AN EXPERT WRITING MODEL FOR STORY COMPOSITION

by

MICHAEL ROBERT GARDNER, B.Sc

A Doctoral Thesis
Submitted in partial fulfilment of the requirements
for the award of Doctor of Philosophy of the
Loughborough University of Technology

November 1991

I would like to acknowledge the assistance given by my research director Professor Ernest Edmonds and my supervisor Dr John Connolly throughout the course of the research and in the preparation of this thesis.

Also thanks must go to my wife Maura and to my parents who kept me on the straight and narrow.

An Expert Writing Model for Story Composition

by M. Gardner

ABSTRACT

First the thesis reviews the development of Intelligent Computer Assisted Instruction (ICAI) systems by outlining the different ways that computers have been used in education followed by a description of the functionality of ICAI systems in terms of the Hartley-Sleeman model of classification. This is followed by a discussion of the skills required within writing and their pedagogical context. The different strategies that have been applied to computer supported composition are then discussed with examples of systems where appropriate.

The rationale for a new composition support system is then argued and the criteria for its development described in terms of the functionality of ICAI systems, and the constraints imposed by the requirement for natural language processing. This is followed by the description of an experimental system called MULTISTORY which can assist the writer throughout the writing process in making plot level decisions.

A critique is then made of the MULTISTORY system and the requirement for an Expert Writing Model is identified. An architecture for the Expert Writing Model is then proposed and the components are described in terms of top-down and bottom-up knowledge sources. An example is then used to illustrate the application of the Expert Writing Model to a sample story. Finally the Expert Writing Model is placed within the framework of the Writer's Assistant and further investigations are proposed.

CONTENTS

Chapter 1. INTELLIGENT COMPUTER ASSISTED INSTRUCTION

Interest in Britain in the field of Intelligent Computer Assisted Instruction (ICAI) is reflected by it being identified by the Alvey programme for action on Intelligent Knowledge Based Systems (IKBS) (SERC/DoI, 1983) and the subsequent establishment of an Alvey IKBS Special Interest Group for ICAI (Ford & Yazdani, 1988). This surge of interest was paralleled with the rise of Artificial Intelligence as an exciting and promising vision of the future. However much of the early work raised new questions to be answered rather than solving existing ones; this led to a general disillusionment with A.I and ICAI, especially within the commercial sector. However the field continues to offer new insights into the fundamental problem of knowledge communication and we are beginning to see a fresh impetus within the subject.

What is an ICAI system and how has this field developed over the past 15 to 20 years ? This chapter will try to answer these points by giving a brief history of computers in education, describe the emergence of ICAI systems and the components which identify such systems, outline some key systems and put forward some relevant points of view about this emerging technology. The framework for ICAI systems described in this chapter will be used later on in the thesis as a basis for an evaluation of a writing support system and also to provide a context for the Expert Writing Model.

The term ICAI rather than ITS (Intelligent Tutoring Systems) will be used throughout this thesis, as it encompasses a wider domain than is implied by the 'Tutoring' component of ITS.

1.1. Computers and education

Anderson, Boyle and Reiser (1985) estimated there to be over 10,000 pieces of educational software available. However, almost all of these were classified as 'Computer Assisted Instruction' (CAI) as opposed to 'Intelligent Computer Assisted Instruction' (ICAI). What therefore are the broad categories of educational software? O'Shea and Self (1983) have attempted to classify the different types of educational software and have identified 11 approaches used in CAI each of which reflect the way people have regarded the educational use of computers. The following are a capitulation of these categories:

Linear Programs: these programs attempt to reinforce ideas put forward by the teacher. A simple question/answer dialogue prompts the pupil to answer a question and regardless of the answer given continues on to the next question. The main drawback of this type of system is that each student receives the same material and in the same order regardless of their aptitude. The feedback given by the system is only relevant if an answer is correct; the machine cannot recognise a nearly correct answer.

Branching Programs: this is a similar technique to Linear Programs with the added feature of branching to different sets of questions based on the answer given. Authoring languages with IF and GOTO facilities allow teachers to construct their own branching programs, but the main problem is that as more questions are added there is a build up of a large numbers of rules, making the programs unwieldy and difficult to manage.

Generative Computer-Assisted Learning: mainly used for arithmetic problems, generative systems are able to generate questions of suitable difficulty corresponding to the ability of the student. (see Wexler 1970). By using program variables the generative system can provide as many problems as the student needs but due to the nature of providing such a precise specification it is usually restricted to subjects which can be very well defined.

Mathematical Models of Learning: the aim of this technique is to develop mathematical theories on which to model the cognitive process of learning. This model can then be used to direct a teaching pattern interactively on the computer. Very little is known about the learning process so the validity of such models alone is doubtful. However, cognitive modelling in unison with other techniques has been put to good use by several ICAI systems, which will be discussed further later on in this chapter.

TICCIT: in 1971 the National Science Foundation of America set up an experiment to test the effectiveness of computer-aided learning which became the TICCIT (Time-shared Interactive Computer Controlled Information Television) project. The system included lessons on pure calculus mathematics and English composition using the same underlying method. First describing general principles, then examples to illustrate these principles and finally exercises for the student to complete. Each TICCIT system served up to 128 terminals and it represents one of the first wide spread implementations of computer aided learning. The system met with a mixed response and was not widely adopted.

PLATO: The PLATO (Programmed Logic for Automatic Teaching Operation) system was similar to TICCIT but was implemented on a larger scale (up to 1200 terminals), used better technology (plasma displays were used which allowed extensive use of graphics) and the emphasis was on end users developing their own course software rather than being the sole responsibility of specialists.(see Levy, 1983).

The main drawback of PLATO was that the laborious programming necessary to implement course-ware discouraged many teachers from developing their own course-ware and therefore led to a lack of appropriate software. Also response times were variable and many students only used the system for communicating messages to friends. However, when taking into account the nature of the technology used by PLATO then it can be viewed as a pioneering system for its day.

Simulations: here the computer is used to simulate a process and the student is encouraged to learn by observing the process. This technique particularly requires the careful use of computer graphics which can enhance the simulation. In more advanced systems the student can affect the outcome of the simulation by altering the values of certain parameters. The role of this type of education is limited to applications which can be easily simulated.

