param=box->data.param1;
                              break;
                        case VARIABLE :
                              param=varval(box,fnum,evalerror);
                              break;
        }

      if (!(*evalerror)) switch(box->op_num) {

        case TOK_PLUS :
                         result=input+param;
                         break;
        case TOK_MINUS:  result=input-param;
                         break;
        case TOK_TIMES:
                         result=input*param;
                         break;

        case TOK_DIVIDE:
                        if (param!=0) result=input/param;
                        else run_time_error(divbyzero,fnum,bnum,evalerror);
                        break;

       case TOK_RECIP:  if(input!=0) result=1/input;
                        else(run_time_error(reciperror,fnum,bnum,evalerror));
                        check_bounds(result,fnum,bnum,evalerror);
                        break;

       case TOK_SQRT:   if (input>=0) result=sqrt(input);
                        else {
                            run_time_error(sqrtrange,fnum,bnum,evalerror);
                            result=0;
                        }
                        check_bounds(result,fnum,bnum,evalerror);
                        break;

       case TOK_POW:
                      if((input==0)&&(param==0)) {
                         run_time_error(udzpz,fnum,bnum,evalerror);
                         break;
                      } else {
                            result=pow(input,param);
                            result=check_bounds(result,fnum,bnum,evalerror);
                       }
                       break;

       case TOK_ROOT :
                      if((input==0)&&(param==0)) {
                         run_time_error(udzpz,fnum,bnum,evalerror);
                         break;
                      } else {
                         result=pow(input,1/param);
                         result=check_bounds(result,fnum,bnum,evalerror);
                      }
                      break;
```

```
case TOK_SIN :  if(degrees) input=d_to_r(input);
                input=check_range(input,-5000.0,5000.0,fnum,bnum,evalerror);
                result=sin(input);
                check_bounds(result,fnum,bnum,evalerror);
                break;

case TOK_COS :  if(degrees) input=d_to_r(input);
                input=check_range(input,-5000.0,5000.0,fnum,bnum,evalerror);
                result=cos(input);
                check_bounds(result,fnum,bnum,evalerror);
                break;

case TOK_TAN : {
                double rembit,modbit;
                if(degrees)  input=d_to_r(input);
                rembit=modf(input/PI,&modbit)*PI;
                if(fabs(rembit-PI/2)<1E-4) {
                        result=500;
                        run_time_error(tanrange,fnum,bnum,evalerror);
                } else  result=tan(input);                                  /* radians */
                check_bounds(result,fnum,bnum,evalerror);
                break;
               }
case TOK_ASN :
                if(fabs(input)>1) {
                            run_time_error(atrigrange,fnum,bnum,evalerror);
                     } else {
                        if(degrees) result=r_to_d(asin(input));
                        else result=asin(input);
                        check_bounds(result,fnum,bnum,evalerror);
                     }
                break;

case TOK_ACS :  if(fabs(input)>1) {
                    run_time_error(atrigrange,fnum,bnum,evalerror);
                } else {
                    if(degrees) result=r_to_d(acos(input));
                    else result=acos(input);
                }
                break;

case TOK_ATN :  if(degrees) result=r_to_d(atan(input));
                else result=atan(input);
                check_bounds(result,fnum,bnum,evalerror);
                break;

case TOK_LOG :  if(input>0) {
                  result=log10(input);
                  check_bounds(result,fnum,bnum,evalerror);
                } else run_time_error(logrange,fnum,bnum,evalerror);
                break;

case TOK_LN :   if(input>0) {
                   result=log(input);
                   check_bounds(result,fnum,bnum,evalerror);
                } else run_time_error(logrange,fnum,bnum,evalerror);
                break;
```

```
case TOK_EXP :  result=exp(input);
                check_bounds(result,fnum,bnum,evalerror);
                break;

case TOK_TENP :
                if(fabs(input)>27) {
                        run_time_error(erange,fnum,bnum,evalerror);
                        result=0;
                } else  {
                        result=pow(10,input);
                        check_bounds(result,fnum,bnum,evalerror);
                }
                break;


case TOK_PLUSRCL:
                if(memory_set) result=result+memory;
                else run_time_error(undefmem,fnum,bnum,evalerror);
                break;

case TOK_MINUSRCL:
                if(memory_set) result=result-memory;
                else run_time_error(undefmem,fnum,bnum,evalerror);
                break;

case TOK_TIMESRCL:
                if(memory_set) result=result*memory;
                else run_time_error(undefmem,fnum,bnum,evalerror);
                break;

case TOK_DIVIDERCL:
                if(memory_set) {
                   if(memory!=0) result=result/memory;
                   else run_time_error(divbyzero,fnum,bnum,evalerror);
                }
                else run_time_error(undefmem,fnum,bnum,evalerror);
                check_bounds(result,fnum,bnum,evalerror);
                break;


 case TOK_STO :  switch(box->dtype) {
                 case NO_PARM :
                          memory_set=TRUE;
                          result=memory=input;
                          break;

                 case VARIABLE :
                          store(input,box->data.varname);
                          break;

                 case NUMBER :
                          run_time_error(namenotnumerr,fnum,bnum,evalerror);
                          result=memory=input;
                          break;
                  }
                  break;
```

```
case TOK_PSTO :  switch(box->dtype) {
                 case VARIABLE :
                          store(input+param,box->data.varname);
                          break;

                 case NO_PARM :
                 case NUMBER :
                          run_time_error(namenotnumerr,fnum,bnum,evalerror);
                          break;
                 }
                 result=input;
                 break;

case TOK_MSTO :  switch(box->dtype) {
                 case VARIABLE :
                          store(input*param,box->data.varname);
                          break;

                 case NO_PARM :
                 case NUMBER :
                          run_time_error(namenotnumerr,fnum,bnum,evalerror);
                          break;
                 }
                 result=input;
                 break;

case TOK_TSTO :  switch(box->dtype) {
                 case VARIABLE :
                          store(param-input,box->data.varname);
                          break;

                 case NO_PARM :
                 case NUMBER :
                          run_time_error(namenotnumerr,fnum,bnum,evalerror);
                          break;
                 }
                 result=input;
                 break;


case TOK_DSTO :  switch(box->dtype) {
                 case VARIABLE :
                          if(input!=0) store(param/input,box->data.varname);
                          else run_time_error(divbyzero,fnum,bnum,evalerror);
                          break;

                 case NO_PARM :
                 case NUMBER :
                          run_time_error(namenotnumerr,fnum,bnum,evalerror);
                          break;
                 }
                 result=input;
                 break;

case TOK_VAR :
case TOK_RCL :  switch(box->dtype) {
                case NO_PARM :
```

```
                        if(memory_set) result=memory;
                        else run_time_error(undefmem,fnum,bnum,evalerror);
                        break;

                    case VARIABLE :
                              result=param;
                              break;

                    case NUMBER :
                              break;
                    }
                    break;


          case TOK_CHS :
                    result=-input;
                    break;

          case TOK_SINH :
                    result=sinh(input);
                    check_bounds(result,fnum,bnum,evalerror);
                    break;

          case TOK_COSH :
                    result=cosh(input);
                    check_bounds(result,fnum,bnum,evalerror);
                    break;

          case TOK_TANH :
                    result=tanh(input);
                    check_bounds(result,fnum,bnum,evalerror);
                    break;

          case TOK_SUBFLOW : {
                    int sub=(int)param;
                    if(sub==fnum) {
                      run_time_error(recurserr,fnum,bnum,evalerror);
                      result=0;
                    }
                    else                                    /* recursive call to this routine */
                      result=evaluate(input,sub,&flowchart[sub],evalerror);
                   }
                   break;
               default:
                   break;
        }
        input=result;
        box=box->next;
      }
      return(result);
}

static void run_flowchart(int flownumber, char *input, char *output,int *err)
{

/* run the flowchart, taking input from input, and placing the results in  output */

        double in,out;
```

```
        int precision=0;
        char buffer[30];

        read_dbl(input,&in);

        *err=FALSE;
        out=evaluate(in,flownumber,&flowchart[flownumber],err);

        sscanf(Accnumber,"%u",&precision);

        if(precision>7) {
              do_error("Accuracy <=7 please!\nsetting to 7");
              precision=7;
              strcpy(Accnumber,"7");
        }

        if(precision<1) {
              do_error("Accuracy >=1 please!\nsetting to 1");
              precision=1;
              strcpy(Accnumber,"1");
        }
        if (*err) output="Error";
        else switch(accuracy_type) {
          case SIG_FIGS :  strncpy(buffer,print_sf(out,precision),30);
                           strncpy(output,buffer,19);
                           if(strlen(buffer)>19) output[18]='\x89';
                           output[19]=NULL;
                           break;

          case DEC_PLACES:
                           strncpy(buffer,print_dp(out,precision),30);
                           strncpy(output,buffer,19);
                           if(strlen(buffer)>19) output[18]='\x89';
                           output[19]=NULL;
                           break;

          default :        sprintf(buffer,"%f",out);
                           strncpy(output,buffer,19);
                           if(strlen(buffer)>19) output[18]='\x89';
                           output[19]=NULL;

        }
        usr_update_wind(w_vartable);
}


/*******************************************************/
/*      Running a flowchart functions                  */
/*                                                     */
/*******************************************************/

void mouse_enters_runentry(univ_block *block)
{

/* Mouse is entering the runentry box..set up input focus,  so can detect keys */

        input_focus_to(w_runentry);
}
```

```
void mouse_leaves_runentry(univ_block *block)
{
/* Turn off input focus..no longer detect keys */
        input_focus_off();
}

void mouse_enters_runrecord(univ_block *block)
{

/* If mouse in runrecord, then runentry needs the focus..can't type anything into runrecord */

        if(is_window_open(w_runentry)) input_focus_to(w_runentry);
}

void mouse_leaves_runrecord(univ_block *block)
{
        input_focus_off();
}


static int is_function_key(int code)
{

/* determine whether the key pressed is a RED function key */

static struct dc { int key,flownum;
          } fkeycodes[] = {  {FN_1,1},
                             {FN_2,2},
                             {FN_3,3},
                             {FN_4,4},
                             {FN_5,5},
                             {FN_6,6},
                             {FN_7,7},
                             {FN_8,8},
                             {FN_9,9},
                             {FN_10,10},
                             {FN_11,11},
                             {FN_12,12},
/* shift   = +10 */
                             {S_FN_1,11},
                             {S_FN_2,12},
                             {S_FN_3,13},
                             {S_FN_4,14},
                             {S_FN_5,15},
                             {S_FN_6,16},
                             {S_FN_7,17},
                             {S_FN_8,18},
                             {S_FN_9,19},
                             {S_FN_10,20},
                             {S_FN_11,21},
                             {S_FN_12,22},
/* control = +20 */
                             {C_FN_1,21},
                             {C_FN_2,22},
                             {C_FN_3,23},
                             {C_FN_4,24},
```

```
                        {C_FN_5,25},
                        {C_FN_6,26},
                        {C_FN_7,27},
                        {C_FN_8,28},
                        {C_FN_9,29},
                        {C_FN_10,30}
                      };
        int i=0;
        while(i<34&&fkeycodes[i].key!=code) i++;
        if(i==34) return(NO_GOOD);
        else      return(fkeycodes[i].flownum);
}



/*********************************************/
/* Routines dealing with runrecord and       */
/* RunEntry windows                          */
/*                                           */
/*********************************************/

static void update_record(char *in, int op, char *out)
{
/* A flowchart has been run. Update the runrecord with the "transaction"
 ...in, out are input and output of flowchart # op
*/

        int i,acc;
                                                                    /* and shift the rest up.. */
        for(i=0;i<MAX_REC_LINES;i++) rec_line[i]=rec_line[i+1];
        sscanf(Accnumber,"%d",&acc);
        strcpy(rec_line[MAX_REC_LINES].inpt,in);
        rec_line[MAX_REC_LINES].flownum=op;
        switch (accuracy_type) {
        case SIG_FIGS :
                  sprintf(rec_line[MAX_REC_LINES].output,"%s(%d sf)",out,acc);
                  break;

        case DEC_PLACES :
                  sprintf(rec_line[MAX_REC_LINES].output,"%s(%d dp)",out,acc);
                  break;

        case NONE:
        default : sprintf(rec_line[MAX_REC_LINES].output,"%s",out);
                  break;
        }
}


void clearrunrecord(void)
{

/* User wants the runcrecord cleared */

        int i;
        for(i=0;i<=MAX_REC_LINES;i++) {
           rec_line[i].inpt[0]=NULL;                                /* empty the strings */
```

```
            rec_line[i].output[0]=NULL;
            rec_line[i].flownum=0;
        }
        usr_update_wind(w_runrecord);
}

static void update_input(void)
{
/* update only the input box of the runentry window */

        force_update(w_runentry,IN_OUT_X,IN_Y,IN_OUT_X+20*CHAR_WIDTH,
                                IN_Y+96);
}


static void update_output(void)
{

/* Update only the output box of runentry window */

        force_update(w_runentry,IN_OUT_X,OUT_Y,IN_OUT_X+20*CHAR_WIDTH,
                                OUT_Y+96);
}


static void update_fkey(void)
{

/* Update only the fuction key box in runentry window*/

        force_update(w_runentry,F_X,F_Y,F_X+3*CHAR_WIDTH,F_Y+64);
}


void mse_on_runentry(univ_block *block)
{

/* The mouse is active on the runentry box..*/

        int fkey_number,err=FALSE;

        switch (mouseb) {
                case(L_PRESSED) :                                       /* SELECT...find the icon */
                if(mouseicon==runbutton) {
                    fkey_number=atoi(runflownum);
                    if(strlen(numinput)==0) strcpy(numinput,"0");
                    run_flowchart(fkey_number,numinput,numoutput,&err);
                    update_fkey();
                    if(!err) {
                       if(autoswap) {
                                        strcpy(numinput,numoutput);
                                        *numoutput=NULL;
                                        if(strpbrk(numinput,".")) dp_gone=TRUE;
                                        else dp_gone=FALSE;
                       }
                       update_record(numinput,fkey_number,
                                        (autoswap?numinput:numoutput));
                    } else update_record(numinput,fkey_number,"Error");
```

```
                    usr_update_wind(w_runrecord);
                    update_input();
                    update_output();
                    last_was_fkey=TRUE;
                } else if(mouseicon==clearinbutton ) {

                                *numinput=NULL;
                                *numoutput=NULL;
                                update_input();
                                update_output();
                                dp_gone=FALSE;

            } else if(mouseicon==swapbutton) {

                                strcpy(numinput,numoutput);
                                *numoutput=NULL;
                                update_input();
                                update_output();
                                if(strpbrk(numinput,".")) dp_gone=TRUE;
                                else dp_gone=FALSE;

            } else if(mouseicon==autoswapbutton) {

                                autoswap=~autoswap;                                    /* toggle it */
                                usr_update_wind(w_runentry);
            }

            break;
            case M_PRESSED :
                                usr_create_menu(&Edit_menu,mousex,mousey,
                                        Edit_menu_decode);
                                break;
            case R_PRESSED :
                                break;
            }

}

void mse_on_runrecord(univ_block *block)
{

/* Mouse is on the runrecord : Only MENU is active on runrecord & pops up edit menu */

        switch(mouseb) {
            case(M_PRESSED) :

                                usr_create_menu(&Edit_menu,mousex,mousey,
                                        Edit_menu_decode);
                                 break;
            default :
                                break;
        }
}


void key_on_runentry(univ_block *block)
{
```

```
/* Key pressed whilst on runentry window..*/

        static int fkey_number=1,err=FALSE;
        char oldin[30];

        if((fkey_number=is_function_key(keycode))!=NO_GOOD) {

/* User has tapped a red function key whilst on runentry, => he wants to RUN a flowchart */

                sprintf(runflownum,"%d",fkey_number);
                update_fkey();
                if(strlen(numinput)==0) strcpy(numinput,"0");
                run_flowchart(fkey_number,numinput,numoutput,&err);
                if (!err) {
                   if(autoswap) {
                        strcpy(oldin,numinput);                          /*might be needed if not fkey*/
                        strcpy(numinput,numoutput);
                        *numoutput=NULL;
                        if(strpbrk(numinput,".")) dp_gone=TRUE;
                        else dp_gone=FALSE;
                   }
                   if(last_was_fkey) {