Games: here the aim is to combine the fun aspects of computer games with a learning/educational component. For example, the game 'How the West was won' was developed by Anderson in 1977 for the PLATO system. This involved combining numbers in order to move a player around the board. (A COACH for giving advice was later added to the system, which was renamed WEST). Complex adventure games such as 'Granny's Garden' (Granny's Garden,1984) have had much praise for combining an element of learning within an enjoyable game environment.

Problem-Solving: the concept underlying this approach is that the student will learn from trying to solve problems. The computer can simply be used as a tool to enrich this process. The LOGO programming environment (Papert, 1980) was developed with this aim. Here the student is able to move a 'turtle' around either graphically on a computer screen or using a physical robot attached to the computer. Papert has stressed that children 'learn by doing' and that LOGO gives children the power to experiment with mathematics. Other tools such as Micro-Prolog (Ennals,1984) have also been used as part of a learning environment . In Micro-Prolog the student can experiment with logic and in the process learn to solve quite complex problems.

Emancipatory Modes: computer software can be used to relieve the student of tedious tasks (eg. spelling correction) and provide additional support for such tasks as word-processing, calculating and storage of information. If these tools are properly used then they can free the student to concentrate on the real tasks at hand.

Dialogue systems: these systems attempt to mimic the relationship between student and tutor. In a computer based tutoring environment the student should be able to influence the aspects of a course which are to be emphasised by the computer system. The tutoring component may also monitor the progress of the student and modify the teaching to suit the perceived student needs. For Dialogue systems to be truly effective there must be a rich two-way communication between tutor and student. Unfortunately computers can usually support only a highly restricted subset of natural language communication and therefore the power of dialogue systems is highly restricted by the interface bottle-neck.

From this cursory description of so-called 'conventional' educational software the next two sections describe the paradigm of Intelligent Computer Assisted Instruction (ICAI) systems, with a detailed description of their components.

1.2. ICAI systems

ICAI systems can be said to have developed from the fields of Generative CAI and Dialogue Systems (which were described in the previous section) and the application of Artificial Intelligence techniques.

One of the original goals for ICAI was to:

> *'extend the domain of applicability, the power and the accuracy of adaptive systems.'*
> *(Sleeman and Brown, 1982)*

Research focused on the design of systems that could:

> *'offer instruction in a manner that is sensitive to the students strengths, weaknesses and preferred style of learning. The role of AI in computer-based instructional applications is seen as making possible a new kind of learning environment.' (Barr and Clancey, 1982)*

ICAI can be said to be the application of Artificial Intelligence techniques to the design of such tutoring systems. However it shares many of the same goals of Dialogue and Generative CAI systems.

The following diagram (figure 1) attempts to illustrate the range of computer based learning systems. ICAI falls into the middle of this range in that it attempts to combine both directed learning and open-ended problem solving techniques (VanLehn and Soloway, 1985).

Figure 1. The range of computer based learning systems.

DIRECTED LEARNING
eg. Linear/branching programs
question/answer CAI
system in control
student cannot explore

OPEN-ENDED
PROBLEM SOLVING
eg. LOGO
student in control
inefficient use of time

ICAI
controlled by student and system
allows exploration
gives hints,demonstrations and explanations
answers questions and suggests challenges

1.3. Components of an ICAI system

What are the components of an ICAI system? Surprisingly there has been quite a lot of agreement within the research community over the distinction between the different components, the main disagreements seem to be over their relative importance. In most cases an ICAI system can usefully be classified in terms of the Hartley-Sleeman model (Yazdani, 1986) as this model is general enough to fit most kinds of ICAI systems. This model consists of a 'User Interface', an 'Expert Model', a 'Student Model', and a 'Tutoring Strategy'. Some researchers include an additional distinct component called the 'Psychologist' (Brecht & Jones, 1988) which is responsible for maintaining the student model by determining when a skill has been mastered, what errors arise in skills, what skills are being learned, and what is to be taught next. However in this chapter these functions are addressed within the other components and the 'Psychologist' component can be viewed as the glue which binds the separate ICAI components together.

1.3.1. User interface

Generally it can be said that the easier it is to use a system the easier it will be for the user to accomplish the task at hand. This also follows for ICAI systems. The problem solving process should be separated as much as possible from the process of operating the system. As Brown(1977) said:

> 'As computers become more powerful, the student-machine interface will quickly become a bottle-neck in effective learning.'

Various techniques can be used to accomplish this aim such as simple menu operated systems and graphical output through to complex natural language dialogue systems.

For example, the development of the WUMPUS computer game into the WUSOR-I (Goldstein 1982) and WUSOR-II (Goldstein and Carr 1977) ICAI systems illustrates this concept. The WUMPUS game involves the player initially being placed somewhere in a randomly connected warren of caves and information is given to the player about adjoining caves. The players goal is to find the Wumpus monster and slay it. To be a skilled Wumpus hunter one must be able to apply a combination of logical skills, probability, decision theory and geometry.

WUSOR-I implemented this game with an additional computer coach using a simple teletype style interface. For example (Barr and Clancey,1982):

Hello, Timmy. There are 15 caves, 2 bats, 2 pits,
and 1 Wumpus. You are now at cave 15 with neighbours
4, 14 and 0. Brrr! There is a draft. You are near a
pit. What a stench! The Wumpus is near. What now?

**4

You are now at cave 4 with neighbours 15, 14 and 2.
Brrr! Squeak! A bat is near. What now?

**14

etc.

The main problem with this technique was that students used scrap paper to map out the caves and to draw tables ie. the problem solving process was not separated from the process of using the system. Although, it could be said that this approach forced the student to maintain a mental model of the WUMPUS world.

WUSOR-II solved this problem by using a graphical interface which drew a map of the caves thus allowing the student to concentrate on mastering the logic and probability.

Although in a different educational domain, this concept was also utilised by the Anderson-Boyle Geometry tutor (Anderson, Boyle and Yost,1985). Here a graphical interface was used to off-load the mechanics of applying geometry theories. This type of interface explicates theorem proving and so concentrated the student on the real problem to be solved.

WUMPUS and WUSOR also allowed the pupil to enter directions in the form of normal English sentences. Natural language communication between pupil and computer has been implemented in several other ICAI systems; the most notable being SCHOLAR (Carbonnell 1970), SOPHIE (Brown and Burton 1975) and WHY (Stevens and Collins 1977).

SCHOLAR was primarily developed to tutor students on South American Geography although the system could be used for other domains (eg. NLS-SCHOLAR was developed to tutor people on the use of complex text-editing programs (Grignetti, Hausmann and Gould, 1975)). SHOLAR allowed students to enter their queries in a so-called 'comfortable' subset of English.

A mixed-initiative dialogue system was used where both the system and the student could initiate conversation by asking questions. The following is an excerpt from a session with the South American Geography tutor (Barr and Clancey 1982). The students questions and answers are preceded by ** and annotations are shown in italics:

What is the language of Argentina?

**Probably Spanish

Very good.
Approximately what is the area of Argentina?

Student changes topic...

**Tell me something about Peru

Peru is a country.
It is located in South America.
The capital of Peru is Lima.
Now answer the question you didn't answer before.

...finally answers the question.

**The area of Argentina is 1,100,000 square miles

Approximately true, but a more exact value is
1,200,000.
The eastern bordering country of Peru is ---.

etc.

The natural-language processing technique used by SCHOLAR was based on case grammars (Fillmore,1968) with the knowledge representation implemented as semantic nets (Quillian 1968, Raphael 1968).

Theoretically a Natural Language interface promises great wealth. However much caution is needed before embarking on such a venture. At a simplistic level the user may find typing in natural language time consuming and there are many other associated problems such as typing errors, bad spelling, and difficulty in expression. However, the major drawback of such systems is the effort needed to build even a system capable of understanding a restricted subset of English. For example, SOPHIE handles 90% of all student queries but cost 2 man-years over 4 years to build. On the other hand, SOPHIE with a menu interface cost less than one man-month and had excellent performance; but only certain kinds of queries were allowed. The claimed performance figures of such Natural Language systems needs to be carefully scrutinised. In most cases the systems do not support natural language at all, but rather a very restricted subset which bears more resemblance to a structured query language than English text. Also it is not simply

the case that one can build one natural language engine and apply it to any ICAI system. Each ICAI domain will have its own set of semantic and pragmatic criteria which will have to be built in to the natural language engine before it can be properly used.

The WHY system (Stevens A.L, and Collins A, 1977) used a socratic tutoring method (described later) in the domain of meteorology. Here a natural language interface was essential because menus could not list all possible student answers and would anyway reveal the correct answer. A semantic grammar (Burton, 1976) was used to build the language comprehension module, yet the system when finished could only deal with a limited set of natural inputs and failed to understand many sentences typed by the student.

Where a natural language interface is used, often the interface component is implicitly embedded within the overall knowledge representation. Bumbaca (1988) describes a system where the natural language interface uses a conceptual dependency parser (Schank & Rieger, 1974), and the expert and student model's both use conceptual dependency as their root knowledge representation language. The proposed inherent advantages are a firm knowledge representation language framework, and a better intermix of knowledge experts (the system is implemented using a blackboard structure). In this case there might well be a trade-off between developing the natural language engine and the other components of the ICAI systems. However, the previous provisos still apply.

Alternatively, Wilson (1986) advocates the use of a detached interface module similar to the SYNICS type 'interface processor' developed by Edmonds (1982). Some of the advantages of such a system are that the user interface can provide separate interfaces for specific hardware, and there is increased flexibility in handling several user ability/experience profiles. In effect with this approach it may be possible to off-load part of the user model to the interface pre-processor.

In conclusion the user interface should off-load extraneous factors allowing the student to concentrate on the problem at hand.

However, this may involve changing the nature of the task itself. The approach taken (such as the use of a menu driven or a natural language system) will depend on the nature of the task and the facilities available. A Natural Language interface may be essential and/or impossible to implement effectively. Menus can be effective but can also be too restrictive. Alderson and DeWolf, 1985 provide user interface guide-lines for Computer Aided Learning (CAL) applications, many of which can be applied to ICAI systems. Current PC and Unix based windowing environments such as Microsoft Windows (™ Microsoft), and Open Software Foundation Motif allow the user (to a limited extent) to alter the screen to their own preferences, and they provide strict user interface guide-lines for application developers.

The degree to which the user interface is embedded within the rest of the system is also important, and can be viewed as a trade-off between the degree of integration with the other knowledge structures, and the ease by which the user interface can be modified separately. Above all else, the system should be easy to learn and to use.

1.3.2. Representing domain knowledge

Many of the issues involved in representing and applying the expertise or domain knowledge in an ICAI are covered by what has been termed 'expert systems' (see Colbourn, 1984, for an overview of expert systems in education). The following definition of expert systems is provided by Hayes-Roth F, Waterman T, Lenat D B, 1983:

> *'An expert system embodies knowledge of a particular application area combined with inference mechanisms which enable the program to employ this knowledge in problem-solving situations.'*

Typically an expert system will include an inference engine that will load and run various knowledge-bases which encapsulate different areas of expertise. Many of the 'first generation' expert systems were

stand-alone systems. However, the main feature of the new range of advanced so-called 'second-generation' expert systems such as Nexpert Object (™ Neuron Data) is the ability to embed the expert system within other conventional programming languages and systems. This feature has always been a major requirement for ICAI systems where the domain knowledge is one component of the overall system.

Typically, the knowledge-base will include both the 'content' to be taught and the mechanisms of how to use that knowledge to solve related problems. This is a classic A.I problem and involves the necessity to make explicit the deep structure knowledge of the problem domain (a very hard problem). For example, to code the knowledge of an 'expert' computer programmer, a good source of information would be found in computer textbooks particularly information on the syntax and semantics of the programming language to be modelled. However, often the difference between an 'expert' programmer and a novice is the extra knowledge that the expert has gained through experience. This would enable the 'expert' to make informed guesses and to use inbuilt programming 'plans' when solving a problem. Unfortunately, this knowledge is often not available in textbooks and can only be obtained through a process of knowledge engineering. Also, this deeper knowledge can often be difficult to represent and manage in a computer system (Rich, 1983, provides a good overview of A.I representation techniques).

The development of the MYCIN (Shortliffe, 1976) expert system into the tutoring system GUIDON (Clancey, 1979) illustrates many of these problems.

MYCIN was a consultation system for diagnosing infectious diseases. The body of knowledge base was represented as a collection of conditional sentences or 'production rules'. The MYCIN knowledge base contained about 450 such rules each of which consisted of a set of preconditions which, if true justified the conclusion made in the 'action' part of the rule. For example (Clancey, 1979):

'IF (1) the gram stain of the organism is gram negative, and (2) the morphology of the organism is rod, and (3) the aerobicity of the organism is anaerobic, THEN there is suggestive evidence (0.6) that the genus of the organism is Bacteroides.'