/* second function key in succession, use previous input */

                        if(autoswap) update_record("----",fkey_number,
                                     (autoswap?numinput:numoutput));
                        else update_record(numinput,fkey_number,numoutput);
                   } else {
                        if(autoswap) update_record(oldin,fkey_number,numinput);
                        else   update_record(numinput,fkey_number,numoutput);
                   }
                } else update_record(numinput,fkey_number,"Error");
                usr_update_wind(w_runrecord);
                last_was_fkey=TRUE;
                update_input();
                update_output();

        } else if(keycode==COPY_KEY) {                                   /* Move output to input */

                   strcpy(numinput,numoutput);
                   *numoutput=NULL;
                   if(strpbrk(numinput,".")) dp_gone=TRUE;
                   else dp_gone=FALSE;
                   update_input();
                   update_output();
                   last_was_fkey=FALSE;

        } else if(keycode==HOME_KEY) {                                   /* clear the in/out  buffers */

                  *numinput=NULL;
                  *numoutput=NULL;
                  dp_gone=FALSE;
                  update_input();
                  update_output();
                  last_was_fkey=FALSE;

        } else  if((keycode>='0'&&keycode<='9')||
```

```
                ((keycode=='.')&&((!dp_gone)||(last_was_fkey)))||
                (keycode=='-')) {
            int p;
            if(last_was_fkey) {
                *numinput=NULL;                                    /* start numinput again */
                dp_gone=FALSE;
            }
            p=strlen(numinput);
            last_was_fkey=FALSE;
            if(keycode=='.') dp_gone=TRUE;
            if((strlen(numinput)<14)&&!((keycode=='-')&&strlen(numinput)!=0)) {
               numinput[p]=(char)keycode;
               numinput[p+1]=NULL;
               update_input();
            } else vdu(7);                                         /* beep at user if too long */

        } else if((keycode==DELETE_KEY)&&(strlen(numinput)>0)) {

            int p=strlen(numinput)-1;
            if(numinput[p]=='.') dp_gone=FALSE;
            numinput[p]=NULL;
            last_was_fkey=FALSE;
            update_input();

        } else if(keycode==RETURN_KEY) {

                run_flowchart((fkey_number=atoi(runflownum)),
                              numinput,numoutput,&err);
                if(!err) {
                  if(autoswap) {
                     strcpy(oldin,numinput);                       /*might be needed if not fkey*/
                     strcpy(numinput,numoutput);
                     *numoutput=NULL;
                   }
                   if(last_was_fkey) {
                      update_record("----",fkey_number,
                                    (autoswap?numinput:numoutput));
                   } else {
                     if(autoswap) update_record(oldin,fkey_number,numinput);
                     else   update_record(numinput,fkey_number,numoutput);
                   }
                } else update_record(numinput,fkey_number,"Error");
                last_was_fkey=TRUE;
                usr_update_wind(w_runrecord);
                update_input();
                update_output();
        }
}


void key_on_runrecord(univ_block *block)
{

/* a key has been pressed whilst on the runrecord window */

    key_on_runentry(block);                                        /* pass it to runentry */
}
```

```
/*************************************************************/
/**** Routines connected with drawing runrecord entries ****/
/*************************************************************/


#define NORTH 0
#define EAST 1
#define WEST 2
#define SOUTH 3

static void arrowhead(int orient)
{

/* draw an arrowhead */

        switch(orient) {
        case NORTH :
                        moveby(-8,-16);
                        moveby(16,0);
                        plot(81,-8,16);
                        break;
        case SOUTH :
                        moveby(-8,16);
                        moveby(16,0);
                        plot(81,-8,-16);
                        break;
        case WEST :
                        moveby(16,8);
                        moveby(0,-16);
                        plot(81,-16,8);
                        break;

        case EAST :     moveby(-16,8);
                        moveby(0,-16);
                        plot(81,16,8);
                        break;
        }
}



static int gpos_y(void)
{

/* Return position of graphics cursor in y direction */

        int in[2],out[2];
        error *my_error;
        reg_set regs;
        regs.r[0]=(int)in;
        regs.r[1]=(int)out;
        in[0]=139;
        in[1]=-1;
        my_error=swix(OS_ReadVduVariables,&regs);
        if(my_error) wimp_error(my_error,WE_OK);
        return(out[0]);
}
```

```
void redraw_runrecord(redraw_block *block)
{

/* redraw the run record window */

        int i=0,numdone;
        gcol(0,yellow);
        move(0,W_RUNRECORD_HEIGHT-20);
        charsize(16,16);
        puts("Last 30 results   ");
        charsize(8,8);
        for(i=0;i<=MAX_REC_LINES;i++) {
              move(0,W_RUNRECORD_HEIGHT-CHAR_HEIGHT*((i<<1)+5));
              if (strlen(rec_line[i].inpt)) {
                gcol(0,white);
                moveby(0,6);                /
                printf("   %s",rec_line[i].inpt);
                moveby(4,-CHAR_HEIGHT/3);
                gcol(0,grey2);
                drawby(40,0);
                arrowhead(EAST);
                draw(300,gpos_y());

                moveby(4,CHAR_HEIGHT/2);
                gcol(0,white);
                numdone=printf("F %d",rec_line[i].flownum);
                gcol(0,grey2);

/* box surround */
                moveby(6,12);

                drawby(0,-(CHAR_HEIGHT+12));
                drawby(-(CHAR_WIDTH*numdone+12),0);
                drawby(0,CHAR_HEIGHT+12);
                drawby(CHAR_WIDTH*numdone+12,0);

                moveby(-2,-12-CHAR_HEIGHT/2);
                drawby(40,0);
                arrowhead(EAST);
                draw(450,gpos_y());
                gcol(0,white);
                moveby(4,CHAR_HEIGHT/3+3);
                if(!strncmp(rec_line[i].output,"Error",5)) gcol(0,red);
                if(strlen(rec_line[i].output)) puts(rec_line[i].output);
                moveby(0,-6);
              }else puts("");

              gcol(0,grey2);
              plot(17,W_RUNRECORD_WIDTH,0);
              puts("");

        }
}


void redraw_runentry(redraw_block *block)
```

```
{

/* redraw the runentry window */

        if(intersect(block,IN_OUT_X-10,IN_Y+IN_OUT_HT,
                           IN_OUT_X+10*CHAR_WIDTH,
                           IN_Y+IN_OUT_HT+CHAR_HEIGHT)) {
                move(IN_OUT_X-10,IN_Y+IN_OUT_HT+2*CHAR_HEIGHT);
                gcol(0,white);
                puts("Input :");
        }

        if(intersect(block,F_X-CHAR_WIDTH,F_Y,
                           F_X+CHAR_WIDTH,F_Y+CHAR_HEIGHT)) {
               move(F_X-(int)1.5*CHAR_WIDTH,F_Y+CHAR_HEIGHT+4);
               gcol(0,red);
               puts("F");
        }

        if(intersect(block,IN_OUT_X-10,OUT_Y+IN_OUT_HT,
                           IN_OUT_X+10*CHAR_WIDTH,
                           OUT_Y+IN_OUT_HT+2*CHAR_HEIGHT)) {

           move(IN_OUT_X,OUT_Y+IN_OUT_HT+CHAR_HEIGHT);
           gcol(0,white);
           puts("Output :");
        }


        gcol(0,white);
        move(F_X+(int)(1.5*CHAR_WIDTH),IN_Y);
        drawby(0,-50);
        arrowhead(SOUTH);
        draw(F_X+(int)(1.5*CHAR_WIDTH),F_Y+48);
        move(F_X+(int)(1.5*CHAR_WIDTH),F_Y);

        if(!autoswap) {
                                        /* not autoswap.Indicate fact on diagram : line links o/p & i/p */
           drawby(0,-50);
           arrowhead(SOUTH);
           draw(F_X+(int)(1.5*CHAR_WIDTH),OUT_Y+IN_OUT_HT);
        } else {
                                                          /* autoswap : indicate fact on diagram */
           drawby(0,20);
           drawby(50,0);
           drawby(0,40);
           arrowhead(NORTH);
           draw(F_X+(int)(1.5*CHAR_WIDTH)+50,IN_Y);
        }
        move(220,W_RUNENTRY_HEIGHT-190+CHAR_HEIGHT);
        gcol(0,white);
        if(autoswap) puts("On ");
        else puts("Off");
}


/*********************************************************/
/*                                                       */
```

```
/* General flowcharts functions                          */
/*                                                       */
/*******************************************************/


int length_of_flowchart(int number)
{

/* returns number of boxes in flowchart */

        flow_box_type *this_one=&flowchart[number];
        int count=1;                                        /* always at least one : header box ! */
        while(this_one->next!=NULL) {
                count++;
                this_one=this_one->next;
        }
        return(count);
}



/*******************************************************/
/*                                                       */
/* SAVE and LOAD flowcharts functions                    */
/*                                                       */
/*******************************************************/

/* FLOWCHART file specification :

        byte                    contains :

        0-3                     "F","L","O","W"
        4-7                     Version number of file fmt (1=0.1, etc )
        8-11                    number of flowcharts defined

     for each flowchart
        4 bytes                 number of boxes, this chart
        12 bytes                title..not used this version

        for each box
        4 bytes             op_num
        4 bytes             box_num
        8 bytes             data (param1 - title is above and only occurs x10)
        next box
     next flowchart



*/

static void fputdouble(FILE *file, double num)
{

/* send a double number to file */

        int i;
        union {
                char bytes[sizeof(double)];
```

```
                double dbl;
        }fiddle;
        fiddle.dbl=num;
        for(i=0;i<sizeof(double);i++) fputc(fiddle.bytes[i],file);
}

static void fputword(FILE *file, int num)
{

/* send a word to file */

        int i;
        union {
                char bytes[sizeof(int)];
                int word;
        } fiddle;
        fiddle.word=num;
        for(i=0;i<sizeof(int);i++) fputc(fiddle.bytes[i],file);
}

static double fgetdouble(FILE *file)
{

/* get double from file */

        int i;
        union {
                char bytes[sizeof(double)];
                double dbl;
        } fiddle;
        for(i=0;i<sizeof(double);i++) fiddle.bytes[i]=fgetc(file);
        return(fiddle.dbl);
}


void fgetname(FILE *file)
{
        int i;
                                                        /* for the moment, merely read 16 bytes */
        for(i=0;i<16;i++) fgetc(file);
}


static int fgetword(FILE *file)
{
        int i;
        union {
                char bytes[sizeof(int)];
                int word;
        } fiddle;
        for(i=0;i<sizeof(int);i++) fiddle.bytes[i]=fgetc(file);
        return(fiddle.word);
}

/* used in load_flowcharts and save_flowcharts */

#define vname (current->data.varname)
```

```
static int exists(char *fname)
{

/* determine whether fname is a file..return TRUE if is, else  return FALSE */

        osfile_block block;
        error *e;
        int rv ;
        block.action=5;
        block.name=fname;
        e=osfile(&block);
        if(e) {
            wimp_error(e,WE_OK);
            rv=0;
        } else rv=(block.action==1);
        return(rv);
}

int wimp_decision(char *text)
{

/* present user with choice, return his decision */

        error e;
        e.errnum=0;
        strcpy(e.errmess,text);
        return(wimp_error(&e,WE_OK+WE_CANCEL+WE_NOERROR));
}

void save_flowcharts(char *fname)
{

/* save all the flowcharts in the file fname */

        int i,j,resp=1;
        flow_box_type *current;
        FILE *outfile;
        char filename[60];

        if(strlen(fname)==0) {
          do_error("No filename given!");
        } else {
          if (is_pathname(fname)) strcpy(filename,fname);
          else sprintf(filename,"%s.%s",FLOWDIRNAME,fname);
          if(exists(filename)) resp=wimp_decision("This file already exists - Shall I replace it?");
          if(resp==1) {                                                                    /* i.e. YES */
             outfile=fopen(filename,"wb");
             if(outfile==NULL) {
                       do_error("Cannot open file to save it : disk full, or exists and is locked??");
                       return;
             }
             fputc('F',outfile);
             fputc('L',outfile);
             fputc('O',outfile);
             fputc('W',outfile);
             fputword(outfile,FILE_VERS_NUM);
```

```
            fputword(outfile,MAX_FLOWS);


            for(i=1;i<=MAX_FLOWS;i++) {
               fputword(outfile,length_of_flowchart(i));

                                                                  /* name to be implemented */

               current=&flowchart[i];
               do {
                       fputword(outfile,current->op_num);
                       fputword(outfile,current->box_num);
                       fputword(outfile,current->dtype);
                       switch(current->dtype) {
                       case NO_PARM :
                                    break;
                       case NUMBER :
                                    fputdouble(outfile,current->data.param1);
                                    break;

                       case VARIABLE :
                                    fputword(outfile,(int)strlen(vname));
                                    for(j=0;j<strlen(vname);j++)
                                         fputc(vname[j],outfile);
                                    break;
                       }

                       current=current->next;
               } while(current!=NULL);
            }
            fclose(outfile);
            settype(filename,0xF10);
         }
      }
}


void load_flowcharts(char *fname)
{

/* retrieve all the flowcharts from file fname */

        int i,j,k,version,num_charts,num_boxes,len;
        flow_box_type *current,*previous;
        FILE *infile=fopen(fname,"rb");
        max_box_number=0;

        if(infile==NULL) {
                do_error("Cannot open file : insufficient access??");
                return;
        }

        if(fgetc(infile)!='F'||
           fgetc(infile)!='L'||
           fgetc(infile)!='O'||
           fgetc(infile)!='W') {
                fclose(infile);
                do_error("Not a FLOWCHART file");
```

```
                return;
        }

        version=fgetword(infile);                                   /* version of file format */
        num_charts=fgetword(infile);

        for(i=1;i<=MAX_FLOWS;i++) {
                previous=NULL;
                current=&flowchart[i];
                current->next=NULL;
                num_boxes=fgetword(infile);
                if(max_box_number<num_boxes) max_box_number=num_boxes;

                                                            /* title not implemented this version*/

                for(j=0;j<num_boxes;j++) {
                        current->op_num=fgetword(infile);
                        current->box_num=fgetword(infile);
                        current->dtype=fgetword(infile);

                        switch(current->dtype) {
                        case NO_PARM :
                                        break;
                        case NUMBER :
                                        current->data.param1=fgetdouble(infile);
                                        break;

                        case VARIABLE :
                                        len=fgetword(infile);
                                        vname=malloc(len+1);
                                        for(k=0;k<len;k++)
                                              vname[k]=fgetc(infile);
                                        vname[len]=NULL;
                                        break;
                        }

                        current->next=(flow_box_type*)malloc(sizeof(flow_box_type));
                        current->prev=previous;
                        previous=current;                                   /* reestablish links */
                        current=current->next;
                }
                free(current);                                      /* undo the last one,(done in error)*/
                if (previous!=NULL) previous->next=NULL;
        }
        fclose(infile);
        FLOW_HEIGHT=W_FLOWBOX_HEIGHT+max_box_number*TOT_BOX_HT;
        set_window_extent(w_flowcharts,0,0,W_FLOWBOX_WIDTH,FLOW_HEIGHT);
        if(is_window_open(w_flowcharts)) {
                usr_close_wind(w_flowcharts);
                usr_open_wind(w_flowcharts,ON_TOP);
        }
}

static double frac(double num)
{

/* return the fractional part of the number num */
```

```
        double ret=(num>0)?num-floor(num):ceil(num)-num;
        if(fabs(ret-1)<EPSILON) ret=0;                          /* floating point errors */
        return(ret);
}

static int dbp(double num)
{

/* return the number of digits before the decimal point */

        return((int)floor(log10(fabs(num))+1));
}


void read_dbl(char *rawbuffer, double *pnum)
{
        char finbuffer[30];
        int i=0,j=0,dp_gone=FALSE;
        while(i<strlen(rawbuffer)) {
                if(rawbuffer[i]=='.') {
                        if(dp_gone) do_error("Two decimal points!");
                        else dp_gone=TRUE;
                }
                if(rawbuffer[i]!=',') finbuffer[j++]=rawbuffer[i++];
                else i++;
        }
        finbuffer[j]=NULL;
        sscanf(finbuffer,"%lf",pnum);
}


static char *print_dp(double number, int precision)
{

/* Print number to given precision*/

static  char fmt[30],output[30];
        sprintf(fmt,"%%1.%uf",precision);                       /* construct format */
        sprintf(output,fmt,number);                     /* print number into output block*/
        return(output);
}

static char *print_sf(double number, int accuracy)
{

/* return formatted string which represents number to given accuracy, in sig figs... */

        char b[30],acc_string[10];
        static char c[30];
        int i,limit,count=0,dp_gone=FALSE;
        double mant,exp,sign;

        sprintf(acc_string,"%%.%dlf",accuracy-1);

        if(number<0) {
            number=fabs(number);
            sign=-1;
        } else sign=1;
```

```
        if(number!=0) {
            exp=floor(log10(number));                                   /* exponent */
            mant=number/pow(10,exp);                                    /* mantissa */
            sprintf(b,acc_string,mant);

                                        /* b contains string version of mantissa to required accuracy */

          sprintf(c,"%lf",sign*atof(b)*pow(10,exp));

                                                                  /* c contains number to accuracy */

          limit=strcspn(c,"123456789");                          /* limit indexes first non-zero digit */

          for(i=0;i<limit;i++) if(c[i]=='.') {
              dp_gone=TRUE;                                  /* check if dp in first section of number */
              break;
          }

          while((count<accuracy)&&(c[limit]!=NULL)) {
             if(c[limit]!='.') {                              /* Ensure enough digits for accuracy */
                count++;
                limit++;
             } else {
                dp_gone=TRUE;
                limit++;
             }
          }

          if (dp_gone) {
              c[limit]=NULL;
          }
          else c[strcspn(c,".")]=NULL;                            /* must go up to decimal point */
        } else {
                                                                        /* number is 0 */
          strncpy(c,"0.000000000000",accuracy+1);
        }
        return(c);
}
```

9.4.7 C.f_PLOT

```
/* >c.f_plot


@@@@@@@          @@@@@@
@                @     @ @         @@@@     @@@@@
@                @     @ @       @    @     @
@@@@@            @@@@@@  @       @    @     @
@                @       @       @    @     @
@                @       @       @    @     @
@                @       @       @    @     @
@       @@@@@@@  @       @@@@@@   @@@@      @


provides graph plotting module with autoscaling

*/


#include "h.flow"



#define xco(va)  ((int)((va-x_min)*x_scale))
#define yco(va)  ((int)((va-y_min)*y_scale))

#define xval(x)  ((double)(x*x_scale)+x_min)
#define yval(y)  ((double)(y*y_scale)+y_min)

#define ticklength(num)  ((num%5==0)?15:8)
#define grid 1

static int
          xe,                                          /* x tick every...pixels */
          ye,                                          /* y tick every...pixels */
          avail_horiz,                        /* number of points available in each dir'n */
          avail_vert,
          x_origin_pixels,                               /* position of origin */
          y_origin_pixels;

static double
          x_min,                                          /* x minimum value */
          y_min,
          x_max,                                              /* max values */
          y_max,
          x_scale,                                 /* pixels per unit , horizontally */
          y_scale,                                       /* ditto vertically */
          x_range,                                     /* range of x values */
          y_range,                                            /* y range */
          x_origin,
          y_origin;                                /* origin position, not in pixels */

/* prototypes ... */

void graph_plot(int numpoints,double *x,double *y,int horiz,int vert,int colour);

static void doplot(int numpoints,double *x,double *y,int col);
static void calcparams(int num,double *x,double *y);
```

```
static void xaxis(void);
static void yaxis(void);


void graph_plot(int numpoints, double *x, double *y,
                int                                                    /*number of points in set */
    horiz, int vert,
                int col)
{

/* plot the points in the available size */

        avail_horiz=horiz;
        avail_vert=vert;
        calcparams(numpoints,x,y);
        xaxis();
        yaxis();
        doplot(numpoints,x,y,col);
}

void super_plot(int num, double *x, double *y, int col)
{

/* superimpose the points on the existing plot */

        doplot(num,x,y,col);
}


static void do_label(void)
{

/* put a coordinate point label on the graph */

        move(xco(x_origin),yco(y_origin)+32);
        gcol(0,white);
        printf("(%0.0f,%0.0f)",x_origin,y_origin);
}

static void doplot(int numpoints, double *x, double *y, int col)
{

/* this routine actually does the plotting */

 int i,xx,yy;
 gcol(0,col);
 move(xco(x[0]),yco(y[0]));
 for(i=1;i<numpoints;i++) {
       xx=xco(x[i]);
       yy=yco(y[i]);
       if(xx>-2000&&xx<2000&&yy>-2000&&yy<2000) draw(xx,yy);      /* rudimentary truncate. Not used often! */
 }
 do_label();
}


static void calcparams(int num, double *x, double *y)
{
```

```
/* initialisation..calculate the plotting parameters for the given curve */

  int i;
  x_max=x[0];                                                    /* maximum x value */
  x_min=x[0];                                                    /* minimum x value */
  y_max=y[0];                                                    /* maximum y value */
  y_min=y[0];                                                       /* min y value */

  for(i=1;i<num;i++) {
        if(x[i]>x_max) x_max=x[i];                          /* find these four values */
        if(x[i]<x_min) x_min=x[i];
        if(y[i]>y_max) y_max=y[i];
        if(y[i]<y_min) y_min=y[i];
  }

  y_min=floor(y_min);                                      /* ..& adjust to "nice" values */
  y_max=ceil(y_max);
  x_min=floor(x_min);
  x_max=ceil(x_max);

  x_range=x_max-x_min;
  y_range=y_max-y_min;

  if(fabs(x_range)<1) x_range=1;                    /* avoid nasty effects with small scales */
  if(fabs(y_range)<1) y_range=1;

  x_scale=(avail_horiz/x_range);
  y_scale=(avail_vert/y_range);

  x_origin=min(max(0,x_min),x_max);          /* determine origin position, within these scales */
  y_origin=min(max(0,y_min),y_max);

  x_origin_pixels=xco(x_origin);
  y_origin_pixels=yco(y_origin);
}


static void xaxis(void)
{

/* draw the x axis */

  int i,dr,pp;
  gcol(0,grey1);
  xe=max((int)pow(10,(int)(log10(x_range)-1)),1);                  /* x tick every.... */
  if(y_origin_pixels<0)  y_origin_pixels=0;
  move(0,y_origin_pixels);
  draw(avail_horiz,y_origin_pixels);
  dr=(y_origin_pixels<10)?1:-1;

  for(i=1;i<(int)((x_origin-x_min)/xe);i++) {
     pp=(int)floor(x_origin_pixels-i*xe*x_scale);
     move(pp,y_origin_pixels);
     draw(pp,y_origin_pixels+dr*ticklength(i));
     if(grid && (i%5 == 0)) {                 /* if we need a longer tick (every 5 ticks) then do it */
          move(pp,0);
          plot(21,pp,avail_vert);
```

```
      }
  }

  for(i=1;i<=(int)((x_max-x_origin)/xe);i++) {
       pp=(int)floor(x_origin_pixels+i*xe*x_scale);
       move(pp,y_origin_pixels);
       draw(pp,y_origin_pixels+dr*ticklength(i));
       if(grid && (i % 5 == 0))  {
          move(pp,0);
          plot(21,pp,avail_vert);
       }

   }
   move(avail_horiz-32,y_origin_pixels+dr*32);
   putchar('X');
}


static void yaxis(void)
{

/* draw the y axis : more or less exactly as x axis */

 long i;
 int pp,dr;
 gcol(0,grey1);
 ye=max((int)pow(10,(int)(log10(y_range)-1)),1);
 if(x_origin_pixels<0) x_origin_pixels=0;
 move(x_origin_pixels,0);
 draw(x_origin_pixels,avail_vert);
 dr=(x_origin_pixels<10) ? 1:-1;
 for(i=1;i<=(long)((y_origin-y_min)/(float)ye);i++) {
   pp=(int)floor(y_origin_pixels-i*ye*y_scale);
   move(x_origin_pixels,pp);
   draw(x_origin_pixels+dr*(int)(1.5*ticklength(i)),pp);
   if(grid && (i % 5 == 0))  {
       move(0,pp);
       plot(21,avail_horiz,pp);
   }
 }

for(i=1;i<=(long)((y_max-y_origin)/(float)ye);i++) {
     pp=(int)floor(y_origin_pixels+i*ye*y_scale);
     move(x_origin_pixels,pp);
     draw(x_origin_pixels+dr*ticklength(i),pp);
     if(grid && (i % 5 == 0))  {
         move(0,pp);
         plot(21,avail_horiz,pp);
     }
 }
 move(x_origin_pixels+dr*32,avail_vert);
 putchar('Y');
}

/* for redraw_graphinfo... */

#define next_line(x) x=x-36;\
                     move(5,x)
```

```
static void in_betweeny(int *x)
{
                                                    /* small routine, used by redraw_graphinfo below */

        next_line(*x);
        move(0,*x);
        gcol(0,grey3);
        drawby(700,0);
        *x=*x-10;
        move(5,*x);
        gcol(0,white);
}

static void punits(int n)
{
/* used only by redraw_graphinfo */
        gcol(0,white);
        printf("%d",n);
        gcol(0,grey2);
        if(n!=1.0) printf(" units");
        else printf(" unit");
}


void redraw_graphinfo(redraw_block *block)
{

/* redraw the information box */

        int x=380,i;
        move(5,x);
        gcol(0,grey2);
        printf("Origin  at ");
        gcol(0,white);
        printf("(%.3f,%.3f)",x_origin,y_origin);
        in_betweeny(&x);

        gcol(0,grey2);
        printf("X tick every ");
        punits(xe);
        next_line(x);
        printf("Y tick every ");
        punits(ye);
        in_betweeny(&x);

        gcol(0,grey2);
        printf("vertical   line every ");
        punits(xe*5);
        next_line(x);
        printf("horizontal line every ");
        punits(ye*5);

        in_betweeny(&x);
        gcol(0,grey2);
        printf(" X axis from ");
```

```
        gcol(0,white);
        printf("%.3f",x_min);
        gcol(0,grey2);
        printf(" to ");
        gcol(0,white);
        printf("%.3f",x_max);
        next_line(x);
        gcol(0,grey2);
        printf(" Y axis from ");
        gcol(0,white);
        printf("%.3f",y_min);
        gcol(0,grey2);
        printf(" to ");
        gcol(0,white);
        printf("%.3f",y_max);
        in_betweeny(&x);
        gcol(0,grey2);
        charsize(16,8);
        printf("Charts plotted...\n");
        charsize(8,8);
        x=x-40;
        move(5,x);
        for(i=0;i<nextfree;i++) {
                gcol(0,curve_colour[i]);
                printf("Chart %d  ",index[i]);
        }
}
```

9.4.8 C.F_HELP

```
/*> c.f_help

@@@@@@@         @     @ @@@@@@@ @       @@@@@@
@               @     @ @       @       @     @
@               @     @ @       @       @     @
@@@@@           @@@@@@@ @@@@@   @       @@@@@@
@               @     @ @       @       @
@               @     @ @       @       @
@       @@@@@@@ @     @ @@@@@@@ @@@@@@@ @

 */

#include "flow.h"
#include "h.windasm"

/* This file contains all the coding for the extensive HELP features
   of the program. This is intended to be general purpose.

        menu_block *setup_dir_menu(headertext,dirname)

        will make entries in the specified menu corresponding to
        files found in a directory called HELP, off the current
        directory. When the user selects one of these from the
        help menu, help_menu will be called and will display the
        text on the screen in a dialogue box..
        BUGS :



prototypes :
*/

static menu_block *setup_dir_menu1(char *header,char *directory,int special),
                  *setup_dir_menu2(char *header,char *directory);

static char pathname[250];                                  /* records pathname for help search */
static char filename[250];
static char* helptext=NULL;
static int filesize,help_height;

int    oldfilename_pos;

static void add_to_pathname(char *directory)
{

/* Add this element of pathname to the already accumulated pathname to the selected help file */

        if(*pathname!=NULL) strcat(pathname,".");
        strcat(pathname,directory);
}


static void remove_last_from_path(void)
{
```

```
/* remove last element from the path to helpfile */

        int i=strlen(pathname);
        while(pathname[i]!='.'&&i>0) i--;
        pathname[i]=NULL;
}


menu_block *setup_dir_menu(char *header, char *directory)
{

/* build a menu structure reflecting a directory structure */

        menu_block *rval;
        *pathname=NULL;
        rval=setup_dir_menu1(header,directory,FALSE);
        return(rval);
}


menu_block *setup_dir_menu_r(char *header, char *directory)
{
        menu_block *rval;
        *pathname=NULL;
        rval=setup_dir_menu2(header,directory);
        return(rval);
}




menu_block *setup_rename_menu(char *header, char *directory)
{
        *pathname=NULL;
        Rename_menu=setup_dir_menu1(header,directory,TRUE);
        return(Rename_menu);
}



static error *read_directory(char *pathname, char *replyblock,int *pnumread)
{

/* Very machine-specific routine to read the contents of given directory into memory */

        reg_set regs;
        error *result;
        regs.r[1]=(int)pathname;
        regs.r[2]=(int)replyblock;                                  /* where to put reply */
        regs.r[3]=MAXITEMS;                                /* max number of menu entries possible */
        regs.r[4]=0;                                              /* offset to first item */
        regs.r[5]=2000;                                               /* buffer length */
        regs.r[6]=(int)HELPNAMESTYLE;                          /* Pointer to name to match */
        regs.r[0]=9;
        result=swix(OS_GBPB,&regs);
        if(result) wimp_error(result,WE_OK);
        *pnumread=regs.r[3];                                                /* return value */
```

```
        return(result);
}


static menu_block *setup_dir_menu1(char *header, char *directory,int special)
/* special T= submenu in place for all entries */
{

        char *thisentry,
             *buffer=malloc(300),
             *replyblock=malloc(2000);                                  /* room to put directory names */
        menu_block *block=(menu_block*)malloc(sizeof(menu_block));
        int i,j,numread;

        add_to_pathname(directory);
        strncpy(block->title,header,11);
        block->tit_fcol=MENU_TIT_FGCOL;
        block->tit_bcol=MENU_TIT_BGCOL;
        block->work_fcol=MENU_WA_FGCOL;
        block->work_bcol=MENU_WA_BGCOL;
        block->menu_width=200;
        block->menu_height=40;
        block->vert_gap=MENU_GAP;
        block->m[0].menu_flags=LAST_ITEM;
        block->m[0].sub_menu=(special)? &newname_menu:NO_SUB_MENU;
        block->m[0].icon_flags=menutexticon(MENUTI_FGCOL,MENUTI_BGCOL);
        strcpy(block->m[0].data.text,"None..");

        thisentry=replyblock;
        my_error=read_directory(pathname,replyblock,&numread);

        if (my_error!=NULL) {
          if(my_error->errnum!=NOT_FOUND)
            wimp_error(my_error,WE_OK);
          else
            return(block);

        } else {

          i=0;
          j=0;
          while(j<numread) {
                block->m[i].menu_flags=0;
                sprintf(buffer,"%s.%s",pathname,thisentry);

                if(!is_a_dir(buffer)) {
                  block->m[i].sub_menu=(special)?&newname_menu:NO_SUB_MENU;
                  block->m[i].icon_flags=
                                menutexticon(MENUTI_FGCOL,MENUTI_BGCOL);
                  strncpy(&block->m[i].data.text[0],thisentry,11);
                  thisentry +=strlen(thisentry)+1;
                  i++;
                }
                j++;
/* j= total count, files+directories in flow area, i=files count */
          }
          if(paths_allowed) {
             block->m[i].data.writeable.text_buff=oldfilename;
```