These rules were built up over a period of 4 years through a series of consultations with physicians.

The GUIDON system was designed to explore two basic questions: First, how would the problem-solving rules, which performed so well in the MYCIN consultation system, measure up to the needs of a tutorial interaction with a student? Second, what knowledge about teaching would need to be added to MYCIN to make it into an effective tutorial program?

Here is an example session with GUIDON (Van Lehn and Soloway 1985):

Initial factors:
1.Patient age: 59
2.Hospitalized
3.Severely burned
4.X-ray of head: normal
5.White blood count from cerebrospinal fluid = 2500

Guidon: What is the type of the infection?

Student:Bacterial

Guidon: What facts about this case tell you that the type of infection is bacterial? (please enter one factor per line)

Student:Burned
Student:Lumbar puncture
Student:WBC in CSF

Before a session with the student begins, GUIDON asked MYCIN to 'solve' the case to be presented to the student. This information was then used to guide the tutoring session.

Clancey evaluated GUIDON in informal tests with medical students. The main results were that the students found the rules difficult to understand, remember and incorporate into a coherent problem-solving process. GUIDON could not tell students the strategy it had pursued or tell students why a rule was correct from a strategic point of view.

MYCIN's knowledge base was meant only to solve problems (ie. diagnose infections) which meant there was a quick jump from problem to solution. In GUIDON the aim was to teach students the reasoning process that produced MYCIN's compiled knowledge. Because important structural and strategic knowledge was implicit in the rules this knowledge was not available for teaching purposes. To make this implicit 'design knowledge' explicit, a new system, NEOMYCIN (Clancey and Letsinger, 1981) was developed that separated out the diagnostic strategies from the domain knowledge and used a more hierarchical organisation of data and hypotheses. (See Clancey, 1987, for an overview of the GUIDON program).

MYCIN used production rules to represent the domain knowledge. Other methods include 'semantic nets' (Quillian, 1968), 'conceptual dependency' (Schank and Rieger, 1974), 'frames' (Minsky, 1975) and 'scripts' (Schank and Abelson,1977) which are documented elsewhere, and are outside of the scope of this chapter.

The MYCIN/GUIDON system illustrated that it was not possible to simply take an existing expert system and make it into an ICAI system. The underlying reasoning and strategic knowledge must be explicitly represented.

Other ICAI systems have employed different solutions to this problem. For example, the WEST system (Burton and Brown 1982) was a coaching system for the computer board game 'How the West was won'. The object of this game was to be first to traverse the board

by throwing a dice. Along the way an advantage could be gained by landing on a town or on a shortcut and a player could be 'bumped' back by an opponent. The skill of the game was in combining the dice scores to either 'bump' an opponent or to get to a town or shortcut. This involved the player having an ability in basic arithmetic and being able to decide on an appropriate strategy. The knowledge structure used in WEST to diagnose how well a pupil was playing the game was based on 'feature vectors'. Here the skills needed to play the game were broken down into separate alternative strategies such as bumping an opponent, reaching a town, getting the largest number from the available dice thrown, and so on. By analysing the pupils moves it was then possible to decide on which strategies were not being used. This method worked well in the WEST system but tended to be too course grained in other domains where the students' misconceptions could be due to different factors. For example, in diagnosing a students ability at arithmetic as illustrated by the following problem (Van Lehn and Soloway 1985):

756	726
- 129	159 -
---	---
627 CORRECT	647 INCORRECT

Using feature vectors an analyser could diagnose that the probability of using 'borrow' correctly in the above example is 0.5. The real problem could be that the student does not know how to borrow in a column already borrowed from. The actual student misconception is not being recognised at all.

The BUGGY system (Brown and Burton 1978) attempted to provide a more fine grained description of skills necessary for subtraction in the form of procedural nets (Sacerdoti 1977). A skill lattice is constructed for subtraction in which there are correct methods for achieving the goal of subtraction and incorrect or 'buggy' methods. This level of detail led to the identification of 58 sub-skills necessary for subtraction with 110 primitive bugs and 20 common compound bugs. This network is then compared with the students answer to

diagnose which bugs if any, are present and to explain the reason for the student's answer. The system was tested with over a thousand students and was used extensively in classroom situations. BUGGY was very successful in diagnosing correctly the reasons for students mistakes but only worked in the very small domain of subtraction. It was never meant as a cognitive model but a simple framework for relevant pieces of information.

Repair theory (VanLehn, 1983) is an attempt to build a cognitive model for the process of subtraction.

```
a) 756          726   (b)
  -129          159 -
   ---          ---
   627 CORRECT  647 INCORRECT
```

```
c) 726
   150 -
   ---
   617 INCORRECT
```

In the previous example the student's problem is that he does not know how to borrow from a column already borrowed from. The student overcomes this impasse' by a repair which in (a) means taking the difference between the two numbers, while in (b) means writing down the top number in the 10's column. A procedural network would diagnose two bugs where the student really only has one conceptual bug but uses two different repairs. What is needed is a conceptual model alongside the procedural network to recognise bugs caused by the same conceptual error.

Repair theory has also been applied to the problem of machine learning. The SIERRA system (VanLehn, 1987) is a study of the acquisition of mathematical skills. SIERRA's input is an ordered sequence of lessons, where a lesson is an unordered set of examples, and each lesson builds on the procedure learnt in the previous lesson.

This section has given an brief overview of some of the issues involved in representing the domain knowledge in an ICAI system, including specific examples where appropriate. Many of the problems are shared with the expert systems field. In summary, it should be possible to embed the domain knowledge within other software modules, and the knowledge must represent the underlying reasoning mechanisms in order that they can be applyed to specific student problems in which various examples have been given. ICAI systems should benefit from the advances in expert systems technology and particularly the integration of different reasoning paradigms such as machine learning and neural nets. The domain knowledge component can be viewed as the 'analysis' phase within and ICAI session. This will be expanded later when we discuss the domain knowledge in an Expert Writing Model.