```
            block->m[i].data.writeable.val_string=(char*)-1;
            block->m[i].data.writeable.bufflen=60;
            block->m[i].menu_flags=LAST_ITEM+MENU_WRIT;
            block->m[i].icon_flags=menuwriticon(MENUWI_FGCOL,MENUWI_BGCOL);
            block->m[i].sub_menu=(special)? &newname_menu:NO_SUB_MENU;
            oldfilename_pos=i;
          } else {
            oldfilename_pos=-1;
            if (i>0) block->m[i-1].menu_flags|=LAST_ITEM;
            else block->m[0].menu_flags|=LAST_ITEM;
          }
        }

        free(replyblock);
        free(buffer);
        remove_last_from_path();
        return(block);                                          /* address of block created */
}

static menu_block *setup_dir_menu2(char *header, char *directory)
{

/* this routine is recursive. Sets up a menu structure
   exactly reflecting the structure of the files in the help area
   on the current filing system. User can then select the area
   he requires in a tree-structured manner , and the teacher
   can arrange the help files exactly how he requires them to be
*/
        char *thisentry,
             *buffer=malloc(300),
             *replyblock=malloc(2000);                          /* room to put directory names */
        menu_block *block=(menu_block*)malloc(sizeof(menu_block));
        int i,numread;

        add_to_pathname(directory);
        strncpy(block->title,header,11);
        block->tit_fcol=MENU_TIT_FGCOL;
        block->tit_bcol=MENU_TIT_BGCOL;
        block->work_fcol=MENU_WA_FGCOL;
        block->work_bcol=MENU_WA_BGCOL;
        block->menu_width=200;
        block->menu_height=40;
        block->vert_gap=0;
        block->m[0].menu_flags=LAST_ITEM;
        block->m[0].sub_menu=NO_SUB_MENU;
        block->m[0].icon_flags=menutexticon(MENUTI_FGCOL,MENUTI_BGCOL);
        strcpy(block->m[0].data.text,"None..");

        thisentry=replyblock;
        my_error=read_directory(pathname,replyblock,&numread);

        if (my_error!=NULL) {
          if(my_error->errnum!=NOT_FOUND)
            wimp_error(my_error,WE_OK);
          else
            return(block);

        } else {
```

```
          i=0;
          while(i<numread) {
                block->m[i].menu_flags=0;
                sprintf(buffer,"%s.%s",pathname,thisentry);
                if(is_a_dir(buffer)) {
                     block->m[i].sub_menu=
                          setup_dir_menu2(thisentry,thisentry);
                } else     block->m[i].sub_menu=NO_SUB_MENU;
                block->m[i].icon_flags=
                                menutexticon(MENUTI_FGCOL,MENUTI_BGCOL);
                strncpy(&block->m[i].data.text[0],thisentry,11);
                thisentry +=strlen(thisentry)+1;
                i++;
          }

          if (i>0) block->m[i-1].menu_flags|=LAST_ITEM;
          else block->m[0].menu_flags|=LAST_ITEM;
        }

        free(replyblock);
        free(buffer);
        remove_last_from_path();
        return(block);                                          /* address of block created */
}

void help_menu(univ_block * block)
{

/* decodes the help menu : filename is assigned the pathname
   of the file selected. Note that this MAY be a directory
   if the user selected an item which has a right arrow after
   it...
*/
        int i=0,
            numlines=0;
        FILE *handle;
        usr_close_wind(w_help);
        sprintf(filename,"%s.",HELPDIRNAME);
        decode_menu(Help_menu,block,(textbuf*)(filename+strlen(HELPDIRNAME)+1),my_error);
        while(filename[i]!=0x0D) i++;                           /* replace term CR with NULL */
        filename[i]=NULL;
        if(strcmp(filename+i-4, "None")) {
           if(!is_a_dir(filename)) {
              if(helptext!=NULL) free(helptext);
              helptext=malloc(filesize=fextent(filename));
              handle=fopen(filename,"r");
              if(handle==NULL) {
                        do_error("Can't open help file");
                        return;
              }
              fread(helptext,1,filesize,handle);
              fclose(handle);
              for(i=0;i<filesize;i++)
                   if(helptext[i]=='\x0A') {
                        numlines++;
                        helptext[i]=NULL;
              }
```

```
            help_height=numlines*CHAR_HEIGHT+500;
            set_window_extent(w_help,0,0,HELP_WIDTH,help_height);
            usr_open_wind_scroll(w_help,ON_TOP,0,help_height);
         } else do_error("Move pointer over the \x89 symbol from left to right");/* user selected a directory ! *
      }
}


void mouse_on_help(univ_block *block)
{

/* A mouse button was pressed whilst the mouse was on the help icon. Create the help menu */

      switch(mouseb) {

      case M_PRESSED :
                  usr_create_menu(Help_menu,mousex,mousey,
                              help_menu);
                  break;
      default :
                  break;
      }
}



void redraw_help(redraw_block *block)
{

/* prints the contents of helptext (which is assumed to be a text file) on the screen */

      int i=0,numlines=0;
      move(HELP_WIDTH-200,help_height-200);
      put_sprite("help",0,sp_area);
      move(HELP_WIDTH/2-200,help_height-200);
      put_sprite("helptxt",0,sp_area);

      gcol(0,white);
      while (i<filesize) {
         if(intersect(block,0,help_height-300-(CHAR_HEIGHT)*(numlines+3),
                  HELP_WIDTH,help_height-300-(CHAR_HEIGHT)*(numlines-3))) {
           move(5,help_height-300-CHAR_HEIGHT*numlines);
           puts(helptext+i);
         }
         i+=strlen(helptext+i)+1;
         numlines++;
      }
}
```

9.4.9 C.F_SETUP

```
/*> c.f_setup

@@@@@@@          @@@@@  @@@@@@@ @@@@@@@ @     @ @@@@@@
@               @     @ @          @    @     @ @     @
@               @       @          @    @     @ @     @
@@@@@            @@@@@  @@@@@      @    @     @ @@@@@@
@                    @ @          @    @     @ @
@               @    @ @          @    @     @ @
@       @@@@@@@  @@@@@  @@@@@@@    @     @@@@@  @

 */


#include "flow.h"

/*
   The windows are referenced outside this module by the following
   identifiers .......:
*/

       int w_flowcharts,                                    /* flowchart representations */
           w_runrecord,                                          /* record of running */
           w_runentry,                                  /* to enter input and flow choices */
           w_help,                                                        /* help */
           w_boxedit,
           w_plotwindow,
           w_graphinfo,
           w_vartable,
           w_panel,

           calcscreen,                                   /* boxeditor screen handle (icon) */
           clearbutton,                                     /* clear the box editor input */
           enterbutton,
           calcbutton[40],                             /* icon handles for boxeditor buttons */
/* in boxeditor window */


           runbutton,                                           /* in runentry window, tools */
           clearinbutton,
           swapbutton,
           autoswapbutton,

           calcicon,                                                  /* in control panel */
           runicon,
           flowicon,
           screensaveicon,
           helpicon,
           exiticon,
           err_ok_icon;


char *screen,                                               /* boxeditor screen contents */
     *numinput,                                                      /* 20 characters */
     *numoutput,                                                /* also 20 characters */
     *runflownum,                                               /* runentry box icons */
     *Accnumber,                                          /* Accuracy input from user */
```

```
    *del_var_name,                                  /* variable name to delete */
    *crtvar_name,                                     /*created variable name */
    *crtvar_val,                                     /*created variable value */
    *Xmaxnum,*Xminnum,*Fgraphnum,*Super_num,
    *loadfilename,*savefilename;


menu_block *Help_menu,*Load_menu,*Rename_menu,*Delete_menu;


static wind_block  flowcharts = {
                            100,                                                150
                            500,
                            900,
                            200,                                                W_F
                            -1,                             /* open window behind nothing */
                        TITLE_BAR+MOVEABLE+
                        V_SCROLL_BAR+
                        H_SCROLL_BAR,
                        stdcols()
/* extent coords */
                        0,
                        0,
                        W_FLOWBOX_WIDTH,
                        W_FLOWBOX_HEIGHT,
                        0x2D,
                        NOTIFY_APPLIC+tit_bar_cols,                     /* button type */
                        0,
                        0,
                        "Flowcharts",
                        0,
                        };


static wind_block  plotwindow = {
                            0,                                                  100
                            W_PLOTBOX_WIDTH,
                            100+W_PLOTBOX_HEIGHT,
                             0,                                                 W_
                            -1,                             /* open window behind nothing */
                        TITLE_BAR+MOVEABLE+
                        V_SCROLL_BAR+
                        H_SCROLL_BAR,
                        stdcols()
/* extent coords */
                        0,
                        0,
                        W_PLOTBOX_WIDTH,
                        W_PLOTBOX_HEIGHT,
                        0x2D,
                        NOTIFY_APPLIC+tit_bar_cols,                     /* button type */
                        0,
                        0,
                        "Graph",
                        0,
                        };
```

```
static wind_block  control_panel = {
                              0,                                        /* work area coords */
                              0,
                              1280,
                              96,
                              0,                                    /* scroll bars positions */
                              0,
                              -1,                                 /* open window behind nothing */
                        TRESPASS+REDRAW_OK,
                        stdcols()
/* extent coords */
                        0,
                        0,
                        1280,
                        96,
                        0x2D,
                        0,                                              /* button type */
                        0,
                        0,
                        "",
                        0,
                        };


static wind_block  runrecord = {
                              1+W_RUNENTRY_WIDTH,                       /* work area coords */
                              150,
                              800+1+W_RUNENTRY_WIDTH,
                              150+W_RUNENTRY_HEIGHT,
                              0,                                    /* scroll bars positions */
                              0,
                              -1,                                 /* open window behind nothing */
                        TITLE_BAR+MOVEABLE+
                        V_SCROLL_BAR+
                        H_SCROLL_BAR,
                        stdcols()
/* extent coords */
                        0,
                        0,
                        W_RUNRECORD_WIDTH,
                        W_RUNRECORD_HEIGHT,
                        0x2D,
                        NOTIFY_APPLIC+tit_bar_cols,
                        0,
                        0,
                        "Run Record",
                        0,
                        };

static wind_block  varbox = {
                              300,                                      /* work area coords */
                              150,
                              300+W_VARBOX_WIDTH,
                              150+W_VARBOX_HEIGHT,
                              0,                                    /* scroll bars positions */
                              0,
                              -1,                                 /* open window behind nothing */
                        TITLE_BAR+MOVEABLE+
```

```
                        V_SCROLL_BAR,
                        stdcols()
/* extent coords */
                        0,
                        0,
                        W_VARBOX_WIDTH,
                        W_VARBOX_HEIGHT,
                        0x2D,
                        NOTIFY_APPLIC+tit_bar_cols,
                        0,
                        0,
                        "Variables",
                        0,
                        };

static wind_block  runentry = {
                              0,                                    /* work area coords */
                              150,
                              0+W_RUNENTRY_WIDTH,
                              150+W_RUNENTRY_HEIGHT,
                              0,                                 /* scroll bars positions */
                              0,
                              -1,                              /* open window behind nothing */
                        TITLE_BAR+MOVEABLE,
                        stdcols()
/* extent coords */
                        0,
                        0,
                        W_RUNENTRY_WIDTH,
                        W_RUNENTRY_HEIGHT,
                        0x2D,
                        NOTIFY_APPLIC+tit_bar_cols,                        /* button type */
                        0,
                        0,
                        "Run Info",
                        0,
                        };



static wind_block  boxeditor = {
                              700,                                  /* work area coords */
                              200,
                              700+W_BOXEDIT_WIDTH,
                              200+W_BOXEDIT_HEIGHT,
                              0,                                 /* scroll bars positions */
                              0,
                              -1,                              /* open window behind nothing */
                        TITLE_BAR+MOVEABLE,
                        stdcols()
/* extent coords */
                        0,
                        0,
                        W_BOXEDIT_WIDTH,
                        W_BOXEDIT_HEIGHT,
                        0x2D,
                        SELECT_NOTIFY+tit_bar_cols,
                        0,
```

```
                        0,
                        "Box Editor",
                        0,
                        };


static wind_block  graphinfo = {
                              0,                                    /* work area coords */
                              0,
                              700,
                              400,
                              0,                                  /* scroll bars positions */
                              0,
                              -1,                                /* open window behind nothing */
                        TITLE_BAR+MOVEABLE,                             /* no title bar */
                        stdcols()
/* extent coords */
                        0,
                        0,
                        700,
                        400,
                        0x2D,
                        tit_bar_cols,                                   /* button type */
                        0,
                        0,
                        "Graph info",
                        0,
                        };


static wind_block  help_wind_block = {
                              0,                                /* work area coords on screen */
                              200,
                              HELP_WIDTH,
                              HELP_HEIGHT,
                              0,                                  /* scroll bars positions */
                              0,
                              -1,                                /* open window behind nothing */
                        TITLE_BAR+MOVEABLE+
                        V_SCROLL_BAR+H_SCROLL_BAR,
                        stdcols()
/* extent coords */
                        0,
                        0,
                        HELP_WIDTH,
                        HELP_HEIGHT,
                        0x2D,
                        CLICK_SELECT+tit_bar_cols,                      /* button type */
                        0,
                        0,
                        "HELP",
                        0,
                        };


static menu_block Varval_menu = {
                          "Value",
```

```
                        stdmencols()
                        240,                                        /*menu width*/
                        80,                                        /*menu height*/
                        MENU_GAP,                                /*gap betweenentries*/
/* menu items ... */

                        LAST_ITEM+MENU_WRIT,
                        NO_SUB_MENU,                                /*sub menu*/
                        menuwriticon(MENUWI_FGCOL,MENUWI_BGCOL),
                        "dummy",
                    };

static menu_block Crtvar_menu = {
                        "Name ?",
                        stdmencols()
                        240,                                        /*menu width*/
                        80,                                        /*menu height*/
                        MENU_GAP,                                /*gap betweenentries*/
/* menu items ... */

                        LAST_ITEM+MENU_WRIT,
                        &Varval_menu,                                /*sub menu*/
                        menuwriticon(MENUWI_FGCOL,MENUWI_BGCOL),
                        "name",
                    };


menu_block newname_menu = {
                        "Change to..",
                        stdmencols()
                        240,                                        /*menu width*/
                        80,                                        /*menu height*/
                        MENU_GAP,                                /*gap betweenentries*/
/* menu items ... */

                        LAST_ITEM+MENU_WRIT,
                        NO_SUB_MENU,                                /*sub menu*/
                        menuwriticon(MENUWI_FGCOL,MENUWI_BGCOL),
                        "dummy",
                    };


static menu_block Name_menu = {
                        "Name ?",
                        stdmencols()
                        240,                                        /*menu width*/
                        80,                                        /*menu height*/
                        MENU_GAP,                                /*gap betweenentries*/
/* menu items ... */

                        LAST_ITEM+MENU_WRIT,
                        NO_SUB_MENU,                                /*sub menu*/
                        menuwriticon(MENUWI_FGCOL,MENUWI_BGCOL),
                        "name",
                    };
```

```
menu_block Vars_menu = {
                        "Variables",
                        stdmencols()
                        240,                                        /*menu width*/
                        40,                                        /*menu height*/
                        MENU_GAP,                                /*gap betweenentries*/
/* menu items ... */

                        0,
                        NO_SUB_MENU,                                /*sub menu*/
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "Clear all",

                        0,
                        &Name_menu,                                /*sub menu*/
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "Delete var",


                        0,
                        &Crtvar_menu,
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "Create var",


                        LAST_ITEM,
                        &Crtvar_menu,                                /*sub menu*/
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "Adjust var",
                    };


static menu_block save_menu = {
                        "Name ?",
                        stdmencols()
                        240,                                        /*menu width*/
                        80,                                        /*menu height*/
                        MENU_GAP,                                /*gap betweenentries*/

                        LAST_ITEM+MENU_WRIT,
                        NO_SUB_MENU,
                        menuwriticon(MENUWI_FGCOL,MENUWI_BGCOL),
                        "dummy",
                    };


menu_block File_menu = {
                        "Filing",
                        stdmencols()
                        240,                                        /*menu width*/
                        40,                                        /*menu height*/
                        MENU_GAP,                                /*gap betweenentries*/
/* menu items ... */