1.3.3.The embedded student model

Self (1985), generalises existing ICAI systems into two groups; small scale 'paradigmatic' programs and large-scale 'expert system' based programs. The former include such systems as BUGGY (Brown and Burton 1978), WEST (Burton and Brown 1982) and WUMPUS (Goldstein 1982). In the words of Self (1985) they *'concentrate on small domains and attempt to establish paradigms for the implementation of larger scale, realistic systems'*. Self argues that these systems have never been developed into complete tutorial systems and implies that this is because of inappropriate design strategies. The 'expert system' class includes SOPHIE (Brown, 1977) and GUIDON (Clancey 1979). Self argues that the *'existence of an expert system to solve problems...does not necessarily mean that the domain is an appropriate one for Intelligent Computer Assisted Learning'*.

Self has argued that too much emphasis is placed upon representing domain knowledge where the model of the student's knowledge is a subset of the internal idealised knowledge base and learners are viewed only as errorful experts. Instead Self argues, *'the central*

component of ICAI systems should be the student model' and he describes this as a 'Learner centred' approach (Self, 1985).

This is a clear indictment of past approaches at student modelling and it gives an indication on the controversy of this subject. It will be seen that the student model is a major component of an ICAI system. This section will describe some of the techniques used for student modelling with example systems. This is followed by a summary of the main arguments concerning the validity of the student model.

1.3.3.1.Overlays

An Overlay student model is regarded as a subset of a larger expert model or knowledge base (the domain model). It requires that the method of tutoring should discover which rules are missing from the student's model, but which are present in the expert model.

To illustrate this method the following example based on a production-rule representation is given (Van Lehn and Soloway 1985):

For each rule in the knowledge base there are two counters; a 'used-counter' and a 'missed-opportunity-counter'.

Given the student's move, analyse it into a line of reasoning whose steps are applications of rules (using the domain knowledge).

For each rule used by the student increment its used-counter in the overlay.

For each better move (from the expert model) analyse it into a line of reasoning and increment the rule's missed-opportunity-counter in the overlay.

It can then be said that a student knows a rule if the used-counter is greater than the missed-opportunity-counter and does not know a

rule if the used-counter is zero and the missed-opportunity-counter is greater than zero.

A problem with this technique is the apportionment of blame to a rule when a particular line of reasoning requires more than one rule. Which rule(s) is/are to blame. WEST (Burton and Brown, 1982) overcame this problem by only tutoring the student on a particular rule when the missed-opportunity-counter was significantly larger than the used-counter. But this can be problematic.

Another problem occurs if the student's strategy is different to the expert's. How can we identify which strategy the student is using? WEST overcame this problem by mapping the student's moves with several different sets of strategies. The current strategy would be the strategy with the lowest missed-opportunity-counter.

Also, how can this method decide which rules are being used if the student's move can be generated by more than one line of reasoning (ie. an apportionment of credit problem). GUIDON (Clancey, 1979), which tutored students on diagnosing infections overcame this problem by asking the student to explain which line of reasoning was being used whenever it was not obvious.

The overlay technique provides a very coarse-grained method for creating a student model. The main problem with it is that a student can only ever be compared to a rigid expert model. The strength of the student model will depend entirely on the completeness of the expert model. It is very difficult to make the expert model sufficiently flexible enough so that it can recognise more than one strategy for a given problem. However, the overlay technique does provide an inexpensive method for constructing student models.

1.3.3.2.Bug collections

With a Bug Collection model the set of expert rules used in overlays are augmented by a set of production rules called mal-rules which represent possible bugs or errors made by a student. The student model is then a combination of correct rules which represent skills

used properly and mal-rules which represent sub-skills not learnt yet. This provides a more accurate student model with the ability to pin-point specific student problems.

This type of student model has been represented as a production rule system (as in LMS, Sleeman and Smith, 1981) and as procedural nets (BUGGY, Brown and Burton, 1978). But regardless of the representation the main problem is in determining which bugs the student has. One solution is to generate all possible student models and a predicted test solution for each of these models. The most accurate student model will be the one whose solution most closely resembles the student's answer. The permutations of student models becomes very large and unmanageable as more bugs are represented. Often there will be more than one bug in the student's behaviour which means combinations of buggy models have to be computed to form a diagnostic model. BUGGY (Brown and Burton 1978) represents the expert and student bugs as procedural nets in the form of LISP functions. This makes it much faster than production systems, but it still requires a large amount of compute time and is amenable only to domains where bugs can be explicated in a more or less complete way.

The LMS Leeds Modelling System (Sleeman and Smith, 1981) is an attempt to cut down on the number of buggy models generated in the student model. The key assumption is that parts of the student's skill that function correctly in simple situations will not 'break down' in more complicated ones. The set of expert rules are partitioned into 'skill' levels each of which is sufficient to solve a certain class of problems. The mal-rules are then grouped with their appropriate group of expert rules. The student is given a problem from the lowest skill level and a student model is computed by generate and test. Moving up the hierarchy at each level a separate student model is computed which is used to augment the previous level. LMS was initially developed to create student models for arithmetic problems. This work was taken a stage further with the PIXIE system (Moore & Sleeman, 1988). PIXIE is a shell for creating ICAI systems that attempt to diagnose and remediate student errors in a particular domain, and it has been used for tutoring algebra. PIXIE has three

phases: (1) off-line phase - generate malrule models (2) Online phase - tutoring using these models (3) Analysis phase - diagnosis of performance and modification of malrule models.

Although this technique provides a more fine grained method for modelling students problems, particularly for identifying skills not learnt yet, it does suffer from becoming very large very quickly. This does restrict it to small domains (such as a subset of arithmetic) and also to domains which can easily be modelled in terms of skills learnt and not learnt (such as arithmetic). In domains that involve a combination of sub-skills the bug-collection technique becomes more difficult to apply. It is often necessary to specifically pin-point the target group of students who will use the ICAI system. The DEBUGGY system (Burton, 1982) found that 49% of 3rd year students were buggy as compared with 13% of 5th year students. With the marked changes in school pupils abilities over a relatively short period of time an ICAI system may only be relevant to a small subset of the student population.

1.3.3.3. Bug construction

The technique used in bug construction is to construct buggy student models without a bug database. The difference between a bug collection system like DEBUGGY (Burton, 1982) and a bug construction system like ACM (Langley and Ohlsson, 1984) is that DEBUGGY makes considerable use of a 'bug library' containing errors that student's are likely to make, while ACM constructs explanations of errorful behaviour from the same components used to model correct behaviour. ACM (like DEBUGGY) was able to diagnose student's subtraction errors, but ACM used a cognitive modelling technique to model the processes required in subtraction (see Anderson,Boyle and Reiser 1985 for a description of the ACT theory of cognition). In order to model subtraction behaviour ACM defined the problem space for subtraction as a set of production rules that would combine to produce the correct subtraction strategy (see example rule for 'find-difference' below).

find-difference
If you are processing column-1,
and number-1 is in column-1 and row-1,
and number-2 is in column-1 and row-2,
[and row-1 is above row-2],
[and number-1 is greater than number-2],
then find the difference between number-1 and number-2,
and write this difference as the result for column-1.

These rules would be combined to form a search tree to find the correct answer to a problem. A student's incorrect answer would also be represented as a search tree and would be compared with the correct answer search tree to find out which rule(s) were being incorrectly used.

The ACT theory has been applied in a Lisp and Geometry tutor (Anderson et al, 1985) and more recently by Walsh (1988) as an ICAI predicate logic teaching tool. The Lisp tutor is commercially available from Advanced Computer Tutoring Inc (ACT) in Pittsburgh.

The advantage of bug construction is that no bug collecting is required because it is done automatically from the cognitive model. The main problem is one of defining the problem space and the combinatorial problems of computing a complete cognitive model from it. For example ACM which runs on a VAX 750 takes some 2 CPU hours to generate a complete cognitive model for a set of 20 subtraction problems.

The bug-construction method appears to offer some advantages over the previous methods, namely a single model without complicated bug collections. However, it is limited to amenable domains which can be modelled to the level of detail required by bug-construction. Basically, this is a technique which combines the expert and bug-collection methods. The expert model is made sufficiently detailed so that sub-skills can be explicitly represented. ACT term this a cognitive model which is somewhat grandiose. It is combined with

sophisticated generate-and-test tree searching and pattern-matching to identify student problems. This technique could provide a good framework for student/expert models but it is very computationally intensive and is limited to amenable domains.

The notion of generating a student model and matching it against an expert model will be continued later in the discussion of the Expert Writing Model.

1.3.3.4. Student modelling - where next?

The above approaches to student modelling to a lesser or greater extent treat the student model as a subset of the expert model. Self (1985) outlines several objections to this approach:

> *'the existence of an expert system to solve problems in a certain domain does not necessarily mean that that domain is an appropriate one for Intelligent Computer Assisted Learning ... the emphasis of these systems is on expertise and only as an afterthought on what learners actually do and know ... the production rule framework is unsuitable for ICAI'*

Self argues that the central component of an ICAI system should be the student model rather than the expert model as in previous systems. The most promising insights for the design of ICAI systems being research on machine learning rather than expert system technology. Self fore-sees a tutoring system which learns a subject at the same pace as the student, supporting the student through collaboration. VanLehn's work on repair theory and its application to machine learning may provide us with a greater insight into this aspect of ICAI systems.

Self (1988) continues to point out some of the weaknesses of current user/student modelling techniques and applications.

> *'Modelling users of ICAI systems in terms of what they know is both epistemologically unsound and educationally*

undesirable ... instead we have proposed that the user model be interpreted as describing what a user 'believes"

Self proposes that a link be forged between work being done on so-called 'belief structures' and how they can be defined and represented computationally (in a student model).

This section has given an overview of student modelling and has described some of the techniques used. The student model can generally be viewed as a subset of the expert model, the difference between approaches reflect the ways in which the student model is generated.

Machine learning and neural net techniques are now coming to fruition with several commercially available products on the market (eg. ID3, KATE) and they are also being integrated with traditional expert systems technology. This flexibility of these new techniques could point the way to the future and the possibility of self-learning tutoring systems.

1.3.4.Tutoring strategies

A domain expert is not necessarily a teaching expert. As with human teachers, a teacher who is an expert in their subject may not be good at communicating that subject to an audience. The important issues that an ICAI system must address are when to interrupt the student, what to say, and what to do next. This section discusses a range of tutoring strategies including coaching, consultants, and socratic/mixed-initiative tutoring. This is followed by a discussion of future issues.

1.3.4.1.Coaching

The coaching approach to tutoring can be applied where the student is set a task and the ICAI system must be active in the background monitoring the students actions. The ICAI system must decide when to interrupt and give advice, and choose what to say. For

example, the WEST system (Burton and Brown, 1982) is a good example of a coaching ICAI system. It allows a student to traverse a board, the aim being to reach the end before their opponents by combining dice scores in the most effective way. Different strategies can be employed such as 'bumping' opponents or by moving as quickly as possible. The coach can give advice to the student if beneficial arithmetic strategies are not being used. This is done by using 'issue recognizers' which will look at 'missed-opportunity' and 'used' counters in the student model (see previous section).

The overall philosophy of WEST and any coaching system is that if the system is going to break in and give advice then the information should be relevant and memorable. More specifically in WEST if the system does break in then the better move suggested should be 'significantly' better and also the system should only break in if it is completely sure that there is a fault in the student's strategy. This means that early on in the game the system cannot tell if there are any bugs so it has to wait until it has more information. WEST is fairly conservative in giving advice in that it waits until the 'missed-opportunity-counter' is significantly bigger than the 'used-counter' before deciding that a bug is present.

As WEST has shown, just because a system does not intervene often, it does not mean that it is a trivial system. (See Goldstein and Carr, 1977, for a good overview of computer coaching). However, it does mean that in this type of environment advice can not be given until the system has built up enough information on which to base a suggestion.

1.3.4.2. Consultants

The consultant approach to tutoring involves a tutoring component sitting between the student and the domain whereas in the coaching approach the student and domain interact freely and the tutor decides when to intervene.

Consultant tutoring has been used most effectively in the domain of teaching computer programming, as in MENO-II (Soloway et al

1981), PROUST (Johnson and Soloway, 1984) and the LISP tutor (Reiser,Anderson and Farrell, 1985).

The LISP tutor was designed to tutor students taking introductory LISP programming and was actively being used as part of a university teaching program (Carnegie-Mellon University, USA). The system would present programming problems and would interrupt the student when there was a deviation from the ideal model for the correct answer. There is immediate corrective feedback when a mistake occurs with the benefit that each problem is dealt with as it occurs. The main danger of this type of system is that students may be trying different ideas out or they may be taking a different path to a correct solution than that dictated by the ICAI system.

There is a fixed lesson sequence where each lesson introduces new concepts and builds on previously learned ones. The consultant approach allows for a far more complex tutoring strategy. Depending on the answers given by the student, the tutor may modify the lesson plan to better suit the students needs.