                        0,
                        NO_SUB_MENU,                                /*sub menu*/
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
```

```
                        "Rename ",

                        0,
                        NO_SUB_MENU,                                    /*sub menu*/
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "Delete",

                        DOT_LINE,                                      /*menu flags */
                        NO_SUB_MENU,                                    /*sub menu*/
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "Load",

                        LAST_ITEM,
                        &save_menu,                                     /*sub menu*/
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "Save",

                    };


menu_block Prefs_menu = {
                        "Preferences",
                        stdmencols()
                        200,                                            /*menu width*/
                        40,                                            /*menu height*/
                        MENU_GAP,                                    /*gap betweenentries*/
/* menu items ... */
                        0,                                             /*menu flags */
                        NO_SUB_MENU,                                    /*sub menu*/
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "Degrees",

                        DOT_LINE,                                      /*menu flags */
                        NO_SUB_MENU,                                    /*sub menu*/
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "Radians",

                        0,                                             /*menu flags */
                        NO_SUB_MENU,                                    /*sub menu*/
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "Sig Figs",

                        0,                                             /*menu flags */
                        NO_SUB_MENU,                                    /*sub menu*/
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "Dec Places",

                        0,                                             /*menu flags */
                        NO_SUB_MENU,                                    /*sub menu*/
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "None",

                        MENU_WRIT+LAST_ITEM,
                        NO_SUB_MENU,
                        menuwriticon(MENUWI_FGCOL,MENUWI_BGCOL),
                        "dummy",                                    /* contents irrelev */
```

```
                        };

menu_block Clear_menu = {
                            "Clear",
                            stdmencols()
                            200,                                        /*menu width*/
                            40,                                        /*menu height*/
                            MENU_GAP,                                /*gap betweenentries*/
/* menu items ... */
                            0,                                          /*menu flags */
                            NO_SUB_MENU,                                  /*sub menu*/
                            menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                            "All charts",

                            0,                                          /*menu flags */
                            NO_SUB_MENU,                                  /*sub menu*/
                            menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                            "This chart",

                            0,                                          /*menu flags */
                            NO_SUB_MENU,                                  /*sub menu*/
                            menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                            "Graphs",

                            0,
                            NO_SUB_MENU,
                            menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                            "Run record",

                            LAST_ITEM,
                            NO_SUB_MENU,
                            menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                            "Variables",


                        };


static menu_block Super_menu = {
                            "Superimpose",
                            stdmencols()
                            200,                                        /*menu width*/
                            40,                                        /*menu height*/
                            MENU_GAP,                                /*gap betweenentries*/
/* menu items ... */
                            0,                                          /*menu flags */
                            NO_SUB_MENU,                                  /*sub menu*/
                            menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                            "chart num:",

                            DOT_LINE+MENU_WRIT,                         /*menu flags */
                            NO_SUB_MENU,                                  /*sub menu*/
                            menuwriticon(MENUWI_FGCOL,MENUWI_BGCOL),
                            "dummy",
```

```
                        LAST_ITEM,
                        NO_SUB_MENU,
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "Do it",

                );

menu_block Graph_window_menu = {

                        "Graphing",
                        stdmencols()
                        200,                                        /*menu width*/
                        40,                                        /*menu height*/
                        MENU_GAP,                                /*gap betweenentries*/


                        DOT_LINE,
                        NO_SUB_MENU,                        /* this is altered later */
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "Info..",


                        DOT_LINE,
                        NO_SUB_MENU,
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "Clear",


                        LAST_ITEM,                                /*menu flags */
                        &Super_menu,                                /*sub menu*/
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "Superimpose",

                       };


static menu_block Graph_menu = {
                        "Graph",
                        stdmencols()
                        200,                                        /*menu width*/
                        40,                                        /*menu height*/
                        MENU_GAP,                                /*gap betweenentries*/
/* menu items ... */
                        0,                                        /*menu flags */
                        NO_SUB_MENU,                                /*sub menu*/
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "chart num:",

                        DOT_LINE+MENU_WRIT,                        /*menu flags */
                        NO_SUB_MENU,                                /*sub menu*/
                        menuwriticon(MENUWI_FGCOL,MENUWI_BGCOL),
                        "dummy",

                        0,
                        NO_SUB_MENU,                                /*sub menu*/
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
```

```
                        "X min :",

                        DOT_LINE+MENU_WRIT,                                  /*menu flags */
                        NO_SUB_MENU,                                           /*sub menu*/
                        menuwriticon(MENUWI_FGCOL,MENUWI_BGCOL),
                        "dummy",

                        0,
                        NO_SUB_MENU,                                           /*sub menu*/
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "X max:",

                        DOT_LINE+MENU_WRIT,                                  /*menu flags */
                        NO_SUB_MENU,                                           /*sub menu*/
                        menuwriticon(MENUWI_FGCOL,MENUWI_BGCOL),
                        "dummy",

                        LAST_ITEM,
                        NO_SUB_MENU,
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "Do it",
                    };


menu_block Edit_menu= {
                        "Edit Menu",
                        stdmencols()
                        200,                                                  /*menu width*/
                        40,                                                  /*menu height*/
                        MENU_GAP,                                       /*gap betweenentries*/
/* menu items ... */
                        0,                                                    /*menu flags */
                        NO_SUB_MENU,                                           /*sub menu */
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "Append",                                                  /*text*/

                        0,                                                    /*menu flags */
                        NO_SUB_MENU,                                           /*sub menu */
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "Overwrite",                                               /*text*/


                        0,                                                    /*menu flags */
                        NO_SUB_MENU,                                           /*sub menu*/
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "Add After",

                        DOT_LINE,                                             /*menu flags */
                        NO_SUB_MENU,                                           /*sub menu*/
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "Add Before",

                        0,                                                    /*menu flags */
                        NO_SUB_MENU,                                           /*sub menu*/
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "Delete box",

                        0,                                                    /*menu flags */
```

```
                        &Clear_menu,                                    /*sub menu*/
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "Clear",


                        0,                                              /*menu flags */
                        NO_SUB_MENU,                                    /*sub menu*/
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "Run",


                        0,                                              /*menu flags */
                        &Graph_menu,                                    /*sub menu*/
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "Graph",

                        0,                                              /*menu flags */
                        NO_SUB_MENU,                                    /*sub menu*/
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "HELP",

                        0,                                              /*menu flags */
                        &Prefs_menu,                                    /*sub menu*/
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "Preferences",

                        0,                                              /* file menu */
                        &File_menu,
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "Filing",

                        LAST_ITEM,                                      /* file menu */
                        NO_SUB_MENU,
                        menutexticon(MENUTI_FGCOL,MENUTI_BGCOL),
                        "Variables",

                    };

/*
   Two sets of text : one for icon defs, and the other for
   inserting in the boxeditor screen...
   cbtext[] is available outside this module, and is therefore not static.
   cbtext is what appears in the screen for each button press
   cbtextdisp is used only to define the icons
*/


char* cbtext[CALC_ROWS*CALC_COLS] = {
                        "0",   ".",   "+/-",   "",
                        "1",   "2",   "3",   "+",
                        "4",   "5",   "6",   "-",
                        "7",   "8",   "9",   "x",
                        "SIN", "COS", "TAN", "/",
                        "ASN", "ACS", "ATN", "+RCL",
                        "LOG", "LN",  "^",   "*RCL",
                        "EXP", "1/x","10^x", "-RCL",
                        "STO", "RCL", "1/x","/RCL",
                        "",  "",   "",  "",
```

```
);

static int oplevel[CALC_ROWS*CALC_COLS] = {
                                0,0,0,0,
                                0,0,0,1,
                                0,0,0,1,                                    /* 456- */
                                0,0,0,1,                                    /* 789x */
                                3,3,3,1,                                /* SIN, COS, TAN, / */
                                3,3,3,4,                                /* ASN,ACS,ATN, +RCL */
                                3,3,3,4,                                        /* Log etc */
                                3,2,3,4,                            /* exp. sqroot, 10^x, -RCL */
                                4,4,2,4,                                /* STO,RCL,1/x, /RCL */
                            };

static char* cbtextdisp[CALC_ROWS*CALC_COLS] = {
                                " 0 ",   " . ",   "+/-",    "",
                                " 1 ",   " 2 ",   " 3 ",   " + ",
                                " 4 ",   " 5 ",   " 6 ",   " - ",
                                " 7 ",   " 8 ",   " 9 ",   " x ",
                                "SIN",   "COS",   "TAN",   " / ",
                                "ASN",   "ACS",   "ATN",   "+RCL",
                                "LOG",   "LN ",   " ^ ",   "*RCL",
                                "EXP",   "√x ",   "10ˣ", "-RCL",
                                "STO",   "RCL",    "1/x", "/RCL",
                                "",   "",   "",   "",
};


static void define_runentry_icons()
{
        numinput=malloc(30);
        numoutput=malloc(30);
        runflownum=malloc(4);

        strcpy(numinput,DFLT_NUMINPUT);
        make_text_icon(w_runentry,IN_OUT_X,IN_Y,
                        numinput,0,grey6,white,
                        IN_OUT_HT);


        strcpy(runflownum,DFLT_RUNFLOWNUM);
        make_text_icon(w_runentry,F_X,F_Y,
                        runflownum,0,red,black,0);


        strcpy(numoutput,DFLT_NUMOUTPUT);
        make_text_icon(w_runentry,IN_OUT_X,OUT_Y,
                        numoutput,0,grey6,white,
                        IN_OUT_HT);


        runbutton=make_text_icon(w_runentry,
                                100,
                                W_RUNENTRY_HEIGHT-100,
                                "RUN",
                                4,
                                dark_blue,
                                white,
```

```
                                64);


        clearinbutton=make_text_icon(w_runentry,
                                200,
                                W_RUNENTRY_HEIGHT-100,
                                "CLEAR",
                                4,
                                dark_blue,
                                white,
                                64);


        swapbutton=make_text_icon(w_runentry,
                                260,
                                F_X+60,
                                "IN<-OUT",
                                4,
                                dark_blue,
                                white,
                                0);


        autoswapbutton=make_text_icon(w_runentry,
                                50,
                                W_RUNENTRY_HEIGHT-200,
                                "Autoswap ",
                                4,
                                dark_blue,
                                white,
                                64);


        *numinput=END_OF_STRING;
        *numoutput=END_OF_STRING;
        *runflownum=END_OF_STRING;

}

static void define_calc_icons()
{
        int i,j,n;
        const int width=100,                                          /* of each icon */
                  height=50,                                          /* of each icon */
                  basex=50,
                  basey=50,
                  rows=CALC_ROWS,
                  cols=CALC_COLS;
        for(i=0;i<rows;i++) {
            for(j=0;j<cols;j++) {
                n=j+i*cols;
                if((*cbtextdisp[n]!=NULL)&&(oplevel[n]<=set_op_level))
                   calcbutton[n]=make_text_icon(w_boxedit,basex+j*width,
                                        basey+i*height,
                                        cbtextdisp[n],4,dark_blue,white,0);
                else calcbutton[n]=-1;
            }
        }
```

```
        calcscreen=make_writeable_icon(w_boxedit,
                                       80,W_BOXEDIT_HEIGHT-4*CHAR_HEIGHT,
                                       screen,SCREENLENGTH,
                                       96,dark_blue,white);

        clearbutton=make_text_icon(w_boxedit,
                                   30,W_BOXEDIT_HEIGHT-7*CHAR_HEIGHT,"CLR",4,
                                   dark_blue,white,64);

        enterbutton=make_text_icon(w_boxedit,
                                   120,W_BOXEDIT_HEIGHT-7*CHAR_HEIGHT,"ENT",4,
                                   dark_blue,white,64);
}



#define pos 10
#define allow_for(x)    (CHAR_WIDTH*x+30)


static void define_panel_icons()

{
        int cpr_icon;
        int xpos=10;

       exiticon=make_text_icon(w_panel,xpos,10,"Exit",4,dark_blue,white,64);
       xpos+=allow_for(4);
       runicon=make_text_icon(w_panel,xpos,10,"Run",4,dark_blue,white,64);
       xpos+=allow_for(3);
       flowicon=make_text_icon(w_panel,xpos,10,"Flows",4,dark_blue,white,64);
       xpos+=allow_for(5);
       calcicon=make_text_icon(w_panel,xpos,10,"Edit",4,dark_blue,white,64);
       xpos+=allow_for(4);
       helpicon=make_text_icon(w_panel,xpos,10,"Help",4,dark_blue,white,64);
       xpos+=allow_for(4);
       if(advanced_status) {                                          /* not normally allowed... */
          screensaveicon=make_text_icon(w_panel,xpos,10,"Save screen",4,
                                        dark_blue,white,64);
          xpos+=allow_for(10);
       }
       cpr_icon=make_text_icon(w_panel,800,10,"v 1.20 - M. Twells 1989",4,dark_green,white,64);

}

static void setup_file_menu()
{

#define SF save_menu.m[0].data.writeable

        savefilename=(char*)malloc(50);

        SF.text_buff=savefilename;
        SF.val_string=(char*)-1;
        SF.bufflen=50;

        strcpy(savefilename,"Flowfile");
```

```
#undef SF

}

static void setup_graph_menu()
{

#define Xmin Graph_menu.m[GM_XMIN].data.writeable
#define Xmax Graph_menu.m[GM_XMAX].data.writeable
#define Fnum Graph_menu.m[GM_FNUM].data.writeable

        Xminnum=(char*)malloc(12);
        Xmaxnum=(char*)malloc(12);
        Fgraphnum=(char*)malloc(12);

        Xmin.text_buff=Xminnum;
        Xmin.val_string=(char*)-1;
        Xmin.bufflen=12;

        Xmax.text_buff=Xmaxnum;
        Xmax.val_string=(char*)-1;
        Xmax.bufflen=12;

        Fnum.val_string=(char*)-1;
        Fnum.bufflen=12;
        Fnum.text_buff=Fgraphnum;

        strcpy(Xminnum,"0");
        strcpy(Xmaxnum,"10");
        strcpy(Fgraphnum,"1");

}


static void setup_super_menu()
{
#define  Snum  Super_menu.m[GSM_NUM].data.writeable

        Super_num=(char*)malloc(12);

        Snum.text_buff=Super_num;
        Snum.val_string=(char*)-1;
        Snum.bufflen=12;

        strcpy(Super_num,"1");
}

static void setup_prefs_menu()
{

#define AccD Prefs_menu.m[PM_ACCURACY].data.writeable

/* Accnumber set up in initialise */

        AccD.text_buff=Accnumber;
        AccD.val_string=(char*)-1;
        AccD.bufflen=12;
```

```
}

static void setup_vars_menu()
{

#define entry Name_menu.m[0].data.writeable
#define entry1 Crtvar_menu.m[0].data.writeable
#define entry2 Varval_menu.m[0].data.writeable

        del_var_name=(char*)malloc(12);
        crtvar_name=(char*)malloc(12);
        crtvar_val=(char*)malloc(12);

        entry.text_buff=del_var_name;
        entry.val_string=(char*)-1;
        entry.bufflen=12;
        *del_var_name=NULL;

        entry1.text_buff=crtvar_name;
        entry1.val_string=(char*)-1;
        entry1.bufflen=12;
        *crtvar_name=NULL;

        entry2.text_buff=crtvar_val;
        entry2.val_string=(char*)-1;
        entry2.bufflen=12;
        *crtvar_val=NULL;


#undef entry

}


void setup_winds()
{
        screen=malloc(20);
        *screen=NULL;
        w_flowcharts=usr_create_wind(&flowcharts);
        w_runrecord=usr_create_wind(&runrecord);
        w_help=usr_create_wind(&help_wind_block);
        w_panel=usr_create_wind(&control_panel);
        w_boxedit=usr_create_wind(&boxeditor);
        w_runentry=usr_create_wind(&runentry);
        w_graphinfo=usr_create_wind(&graphinfo);
        w_plotwindow=usr_create_wind(&plotwindow);
        w_vartable=usr_create_wind(&varbox);

        setup_file_menu();
        setup_prefs_menu();
        setup_graph_menu();
        setup_super_menu();
        setup_vars_menu();