1.3.4.3. Socratic and mixed-initiative tutoring

The socratic style of tutoring with mixed-initiative dialogues was used in two of the early ICAI systems SCHOLAR (Carbonnell 1970) and WHY (Stevens and Collins 1977, which extended the Socratic aspects of SCHOLAR).

Mixed-initiative dialogue allows for two way communication between the computer tutor and the student (see the section on the computer-human interface for an example dialogue with SCHOLAR). Both the student and the tutor can ask and answer questions. An important aspect in the implementation of this kind of dialogue is the Socratic style of tutoring, where the tutor first attempts to diagnose the student's misconceptions and then presents material that will force the student to see their own errors. An example Socratic heuristic used in the WHY system to control student/system interaction is as follows (taken from Barr and Clancey 1982):

IF the student gives as an explanation of causal dependence one or more factors that are not necessary
THEN select a counter example with the wrong value of the factor and ask the student why his causal dependence does not hold in that case.

Building systems which encompass mixed-initiative dialogue and a socratic tutoring strategy requires a robust natural language understanding component. Few existing systems can support the level of language comprehension necessary to understand all of the student's responses. A robust goal structure is needed for good Socratic dialogues. It may be very difficult to diagnose a students misconceptions unless it is possible to characterise the goals and plans being applied by the student. The goals and rules in WHY only provide an initial characterisation. Unless the socratic tutoring is carefully controlled there is a danger that the student will miss the point altogether.

1.3.4.4. Curriculums

Teachers use a curriculum to teach a subject, with good reason. An ICAI system has to have an explicit curriculum with easier tasks at the beginning to increase student confidence and motivation. Some skills are too complex to present all at once. The problem is in deciding on how a subject should be decomposed into a curriculum.

Step theory (Van Lehn 1983) is an attempt to formalise information transfer. The following felicity conditions have been discovered:

1.Students expect a lesson to introduce at most one new 'chunk' of procedure (called sub-procedures).
2.Students expect the lesson to augment their procedure rather than making parts of it obsolete.
3.Student's induce their new sub-procedure from examples and exercises.
4.Student's expect the lesson to 'show all the work'

of the target sub-procedure. ie. all intermediate work is explicitly stated even though once the sub-procedure is learnt this can be left out.

These conditions may seem like common sense to an experienced teacher, yet they are crucial to the success of an ICAI system. Lessons should be structured into coherent chunks and the students progress through a curriculum carefully monitored.

An alternative approach has been taken by O'Shea's self-improving quadratic tutor (1982). Here the teaching strategy is expressed as a series of production rules. The proposed cycle of operations for the system is to select an educational objective, make an experimental change to the teaching strategy by altering the production rules, statistically evaluate the resulting performance and update both the production rules and set of assertions. A system has been developed to teach techniques for solving quadratic equations. This is still a relatively undeveloped area and may not be easily applied to a larger subject domain.

This section has described several different tutoring strategies, each of which have their own advantages and disadvantages. The choice of which is the most applicable is largely dependent on the type of ICAI system that is being developed. Ford (1987) argues that not enough attention has been given to investigating different teaching strategies and strategies should be based on teaching models. Step Theory is one step along this road, but more research is needed in this area.

Chapter 2. COMPOSITION SUPPORT SYSTEMS

What are composition support systems and how can they be used within an educational environment?

This chapter contains a brief discussion of the skills involved within writing, and their pedagogical context. The use of a word-processor for composition is proposed as being a useful teaching tool, provided that it is used as part of a coherent educational strategy. Other applications which have been used to support writing are presented, including 'non-educational' text processing systems. Finally some observations are made about composition support systems and their continued use as educational tools.

2.1. Pedagogical context

Written language has always played a dominant role in the school system. Typically, the acquisition of literacy is considered to be one of the most important tasks of the school, for it forms the foundation for all subsequent learning and writing is an essential component of overall literacy development.

Writing has several functions, among them the acquisition of knowledge (writing to learn), the development of logical thinking, and the communication of thought and ideas. In many school systems these objectives are restricted. However conventional methods of teaching writing, which have emphasised grammar drills, formal presentations by the teacher, and detailed editing of student products, are giving way to approaches that emphasize significant interactions between reading, writing, speaking and listening; a focus on meaningful communications with real audiences; and ample opportunities for feedback and revision.

What makes a good writer? Bruce (1986):

> "In simplest terms, good writers (in a given domain) 'know more' -- more words, more ways of expressing ideas, more forms of text organisation; they are also more skilled at applying the knowledge they have; and, they have better strategies for putting everything together, even going beyond what they know"

Bereiter and Scardamalia (1982) have investigated childrens' writing development and have identified several important transitions. One such transition occurs between the ages of 9 to 11 when the child becomes aware of their own thoughts and language and begins to take command of their writing.

Good writing depends on a rich environment that teaches knowledge of the world, and skills to represent and manipulate that world symbolically. Development of expertise in writing is enhanced through social interactions at home and at school.

Sharples (1985) attempts to derive a model of the writing process. He identifies the following factors.

text structure a piece of text has embedded layers of structure. The writer will choose between possible sentence constructions. Sharples calls an un-instantiated text structure a *plan* eg. the choice between describing an action with either a passive or active sentence; and any instantiation of a *plan* a *draft*. eg. the final choice for the sentence.

constraints The act of writing is best described as the act of juggling a number of simultaneous constraints. This is in contrast to seeing it as a series of steps that add up to a finished product (Flower and Hayes, 1979).

Text Production there are three fundamental procedures of writing. (a) 'generate and select' -- produce one or more alternative text forms to express a concept, compare the text forms by their ability to satisfy the current constraints, and select one or more suitable forms. (b) 'verify' -- match text already created against the current constraints. (c) 'transform and select' -- identify mismatches between constraints and text.

The teaching of writing should first help a child to become aware of the language that they use and of the process of writing itself. The child can then be helped to build a repertoire of techniques to extend their writing abilities. Ideally a substantial part of the school day should be devoted to writing. Progress depends upon opportunities to write and frequent practise. Individualised instruction is needed, in order to pay attention to the specific needs and competencies of individual students.