        Help_menu=Edit_menu.m[EM_HELP].sub_menu=
                                setup_dir_menu_r("Help..",HELPDIRNAME);
```

```
        Load_menu=File_menu.m[FM_LOAD].sub_menu=
                                setup_dir_menu("Load chart",FLOWDIRNAME);


        if (!vars_allowed) {

                Edit_menu.m[EM_VARS].icon_flags|=NOSELECT;

/*change the word "Variables" in the clear menu to "Memory"..
  we are merely clearing the anonymous memory...
*/
                strcpy(Clear_menu.m[CM_VARS].data.text,"Memory");
        }

        if(!advanced_status) {
                File_menu.m[FM_RENAME].icon_flags|=NOSELECT;
                File_menu.m[FM_DELETE].icon_flags|=NOSELECT;
                Delete_menu=NULL;
                Rename_menu=NULL;

        } else {

          Delete_menu=File_menu.m[FM_DELETE].sub_menu=
                                setup_dir_menu("Delete..",FLOWDIRNAME);

          File_menu.m[FM_RENAME].sub_menu=
                                setup_rename_menu("Rename..",FLOWDIRNAME);

        }

        define_calc_icons();
        define_panel_icons();
        define_runentry_icons();
}
```

9.4.10 C.W_SHELL

```
/*>c.w_shell


a     a          aaaaa
a  a  a         a     a  a    a  aaaaaa  a       a
a  a  a         a        a    a  a       a       a
a  a  a          aaaaa   aaaaaa  aaaaa   a       a
a  a  a              a   a    a  a       a       a
a  a  a         a    a   a    a  a       a       a
 aa aa  aaaaaaa  aaaaa    a    a  aaaaaa  aaaaaa  aaaaaa


        Object module to do all the hard work of
        policing a wimp  environment

        MAIN routine is poll(), which calls routines
        according to user's actions

*/

#define ARTHUR_OLD_NAMES
#include <stdio.h>
#include <arthur.h>
#include <stdlib.h>
#include <string.h>
#include "winds.h"

#define lsb(x)  (char)(x&0xFF)
#define msb(x)  (char)((x>>8)&0xFF)

/*  reason codes returned by WIMP from poll... */

#define DO_NOTHING 0
#define DO_REDRAW  1
#define DO_OPEN    2
#define DO_CLOSE   3
#define DO_LEAVE   4
#define DO_ENTER   5
#define DO_MOUSE   6
#define DO_DRAG    7
#define DO_KEY     8
#define DO_MENU    9
#define DO_SCROLL  10


error *my_error;


/*********************************/
/* UNIV BLOCK Allocations for various
   routines.
   UNIV_BLOCK is the block used by POLL
   to pass information to various functions
   (The functions always declare the univ block as *block)
   The routines may or may not use block. They may be
```

```
   able to perform their tasks without referring to the
   information passed to them.
*/
/*********************************/

/* for mouse button presses ....  */

#define mousex    (block->a[0])
#define mousey    (block->a[1])
#define mouseb    (block->a[2])
#define mouseicon (block->a[4])


/*for key presses ...... */

#define keyicon    (block->a[1])
#define keycrtxoff (block->a[2])
#define keycrtyoff (block->a[3])
#define caretht    (block->a[4])
#define crtindex   (block->a[5])
#define keycode    (block->a[6])


/* for menu select ............. */

#define item(i)         (block->a[i])


/* for redrawing...... */

#define xmin            (block->x0)
#define xmax            (block->x1)
#define ymin            (block->y0)
#define ymax            (block->y1)


/*****************************************************/
/* Prototypes                                        */
/*****************************************************/


typedef struct poll_return
{
        int reason;                                          /* reason code */
        univ_block block;                                  /* associated data */

}poll_return;


static poll_return *one_poll(int mask);
static poll_return prb;                                /* space used for return of data */



typedef struct wkidx {
              int wind_handle;                                      /*FOR EACH WINDOW...*/
              void (*key_fn)(univ_block*);                    /*Function to visit for keys */
              void (*mse_fn)(univ_block*);                    /* ..and mouse button presses*/
```

```
            void (*rdw_fn)(redraw_block*);                    /* ..and redrawing the window*/
            void (*ent_fn)(univ_block*);                        /*.. entering a window */
            void (*lve_fn)(univ_block*);                         /*..leaving a window */
            void (*close_fn)(void);                               /*..before closing */
            void (*open_fn)(void);                                 /* before opening */
            } wkidx;

#define max_num_windows 32
wkidx wkey_fns[max_num_windows];                              /*Max 32 windows, this system*/


void (*menu_decode)(univ_block*)= NULL;    /* will be set to point to function to decode menu currently UP */
void (*claimpoll)(univ_block*)=NULL;                /* to point to function to be called on poll, if any */
void (*bg_mouse_fn)(univ_block*)=NULL;                                    /* mouse on background */

int max_wkidx = 0;
static char *taskname;

/*************************************************/
/* Prototypes                                    */
/*************************************************/

void check_error(char*,error*),
     do_error(char*),
     redraw_window(int),
     do_decode_menu(univ_block*),
     get_wind_info(wind_info_block*,error*),
     originlwa(int,int*,int*),
     adjust_to_origin(int,univ_block*);


int
     is_icon_selected(int,int);


extern void
       w_redraw(redraw_block*),
       m_select(univ_block*),
       usr_leave_wind(int),
       usr_enter_wind(int);

void open_wimp(int bg)
{
/* initialise the WIMP environment.
   Begin a non-multitasking program
*/
        int i;
        for(i=0;i<max_num_windows;i++) {
                wkey_fns[i].wind_handle=NULL;
                wkey_fns[i].key_fn=NULL;
                wkey_fns[i].mse_fn=NULL;
                wkey_fns[i].rdw_fn=NULL;
                wkey_fns[i].ent_fn=NULL;
                wkey_fns[i].lve_fn=NULL;
                wkey_fns[i].close_fn=NULL;
                wkey_fns[i].open_fn=NULL;
        }
```

```
        vdu(19);
        vdu(15);
        vdu(16);
        vdu(0x70);
        vdu(0x70);
        vdu(0x70);

        w_initialise(my_error);

}

int find_fn(int handle)
{
/* Find the function record relating to the window whose handle is given.
   return the index into the array of such records
*/
        int i=0;
        if(handle==-1) return(-1);
        while(i<=max_wkidx && wkey_fns[i].wind_handle!=handle) i++;
        if (wkey_fns[i].wind_handle==handle) return(i);
        else return(-1);
}


void check_error(char *name, error *errorp)
{
/* if there has been an error, report the fact to the user.
   name is printed in the title area
*/
        reg_set regs;
        if(errorp->errnum!=0) {
                vdu(7);
                mode(12);
                printf("%s :",name);
                printf("Error %d, %s\n",errorp->errnum,errorp->errmess);
                swix(0x400DD,&regs);
                exit(0);
        }
}


/*****************************************************/
/* ROUTINES to set up functions to perform tasks     */
/* for each window in the user program               */
/*****************************************************/



void claim_poll_function(fn)
void(*fn)(univ_block*);
{
/* fn is to be called on every poll loop from now onwards
*/
       claimpoll=fn;
}


void close_window_function(handle,fn)
```

```
int handle;
void (*fn)(void);
{

/* fn is to be called when window whose handle is handle is about to close */

        int idx=find_fn(handle);
        if (idx==-1) {
             wkey_fns[max_wkidx].wind_handle=handle;
             wkey_fns[max_wkidx++].close_fn=fn;
        } else {
             wkey_fns[idx].close_fn=fn;
        }
}

void open_window_function(handle,fn)
int handle;
void (*fn)(void);
{

/* fn is to be called when window whose handle is handle is about to open */

        int idx=find_fn(handle);
        if (idx==-1) {
             wkey_fns[max_wkidx].wind_handle=handle;
             wkey_fns[max_wkidx++].open_fn=fn;
        } else {
             wkey_fns[idx].open_fn=fn;
        }
}

void key_pressed_function(handle,fn)
int handle;
void (*fn)(univ_block*);
{
/* assign the function to cover key presses for this window */

        int idx=find_fn(handle);
        if (idx==-1) {
             wkey_fns[max_wkidx].wind_handle=handle;
             wkey_fns[max_wkidx++].key_fn=fn;
        } else {
             wkey_fns[idx].key_fn=fn;
        }
}

void wind_enter_function(handle,fn)
int handle;
void (*fn)(univ_block*);
{

/* call fn when the pointer enters the window */

        int idx=find_fn(handle);
        if (idx==-1) {
             wkey_fns[max_wkidx].wind_handle=handle;
             wkey_fns[max_wkidx++].ent_fn=fn;
        } else {
```

```
            wkey_fns[idx].ent_fn=fn;
        }
}

void wind_leave_function(handle,fn)
int handle;
void (*fn)(univ_block*);
{

/* call the function fn when the pointer leaves the window */

        int idx=find_fn(handle);
        if (idx==-1) {
            wkey_fns[max_wkidx].wind_handle=handle;
            wkey_fns[max_wkidx++].lve_fn=fn;
        } else {
            wkey_fns[idx].lve_fn=fn;
        }
}



void window_redraw_function(handle,fn)
int handle;
void (*fn)(redraw_block*);
{

/* Function fn redraws the  window whose hande is handle*/

        int idx=find_fn(handle);
        if (idx==-1) {
            wkey_fns[max_wkidx].wind_handle=handle;
            wkey_fns[max_wkidx++].rdw_fn=fn;
        } else {
            wkey_fns[idx].rdw_fn=fn;
        }
}


void menu_decode_function(fn)
void (*fn)(univ_block*);
{
        /* fn is a function to decode menus */
        menu_decode=fn;
}

void background_mouse_function(fn)
void (*fn)(univ_block*);
{
        bg_mouse_fn=fn;
}

void mse_pressed_function(handle,fn)
int handle;
void (*fn)(univ_block*);
{

/* assign the function to cover mouse button presses for this window */
```

```
        int idx=find_fn(handle);
        if (idx==-1) {
             wkey_fns[max_wkidx].wind_handle=handle;
             wkey_fns[max_wkidx++].mse_fn=fn;
        } else {
             wkey_fns[idx].mse_fn=fn;
        }
}


static poll_return *one_poll(int mask)
{
/* call the WIMP poll routine once, and process the action according to functions
   defined in the preceding section
*/

             int idx;

             prb.reason=poll_wimp(mask,&prb.block,my_error);
             if(my_error) wimp_error(my_error,WE_OK);
             if(claimpoll!=NULL) (*claimpoll)(&prb.block);
             switch(prb.reason) {
                  case DO_NOTHING:
                         break;

                  case DO_REDRAW:
                         redraw_window(prb.block.w.wind_handle);
                         break;

                  case DO_OPEN:
                         open_wind(&prb.block.w,my_error);
                         if(my_error) wimp_error(my_error,WE_OK);
                         break;

                  case DO_CLOSE:
                         idx=find_fn(prb.block.a[0]);
                         if((idx!=-1)&&(wkey_fns[idx].close_fn!=NULL))
                                (*wkey_fns[idx].close_fn)();
                         close_wind(prb.block.w.wind_handle,my_error);
                         if(my_error) wimp_error(my_error,WE_OK);
                         break;


                  case DO_LEAVE:                                          /* leaving window */
                         idx=find_fn(prb.block.a[0]);
                         if((idx!=-1)&&(wkey_fns[idx].lve_fn!=NULL))
                                (*wkey_fns[idx].lve_fn)(&prb.block);
                         break;


                  case DO_ENTER:                                          /* entering window */

                         idx=find_fn(prb.block.a[0]);
                         if((idx!=-1)&&(wkey_fns[idx].ent_fn!=NULL))
                                (*wkey_fns[idx].ent_fn)(&prb.block);
                         break;
```

```
                case DO_MOUSE:                                          /* MOUSE Button press */
                        idx=find_fn(prb.block.a[3]);
/* found something... */
                        if((idx!=-1)&&(wkey_fns[idx].mse_fn!=NULL))
                            (*wkey_fns[idx].mse_fn)(&prb.block);
                        else if((prb.block.a[3]==-1)&&(bg_mouse_fn!=NULL))
                            (*bg_mouse_fn)(&prb.block);
                      break;

                case DO_DRAG:
                       break;

                case DO_KEY:       /* handle a key press .. check to see if we know what to do with keys */
                       idx=find_fn(prb.block.a[0]);
                       if((idx!=-1)&&(wkey_fns[idx].key_fn!=NULL))
                           (*wkey_fns[idx].key_fn)(&prb.block);
                      break;


                case DO_MENU:
                      if(menu_decode!=NULL) (*menu_decode)(&prb.block);
                      break;

                case DO_SCROLL:
                      break;

                default:
                     break;
              }

        return(&prb);
}



void poll(int mask)
{
        /* call poll repeatedly */

        do one_poll(mask); while(TRUE);
}



void adjust_to_origin(int window, univ_block *block)
{

/* block contains graphics coordinates..adjust to window coords */

     int xco,yco;
     originlwa(window,&xco,&yco);
     block->a[0]-=xco;
     block->a[1]-=yco;
     block->a[2]-=xco;
     block->a[3]-=yco;
}
```

```
void input_focus_to(int handle)
{
        caret_block cblock;
        mouse_block mblock;
        get_point_info(&mblock,my_error);
        cblock.wind_handle=handle;
        cblock.icon_handle=-1;
        cblock.cx=mblock.mx;
        cblock.cy=mblock.my;
        cblock.height=-1;
        cblock.index=-1;
        set_caret_pos(&cblock,my_error);
        if(my_error) wimp_error(my_error,WE_OK);
}



void redraw_window(int handle)
{
        redraw_block block;
        int flag,xo,yo,idx;
        block.wind_handle=handle;
        origin(0,0);
        flag=redraw_wind(&block,my_error);
        if(my_error) wimp_error(my_error,WE_OK);
        while(flag && !my_error) {
                    xo=block.x0-block.scx;
                    yo=block.y1-block.scy;
                    origin(xo,yo);

                    block.gx0-=xo;
                    block.gy0-=yo;
                    block.gx1-=xo;
                    block.gy1-=yo;

                    idx=find_fn(block.wind_handle);
                    if((idx!=-1)&&(wkey_fns[idx].rdw_fn!=NULL))
                            (*wkey_fns[idx].rdw_fn)(&block);
                    origin(0,0);
                    flag=get_rectangle(&block,my_error);
                    if(my_error) wimp_error(my_error,WE_OK);
        }
}


void originlwa(int handle, int *xco, int *yco)

{
        /* setup the origin of the local work area in *xco, and *yco */
        wind_info_block wiblock;
        wiblock.wind_handle=handle;
        get_wind_info(&wiblock,my_error);
        *xco=wiblock.x0-wiblock.scx;
        *yco=wiblock.y1-wiblock.scy;
}
```

```
void usr_close_wind(int handle)
{

/* Close the window whose handle is given */

          close_wind(handle,my_error);
          if(my_error) wimp_error(my_error,WE_OK);
}


int usr_create_wind(wind_block *wblock)
{

/* create the window whose block is given, returning the window handle */

        int ret;
        ret=create_wind(wblock,my_error);
        if(my_error) wimp_error(my_error,WE_OK);
        return(ret);
}


void usr_open_wind(int handle, int wind_behind)
{

/* OPEN the window in the position given by wind_behind , ON_TOP = on top */

        open_block oblock;
        get_wind_state(handle,&oblock,my_error);
        oblock.n_handle=wind_behind;
        open_wind(&oblock,my_error);
        if(my_error) wimp_error(my_error,WE_OK);
}

void usr_open_wind_scroll(int handle, int wind_behind, int sx, int sy)
{
        open_block oblock;
        get_wind_state(handle,&oblock,my_error);
        oblock.n_handle=wind_behind;
        oblock.x=sx;                                          /* Add in scroll bar postions */
        oblock.y=sy;
        open_wind(&oblock,my_error);
        if(my_error) wimp_error(my_error,WE_OK);
}



void get_wind_info(wind_info_block *wiblock, error *err)
{
/* obtainm up-to-date information about the window */

        reg_set regs;
        regs.r[1]=(int)wiblock;
        err=swix(GetWindowInfo,&regs);
}
```

```
void move_mouse_to(int x, int y)
{

/* Move the mouse to the given screen coordinate */

        char block[10];
        reg_set regs;
        block[0]=3;
        block[1]=lsb(x);
        block[2]=msb(x);
        block[3]=lsb(y);
        block[4]=msb(y);
        regs.r[0]=21;
        regs.r[1]=(int)block;
        swi(OS_Word,&regs);
}


void put_sprite(char *name, int action, char *sp_area)
{
        reg_set regs;
        regs.r[0]=256+28;
        regs.r[1]=(int)sp_area;
        regs.r[2]=(int)name;
        regs.r[3]=action;
        regs.r[5]=8;
        my_error=swix(OS_SpriteOp,&regs);
        if(my_error!=NULL) {
              wimp_error(my_error,WE_OK);
              exit(0);
        }
}


/*****************************************************************/
/* Miscellaneous routines.....                                   */
/*****************************************************************/


void force_update(int handle, int xl, int yl, int xh, int yh)
{

/* mark a portion of window as invalid, ready for replotting */

        redraw_block rblock;
        rblock.wind_handle=handle;
        rblock.x0=xl;
        rblock.y0=yl;
        rblock.x1=xh;
        rblock.y1=yh;
        force_redraw(&rblock,my_error);
        if(my_error) wimp_error(my_error,WE_OK);
}


void usr_update_wind(int handle)
```

```
{
        redraw_block rblock;
        wind_info_block wiblock;

/*
   Mark the whole window as invalid, ready for redrawing (once returned to  poll loop)
*/

        wiblock.wind_handle=handle;
        get_wind_info(&wiblock,my_error);
        if(my_error) wimp_error(my_error,WE_OK);
        rblock.wind_handle=handle;
        rblock.x0=wiblock.exx0;                                  /* window extent, rel to window orig */
        rblock.y0=wiblock.exy0;
        rblock.x1=wiblock.exx1;
        rblock.y1=wiblock.exy1;
        force_redraw(&rblock,my_error);
        if(my_error) wimp_error(my_error,WE_OK);
}

void usr_update_scroll_wind(int handle, int topleft_x, int topleft_y)
{
        open_block oblock;
        wind_info_block wiblock;

/*       change position of scroll bars and redraw the window   */

        wiblock.wind_handle=handle;
        get_wind_info(&wiblock,my_error);
        if(my_error) wimp_error(my_error,WE_OK);
        oblock.wind_handle=handle;
        oblock.x0=wiblock.x0;                                   /* window extent, rel to window orig */
        oblock.y0=wiblock.y0;
        oblock.x1=wiblock.x1;
        oblock.y1=wiblock.y1;
        oblock.x=topleft_x-200;
        oblock.y=topleft_y+150;
        oblock.n_handle=-1;                                          /* re-open window on top */
        close_wind(handle,my_error);
        open_wind(&oblock,my_error);
        if(my_error) wimp_error(my_error,WE_OK);
}



int intersect(redraw_block *block, int x0, int y0, int x1, int y1)
{

/* returns TRUE if the rectangle intersects the redraw block,   else FALSE    */

        if((x0>block->gx1)||
           (x1<block->gx0)||
           (y1<block->gy0)||
           (y0>block->gy1)) return(FALSE);

        else return(TRUE);
```

```
}

/* items checking window status... */


int is_window_open(int handle)
{
        /* return TRUE if the window is open */

        open_block oblock;
        get_wind_state(handle,&oblock,my_error);
        if(my_error) wimp_error(my_error,WE_OK);
        return((oblock.flags&OPEN)!=0);
}


int is_window_top(int handle)
{

/* TRUE if the window is on top of the pile */

        open_block oblock;
        get_wind_state(handle,&oblock,my_error);
        if(my_error) wimp_error(my_error,WE_OK);
        return(oblock.flags&TOP!=0);
}


int is_window_fullsize(int handle)
{

/* TRUE is the window is fullsize */

        open_block oblock;
        get_wind_state(handle,&oblock,my_error);
        if(my_error) wimp_error(my_error,WE_OK);
        return(oblock.flags&FULL_SIZE!=0);

}


void set_window_extent(int handle, int xb, int yl, int xt, int yr)
{

/* change the size of the window work area to that given */

        redraw_block block;
        block.wind_handle=handle;
        block.x0=xb;
        block.y0=yl;
        block.x1=xt;
        block.y1=yr;
        set_extent(&block,my_error);
        if(my_error) wimp_error(my_error,WE_OK);
}


void scroll_window(int handle, int xd, int yd)
```

```
{

/* distance is number of pixel rows,cols  */

        wind_info_block wiblock;
        redraw_block block;
        int flag;
        block.wind_handle=handle;
        wiblock.wind_handle=handle;
        get_wind_info(&wiblock,my_error);
        block.x0=wiblock.exx0;
        block.y0=wiblock.exy0;
        block.x1=wiblock.exx1;
        block.y1=wiblock.exy1;
        flag=update_wind(&block,my_error);                          /* rtn all visible rectangles */
        if(my_error) wimp_error(my_error,WE_OK);
        while(flag) {
                    move(block.gx1,block.gy1);
                    move(block.gx0,block.gy0);
                    plot(185,xd,yd);
                    flag=get_rectangle(&block,my_error);
                    if (my_error) wimp_error(my_error,WE_OK);
        }
}


void put_caret(int handle, int x, int y)
{

/* position the writeable caret somewhere */

        caret_block cblock;
        cblock.wind_handle=handle;
        cblock.icon_handle=-1;
        cblock.cx=x;
        cblock.cy=y;
        cblock.height=32+(1<<24);
        cblock.index=-1;
}


/*********************************************************************/
/* ERROR functions */
/*********************************************************************/

int wimp_error(error *e, int flags)
{

/* There has been an error, the message is e, flags determines what sort of box and
   what sort of response to get from the user
*/

        reg_set regs;
        regs.r[0]=(int)e;
        regs.r[1]=flags;
        regs.r[2]=(int)taskname;
        swix(Wimp_ReportError,&regs);
```

```
        return(regs.r[1]);
}


void do_error_num(int num, char *text)
{

/* pass on error text, number num to the user */

        error a;
        a.errnum=num;
        strcpy(a.errmess,text);
        wimp_error(&a,WE_OK);
}


void do_error(char *text)
{
        /* Here, just the text is passed on */

        do_error_num(0,text);
}

void set_taskname(char *name)
{

/* Set the taskname to given name, used for multitasking
   and other bits and pieces (e.g. error windows) */
        taskname=name;
}


/***********************************
 Icons
***********************************/




void make_unselectable(int icon, int window)
{

/* blank out the icon */

        istate_block isblock;
        isblock.wind_handle=window;
        isblock.icon_handle=icon;
        isblock.a[0]=NOSELECT;
        isblock.a[1]=NOSELECT;                          /* equiv to SET the NOSELECT bit */
        set_icon_state(&isblock,my_error);
}


void make_selectable(int icon, int window)
{

/* ensure icon can be selected */
```

```
        istate_block isblock;
        isblock.wind_handle=window;
        isblock.icon_handle=icon;
        isblock.a[0]=0;                                                    /* EOR with this */
        isblock.a[1]=NOSELECT;                                  /* BIC with this first =Unset the bit*/
        set_icon_state(&isblock,my_error);
}



int is_icon_selected(int icon, int window)
{

/* returns TRUE if icon is selected */

        istate_block isblock;
        isblock.wind_handle=window;
        isblock.icon_handle=icon;
        get_icon_state(&isblock,my_error);
        if (my_error) wimp_error(my_error,WE_OK);
        return((isblock.a[4]&SELECTED)!=0);
}


int make_sprite_icon(int whandle, int xco, int yco, int width,
                     int height, char *sname, char *scblock, int type)
{

/* turn the sprite into an icon */

        int ihandle;
        icon_block iblock;
        iblock.wind_handle=whandle;
        iblock.ix0=xco;
        iblock.iy0=yco;
        iblock.ix1=xco+width;
        iblock.iy1=yco+height;
        iblock.flags=INDIRECT+SPRITE+(type<<12)+(3<<24)+(15<<28);
        iblock.data.writeable.text_buff=sname;
        iblock.data.writeable.val_string=scblock;
        iblock.data.writeable.bufflen=strlen(sname);
        ihandle=create_icon(&iblock,my_error);
        if (my_error) wimp_error(my_error,WE_OK);
        return(ihandle);
}


int make_text_icon(int whandle, int xco, int yco, char *text, int type,
                   int fg, int bg, int height)
{

/* turn the text into an icon */

        int ihandle;
        icon_block iblock;
        iblock.wind_handle=whandle;
        iblock.ix0=xco;
```

```
        iblock.iy0=yco;
        iblock.ix1=xco+(strlen(text)<<4)+20;

        if(height<=48)  iblock.iy1=yco+48;
        else iblock.iy1=yco+height;

        iblock.flags=HOR_CENTRE+
                     VER_CENTRE+
                     FILLED+
                     TEXT+
                     BORDER+
                     INDIRECT+
                     (type<<12)+
                     (bg<<24)+
                     (fg<<28);
        iblock.data.writeable.text_buff=text;
        iblock.data.writeable.val_string=(char*)-1;
        iblock.data.writeable.bufflen=strlen(text);
        ihandle=create_icon(&iblock,my_error);
        if (my_error) wimp_error(my_error,WE_OK);
        return(ihandle);
}


int     make_writeable_icon(int whandle, int xco, int yco, char *text,
                            int length, int height, int fg, int bg)

{

/* create an icon into which the user can write things */

        int ihandle;
        icon_block iblock;

        length+=1;
        iblock.wind_handle=whandle;
        iblock.ix0=xco;
        iblock.iy0=yco;
        iblock.ix1=xco+(length<<4)+16;
        iblock.iy1=yco+height;

        iblock.flags=FILLED+TEXT+BORDER+INDIRECT+(15<<12)+(fg<<24)+(bg<<28);
        iblock.data.writeable.text_buff=text;
        iblock.data.writeable.val_string=(char*)-1;
        iblock.data.writeable.bufflen=length;
        ihandle=create_icon(&iblock,my_error);
        if (my_error) wimp_error(my_error,WE_OK);
        return(ihandle);
}




/*

Menus....
```

```
==========

*/


void usr_create_menu(block,xp,yp,fn)
menu_block *block;
int xp,yp;
void (*fn)(univ_block*);

{

/* create a menu, and set up the function to decode it */

        menu_decode_function(fn);                                   /*set up which function decodes */
        create_menu(block,xp,yp,my_error);
        if (my_error) wimp_error(my_error,WE_OK);
}
```

9.4.11 ASM.F_ASM

```
;>asm.f_asm

;this version 1.00 26/8/89 (C) M. Twells

;Assembly language support for WIMP environment

;These routines are all hardware-dependent, and are here because
;they are unpleasant to express in C, and are rather easier to deal
;with here (requiring spurious structures in C)
;..moving them all here also removes some machine-dependence from the
; C source code.

        module
        area C$Code,code,readonly
        import do_error
        export close_wimp,is_a_dir
        export fextent,settype
        export a_open_wimp,rename_file,delete_file

;software hooks...

Wimp_CloseDown      EQU &400DD
Wimp_Initialise     EQU &400C0
OS_FSControl        EQU &29
OS_File             EQU 8
OS_ReadVduVariables EQU &31

          align

;************** S T A R T    OF  C O D E  *****************

close_wimp                      ;shut it all down

;close down WIMP environment

        swi Wimp_CloseDown
        mov pc,lr

;-----------------------------

a_open_wimp

;Open the WIMP environment up, for RISCOS

        mov 0,#200              ;This won't work prior to WIMP version 2.00
        mov 1,#0                ;not multitasking ("TASK" here if is)
        mov 2,#0                ;no need for name of task (^ name if is )
        swi XWimp_Initialise
        movvc 0,#0              ;no error, make it null
        mov pc,lr

;------------------------

rename_file

;old file name in 0, new one in 1
```

```
     mov 2,1              ;shuffle parameters up one register
     mov 1,0
     mov 0,#25            ;25 is code for renaming objects
     swi XOS_FSControl    ;error returns as usual in R0
     movvc 0,#0           ;set NULL if no error
     mov pc,lr


;----------------------------

delete_file

;file pathname for deletion in r0 on call

     stmfd sp!,{1-6,lr}
     mov 1,0              ;move filename to 1
     mov 0,#6             ;6 is code for deleting objects, PRM vol I page 236
     swi XOS_File         ;error returns as usual in R0
     movvc 0,#0           ;set NULL if no error
     ldmfd sp!,{1-6,pc}

;----------------------------

settype

;filename pointed at by r0, file type in 1

          mov 2,1         ;put file type in 2
          mov 1,0         ;put file name in 1
          mov 0,#&12      ;code for set type
          swi XOS_File
          movvc 0,#0
          mov pc,lr


;----------------------------

fextent

;r0 = pointer to name
;return with r0 = length

        stmfd sp!,{1-6,lr}
        mov 1,0                 ;xfer name pointer to r1
        mov 0,#5                ;=read catalogue info
        swi XOS_File
        mov 0,4                 ;file extent into 0
        ldmfd sp!,{1-6,pc}


;----------------------------


is_a_dir

;r0=pointer to name
;returns TRUE if it is a directory, else FALSE
```

```
        stmfd sp!,{1-6,lr}
        mov 1,0                 ;set up parameters to
        mov 0,#5                ;read catalogue info
        swi XOS_File
        cmp 0,#2
        moveq 0,#1              ;False...
        movne 0,#0              ;TRUE
        ldmfd sp!,{1-6,pc}
```

9.4.12 ASM.HOURGLASS

```
;>ASM.hourglass
;routines for hourglass module
        module
        area C$Code,code,readonly

     ;Void...
        export Hg_On
        export Hg_Off
        export Hg_Smash
     ;take one integer argument...
        export Hg_Start
        export Hg_Percent
     ;take 2 integer arguments
        export Hg_LEDs

Hourglass_On            EQU &406C0
Hourglass_Off           EQU &406C1
Hourglass_Start         EQU &406C3
Hourglass_Smash         EQU &406C2
Hourglass_Percentage    EQU &406C4
Hourglass_LEDs          EQU &406C5

Hg_On   SWI Hourglass_On
        mov pc,lr

Hg_Off  SWI Hourglass_Off
        mov pc,lr

Hg_Smash
        SWI Hourglass_Smash
        mov pc,lr
Hg_Start
        SWI Hourglass_Start
        mov pc,lr

Hg_Percent
        SWI Hourglass_Percentage
        mov pc,lr

Hg_LEDs
        SWI Hourglass_LEDs
        mov pc,lr
```

9.4.13 Index to C routines in listings

Index into Listings, by routine name :
(Files are listed at about 50 lines per page)

Function Name	Line no.	Defining file	First line of definition
Edit_menu_decode	447	c.f_main	void Edit_menu_decode(univ_block *block)
add_to_pathname	45	c.f_help	static void add_to_pathname(char *directory)
adjust_to_origin	451	c.w_shell	void adjust_to_origin(int window, univ_block *block)
allow_for	833	c.f_setup	#define allow_for(x) (CHAR_WIDTH*x+30)
append_box	941	c.f_main	static void append_box(flow_box_type *current, flow_box_type *box)
arrowhead	1400	c.f_ops	static void arrowhead(int orient)
background_mouse_function	335	c.w_shell	void background_mouse_function(fn)
calcparams	113	c.f_plot	static void calcparams(int num, double *x, double *y)
charsize	537	c.f_main	void charsize(int x, int y)
check_bounds	659	c.f_ops	static double check_bounds(double n, int fnum, int bnum,int *evaler
check_error	197	c.w_shell	void check_error(char *name, error *errorp)
check_range	650	c.f_ops	static double check_range(double n, double lower, double upper,int
check_syntax	304	c.f_ops	static int check_syntax(char *buf)
claim_poll_function	221	c.w_shell	void claim_poll_function(fn)
clear_graph_index	258	c.f_main	static void clear_graph_index(void)
clear_graphs	1215	c.f_main	void clear_graphs(void)
clear_menu	357	c.f_main	static void clear_menu(int choice)
clear_variables	585	c.f_ops	void clear_variables(void)
clearrunrecord	1150	c.f_ops	void clearrunrecord(void)
clearup	1415	c.f_main	static void clearup(int code)
close_window_function	230	c.w_shell	void close_window_function(handle,fn)
construct_error_message	369	c.f_ops	static void construct_error_message(char *message, char *tb, char *
create_variable_menu	484	c.f_ops	static void create_variable_menu(univ_block *block, int newvar)
d_to_r	440	c.f_ops	static double d_to_r(double x)
dbp	1859	c.f_ops	static int dbp(double num)
define_calc_icons	796	c.f_setup	static void define_calc_icons()
define_panel_icons	836	c.f_setup	static void define_panel_icons()
define_runentry_icons	727	c.f_setup	static void define_runentry_icons()
delete_box	901	c.f_main	void delete_box(flow_box_type *current)
delete_menu	194	c.f_main	void delete_menu(univ_block *block)
do_error	827	c.w_shell	void do_error(char *text)
do_error_num	815	c.w_shell	void do_error_num(int num, char *text)
do_graph	1266	c.f_main	void do_graph(int chartnum, double xmin, double xmax)
do_label	86	c.f_plot	static void do_label(void)
doplot	96	c.f_plot	static void doplot(int numpoints, double *x, double *y, int col)
drawbox	675	c.f_main	static void drawbox(int flownum, flow_box_type *this_box, int xp,in
ensure_box_is_visible	1072	c.f_main	static int ensure_box_is_visible(int cur_flow, int cur_box)
entering_boxeditor	1175	c.f_main	static void entering_boxeditor(univ_block *block)
entering_flows	1191	c.f_main	static void entering_flows(univ_block *block)
evaluate	678	c.f_ops	double evaluate(double input, int fnum, flow_box_type *box,
exists	1682	c.f_ops	static int exists(char *fname)
fgetdouble	1643	c.f_ops	static double fgetdouble(FILE *file)
fgetname	1658	c.f_ops	void fgetname(FILE *file)
fgetword	1666	c.f_ops	static int fgetword(FILE *file)
file_menu	314	c.f_main	static void file_menu(univ_block *block)
find_fn	184	c.w_shell	int find_fn(int handle)
find_index	1254	c.f_main	static int find_index(int num)
force_update	620	c.w_shell	void force_update(int handle, int xl, int yl, int xh, int yh)

```
fputdouble              1614 c.f_ops      static void fputdouble(FILE *file, double num)
fputword                1629 c.f_ops      static void fputword(FILE *file, int num)
frac                    1849 c.f_ops      static double frac(double num)
get_wind_info            571 c.w_shell    void get_wind_info(wind_info_block *wiblock, error *err)
gpos_y                  1431 c.f_ops      static int gpos_y(void)
graph_menu               293 c.f_main     static void graph_menu(univ_block *block)
graph_plot                61 c.f_plot     void graph_plot(int numpoints, double *x, double *y,
graph_super_menu         243 c.f_main     void graph_super_menu(univ_block *block)
graph_window_closing    1226 c.f_main     void graph_window_closing(void)
graph_window_menu        268 c.f_main     static void graph_window_menu(univ_block *block)
help_menu                257 c.f_help     void help_menu(univ_block * block)
in_betweeny              232 c.f_plot     static void in_betweeny(int *x)
initialise              1466 c.f_main     static void initialise(void)
initialise_functions    1628 c.f_main     static void initialise_functions(void)
initialise_screen       1617 c.f_main     static void initialise_screen(void)
input_focus_to           466 c.w_shell    void input_focus_to(int handle)
insert_after_box         965 c.f_main     static void insert_after_box(flow_box_type *current,flow_box_type *
insert_before_box       1000 c.f_main     static void insert_before_box(flow_box_type *current,flow_box_type
intersect                683 c.w_shell    int intersect(redraw_block *block, int x0, int y0, int x1, int y1)
is_function_key         1064 c.f_ops      static int is_function_key(int code)
is_icon_selected         880 c.w_shell    int is_icon_selected(int icon, int window)
is_op                    346 c.f_ops      static int is_op(char ch)
is_pathname              148 c.f_main     int is_pathname(char *p)
is_token                 206 c.f_ops      static int is_token(char *buf)
is_window_fullsize       723 c.w_shell    int is_window_fullsize(int handle)
is_window_open           700 c.w_shell    int is_window_open(int handle)
is_window_top            711 c.w_shell    int is_window_top(int handle)
item                      82 c.w_shell    #define item(i)         (block->a[i])
key_on_boxeditor        1105 c.f_main     static void key_on_boxeditor(univ_block *block)
key_on_runentry         1271 c.f_ops      void key_on_runentry(univ_block *block)
key_on_runrecord        1382 c.f_ops      void key_on_runrecord(univ_block *block)
key_pressed_function     262 c.w_shell    void key_pressed_function(handle,fn)
leave_program           1711 c.f_main     void leave_program(void)
leaving_boxeditor       1184 c.f_main     static void leaving_boxeditor(univ_block *block)
leaving_flows           1200 c.f_main     static void leaving_flows(univ_block *block)
length_of_flowchart     1569 c.f_ops      int length_of_flowchart(int number)
load_flowcharts         1775 c.f_ops      void load_flowcharts(char *fname)
load_menu                164 c.f_main     void load_menu(univ_block *block)
load_sprites            1397 c.f_main     static void load_sprites(char *name)
lotsa_ops                357 c.f_ops      static int lotsa_ops(char *b)
lsb                       29 c.w_shell    #define lsb(x)  (char)(x&0xFF)
main                    1757 c.f_main     int main(int argc, char *argv[])
make_selectable          865 c.w_shell    void make_selectable(int icon, int window)
make_sprite_area        1378 c.f_main     static char* make_sprite_area(int size)
make_sprite_icon         894 c.w_shell    int make_sprite_icon(int whandle, int xco, int yco, int width,
make_text_icon           917 c.w_shell    int make_text_icon(int whandle, int xco, int yco, char *text, int t
make_unselectable        851 c.w_shell    void make_unselectable(int icon, int window)
make_writeable_icon      951 c.w_shell    int     make_writeable_icon(int whandle, int xco, int yco, char *te
menu_decode_function     328 c.w_shell    void menu_decode_function(fn)
mouse_elsewhere         1369 c.f_main     void mouse_elsewhere(univ_block *block)
mouse_enters_runentry   1035 c.f_ops      void mouse_enters_runentry(univ_block *block)
mouse_enters_runrecord  1050 c.f_ops      void mouse_enters_runrecord(univ_block *block)
mouse_leaves_runentry   1044 c.f_ops      void mouse_leaves_runentry(univ_block *block)
mouse_leaves_runrecord  1058 c.f_ops      void mouse_leaves_runrecord(univ_block *block)
```

```
mouse_on_boxeditor      1136 c.f_main      static void mouse_on_boxeditor(univ_block *block)
mouse_on_flowcharts      654 c.f_main      void mouse_on_flowcharts(univ_block *block)
mouse_on_help            297 c.f_help      void mouse_on_help(univ_block *block)
mouse_on_panel          1332 c.f_main      static void mouse_on_panel(univ_block *block)
mouse_on_plotwindow     1231 c.f_main      static void mouse_on_plotwindow(univ_block *block)
mouse_on_vartable        542 c.f_ops       void mouse_on_vartable(univ_block *block)
move_current_box         603 c.f_main      static void move_current_box(univ_block *block)
move_mouse_to            581 c.w_shell     void move_mouse_to(int x, int y)
msb                       30 c.w_shell     #define msb(x)  (char)((x>>8)&0xFF)
mse_on_runentry         1192 c.f_ops       void mse_on_runentry(univ_block *block)
mse_on_runrecord        1254 c.f_ops       void mse_on_runrecord(univ_block *block)
mse_pressed_function     341 c.w_shell     void mse_pressed_function(handle,fn)
new_vars_menu            511 c.f_ops       static void new_vars_menu(univ_block *block, int newvar)
next_line                229 c.f_plot      #define next_line(x) x=x-36;\
oficon                  1092 c.f_main      static int oficon(int ihandle)
one_poll                 358 c.w_shell     static poll_return *one_poll(int mask)
open_wimp                156 c.w_shell     void open_wimp(int bg)
open_window_function     246 c.w_shell     void open_window_function(handle,fn)
originlwa                511 c.w_shell     void originlwa(int handle, int *xco, int *yco)
overwrite_box            888 c.f_main      static void overwrite_box(flow_box_type *current, flow_box_type *bo
poll                     441 c.w_shell     void poll(int mask)
prefs_menu               401 c.f_main      static void prefs_menu(int choice)
print_dp                1885 c.f_ops       static char *print_dp(double number, int precision)
print_sf                1898 c.f_ops       static char *print_sf(double number, int accuracy)
punits                   246 c.f_plot      static void punits(int n)
put_caret                780 c.w_shell     void put_caret(int handle, int x, int y)
put_sprite               599 c.w_shell     void put_sprite(char *name, int action, char *sp_area)
r_to_d                   448 c.f_ops       static double r_to_d(double x)
read_dbl                1868 c.f_ops       void read_dbl(char *rawbuffer, double *pnum)
read_directory            98 c.f_help      static error *read_directory(char *pathname, char *replyblock,int *
redefine_characters     1726 c.f_main      static void redefine_characters(void)
redraw_boxeditor         808 c.f_main      static void redraw_boxeditor(redraw_block *block)
redraw_current_flowchart 835 c.f_main      static void redraw_current_flowchart(void)
redraw_flowchart_number  798 c.f_main      void redraw_flowchart_number(int num)
redraw_flowcharts        756 c.f_main      static void redraw_flowcharts(redraw_block *block)
redraw_graphinfo         257 c.f_plot      void redraw_graphinfo(redraw_block *block)
redraw_help              315 c.f_help      void redraw_help(redraw_block *block)
redraw_plotbox          1312 c.f_main      static void redraw_plotbox(redraw_block *block)
redraw_runentry         1505 c.f_ops       void redraw_runentry(redraw_block *block)
redraw_runrecord        1449 c.f_ops       void redraw_runrecord(redraw_block *block)
redraw_vartable          560 c.f_ops       void redraw_vartable(redraw_block *block)
redraw_window            483 c.w_shell     void redraw_window(int handle)
remove_last_from_path     55 c.f_help      static void remove_last_from_path(void)
remove_variable          469 c.f_ops       static void remove_variable(univ_block *block)
rename_menu              215 c.f_main      void rename_menu(univ_block *block)
reset_caret             1008 c.f_main      static void reset_caret(void)
run_flowchart            978 c.f_ops       static void run_flowchart(int flownumber, char *input, char *output
run_time_error           623 c.f_ops       static void run_time_error(char *errmess, int fnum, int boxnum,int
save_flowcharts         1711 c.f_ops       void save_flowcharts(char *fname)
scroll_window            752 c.w_shell     void scroll_window(int handle, int xd, int yd)
set_from_env            1449 c.f_main      static void set_from_env(int *var, int dflt, char *envvar)
set_palette             1437 c.f_main      static void set_palette(char *file)
set_taskname             834 c.w_shell     void set_taskname(char *name)
set_tick_item            140 c.f_main      void set_tick_item(int num, menu_block *menu)
```

```
set_window_extent          736 c.w_shell     void set_window_extent(int handle, int xb, int yl, int xt, int yr)
setup_calc_screen          843 c.f_main      void setup_calc_screen(flow_box_type *box)
setup_dir_menu              66 c.f_help      menu_block *setup_dir_menu(char *header, char *directory)
setup_dir_menu1            119 c.f_help      static menu_block *setup_dir_menu1(char *header, char *directory,in
setup_dir_menu2            192 c.f_help      static menu_block *setup_dir_menu2(char *header, char *directory)
setup_dir_menu_r            78 c.f_help      menu_block *setup_dir_menu_r(char *header, char *directory)
setup_file_menu            861 c.f_setup     static void setup_file_menu()
setup_graph_menu           878 c.f_setup     static void setup_graph_menu()
setup_prefs_menu           921 c.f_setup     static void setup_prefs_menu()
setup_rename_menu           89 c.f_help      menu_block *setup_rename_menu(char *header, char *directory)
setup_super_menu           908 c.f_setup     static void setup_super_menu()
setup_vars_menu            934 c.f_setup     static void setup_vars_menu()
setup_winds                966 c.f_setup     void setup_winds()
store                      598 c.f_ops       void store (double val, char *ident)
stripspaces                192 c.f_ops       void stripspaces(char *buf)
super_plot                  77 c.f_plot      void super_plot(int num, double *x, double *y, int col)
syntax_ok                  406 c.f_ops       int syntax_ok(char *buffer, flow_box_type *box)
ticklength                  28 c.f_plot      #define ticklength(num)  ((num%5==0)?15:8)
tokenise                   220 c.f_ops       static int tokenise(union data *dta, char *buffer, char *out)
undo_plot_record           562 c.f_main      void undo_plot_record(int num)
unset_all_ticks            132 c.f_main      static void unset_all_ticks(menu_block *menu)
unset_tick_item            122 c.f_main      static void unset_tick_item(int num, menu_block *menu)
update_boxrecs             556 c.f_main      void update_boxrecs(void)
update_calc_screen         876 c.f_main      void update_calc_screen(void)
update_current_flowchart  1032 c.f_main      static void update_current_flowchart(flow_box_type *box)
update_fkey               1183 c.f_ops       static void update_fkey(void)
update_input              1164 c.f_ops       static void update_input(void)
update_max                 927 c.f_main      static void update_max(int boxnum)
update_output             1173 c.f_ops       static void update_output(void)
update_record             1122 c.f_ops       static void update_record(char *in, int op, char *out)
usr_close_wind             524 c.w_shell     void usr_close_wind(int handle)
usr_create_menu            990 c.w_shell     void usr_create_menu(block,xp,yp,fn)
usr_create_wind            534 c.w_shell     int usr_create_wind(wind_block *wblock)
usr_open_wind              546 c.w_shell     void usr_open_wind(int handle, int wind_behind)
usr_open_wind_scroll       558 c.w_shell     void usr_open_wind_scroll(int handle, int wind_behind, int sx, int
usr_update_scroll_wind     657 c.w_shell     void usr_update_scroll_wind(int handle, int topleft_x, int topleft_
usr_update_wind            636 c.w_shell     void usr_update_wind(int handle)
var_exists                 460 c.f_ops       static int var_exists(char *varname, int *index)
vars_menu_decode           520 c.f_ops       static void vars_menu_decode(univ_block *block)
varval                     638 c.f_ops       static double varval(flow_box_type *box, int fnum, int *evalerror)
wimp_decision             1700 c.f_ops       int wimp_decision(char *text)
wimp_error                 799 c.w_shell     int wimp_error(error *e, int flags)
wind_enter_function        277 c.w_shell     void wind_enter_function(handle,fn)
wind_leave_function        293 c.w_shell     void wind_leave_function(handle,fn)
window_redraw_function     311 c.w_shell     void window_redraw_function(handle,fn)
wipe_all_charts            593 c.f_main      void wipe_all_charts(void)
wipe_chart                 576 c.f_main      void wipe_chart(int num)
xaxis                      153 c.f_plot      static void xaxis(void)
xco                         22 c.f_plot      #define xco(va)  ((int)((va-x_min)*x_scale))
xval                        25 c.f_plot      #define xval(x)  ((double)(x*x_scale)+x_min)
yaxis                      191 c.f_plot      static void yaxis(void)
yco                         23 c.f_plot      #define yco(va)  ((int)((va-y_min)*y_scale))
yval                        26 c.f_plot      #define yval(y)  ((double)(y*y_scale)+y_min)
```