Unfortunately, teaching load and teaching time have not kept pace. Growing pupil numbers and decreasing resources for education only worsen the situation. Bruce (1986) states that, on average, little more

than one hour per week is devoted to writing instruction. There are also other factors. Pre-writing activities are rarely encouraged; in general, writing is unprepared and triggered by short assignments or essay titles only. Exercises in grammar, punctuation and spelling in isolation still predominate. Generally only the teacher, and not peers, give feedback on writing. The feedback that is given is seldom used to revise text and first drafts become final versions.

It is within this context that we consider the use of writing support tools and the notion of an Expert Writing Model.

2.2. Word-processors as composition tools

Word-processing software was one of the first applications to make use of personal micro-computers and today it is probably the single most used (important) software tool. Before the advent of personal computers, word-processing was limited to crude text editing programs, designed primarily for editing software programs, and only available on large mainframes. The first micro-based word-processors, such as MicroPro WordStar which ran on CPM (™ Digital Research) based machines, were solely text based and used strange combinations of control key strokes to alter and rearrange the layout of the finished document. The operating systems were not capable of displaying different character fonts on the screen so control codes had to be used to denote a change in text style (eg. italic). The advent of graphical based operating systems, such as the Apple Macintosh, led to the demise of character based word-processors and the birth of WYSIWYG (What You See Is What You Get) technology. Word-processors such as MacWrite (™ Claris Corporation) could display a multitude of different fonts, styles and text sizes. Also graphics could be integrated into written documents and laser printers could be used to produce documents of a very high quality. Sophisticated desktop publishing facilities can also now be obtained from personal computers (eg. Aldus PageMaker).

The original WordStar has gone through many changes. For example, version 5 offers an easy to use pull-down menu interface (conforming to IBM's System Application Architecture), WYSIWYG, page preview facility, document outline facility, user-defined menus, spell checker and many other powerful features.

Researchers have found that children and adults of different backgrounds can master the communications skills needed for interacting with a text editor. Learning to use the basic commands of a text editor takes from a few hours to a few days, but then it becomes a routine skill (Card, Moran & Newell, 1980).

Some work has been done to examine the benefits gained from using word-processors within English education. Candy (1985) has carried

out several exploratory studies into the use of computers within schools. The use of a word-processor as a tool for composition was studied and some interesting observations were made:

> Pupils were enthusiastic about using a word-processor for composition.
>
> Direct composition on the word-processor was the most common approach.
>
> More emphasis was placed on neatness as opposed to accuracy of the written text.
>
> Tendency to place more emphasis on correct spelling rather than on composition and 'marshalling of ideas'.
>
> Teacher and student expectations were raised about the standard which could be achieved.
>
> More attention to the development of ideas and the ordering of sentences and paragraphs during the actual composition process.
>
> This was accompanied by an increase in discussion between the students and the teacher. The early discussions were mainly concerned with the use of the editing facilities; later this turned to accuracy of spelling and punctuation, and in the case of students who persisted with writing activities, gave way to discussion about ideas and organisation of material.

Daiute (1983 & 1985) outlines some of the difficulties inherent in writing and the benefits to be gained from using a computer (word-processor) for composition.

> The physical act of writing (non-computer) is slow and sometimes painful.

It is difficult to make changes to hand-written text. This can discourage writers from making improvements to the text. Young people are particularly reluctant to add or to change words because the result looks messy.

The computer can temporarily relieve some burdens on short-term memory.

The computer seems like an audience, thus stimulating the writer to take a reader's point of view.

It is fairly clear that a word-processor can be an excellent tool for composition and may indeed help to improve a child's writing skills. But, as Candy (1985) points out, the use of computers (word-processors) without an appropriate strategy can be unproductive and even be detrimental to the development of writing skills. Candy states that there is a requirement for structured activities which are integrated with the other aspects of the classroom such as talking, reading and collaborative work in general.

> *"Where students were left to their own devices, there was a tendency to copy up for neatness without using the full potential of the word processor. This short term emphasis on presentation of material is counter-productive to establishing a constructive approach to writing and a genuine sense of achievement in the age group concerned ... the use of word processors can place an increased demand on teacher attention rather than the reduced role which is sometimes imagined."* Candy (1985)

Cummings (1988) is very enthusiastic about the educational benefits of using computers (word-processors) to teach English composition. A picture is painted of classrooms being transformed into 'composing workshops' where 'students spend class time writing, talking freely about their writing, pacing themselves, and copiously revising their papers as they consult the instructor and their peers'. She talks about the redefinition of teacher-student roles, relationships and expectations, with the teacher guiding and

directing the student to make new writing discoveries. These observations all point to the fact that rather than reducing the workload of the teacher, the use of word-processors require the teacher to acquire new skills, to have greater managerial control, and to spend more time guiding and revising the students work.

2.3. New composition environments and research

Progress continues to be made in the development of specific applications to assist written composition and language exploration. As discussed above, there are several important factors in writing such as, getting ideas, organising thoughts, composing, editing and revising, and obtaining feedback about one's writing.

The software aids for writing tend to specialise in a particular domain rather than attempt to support every single aspect of writing. In this chapter these domains have been grouped into programs which aim to develop the child's underlying language model, tools for organising thoughts, environments for stimulating 'invention', and text analysis and support systems.

2.3.1. Language models

Models can be valuable as learning aids. A good way to understand the laws, constraints and possibilities of a complex rule-governed system is to build models of the system, subject to the same rules, and then perform experiments on them. This has been shown to good effect in mathematics using the Logo (Papert, 1980) programming environment. Sharples (1985) suggests that a child needs to create and manipulate language structures at all textual levels, from the 'word' to the 'section'. Computer-based modelling aids have been created to allow such exploration.

A/AN and S-ENDING are two programs devised by Johns (1983). The A/AN program places the correct form of indefinite article before a noun phrase typed in by the user. S-ENDING carries out a similar process for plurals. The purpose of this exercise is to allow the children to infer the rules of word and phrase formation.

ILIAD (Bates and Wilson 1982) was devised to manipulate language at the sentence level, and is aimed at deaf children who have difficulties in mastering such language forms as negation, question formation and sentences containing complex verb-phrases. ILIAD is

based on a powerful sentence generator based on a transformational grammar. The child is prompted to make a sentence about a proposed object. The grammar then checks the validity of the child's answer and prompts for any necessary corrections.

STORYMAKER (Watson 1986) is a branching program that allows a child to build computer adventure games. The emphasis is placed on the student developing a plot and structure on which to base an adventure. The content of the adventures/stories is not the main stimulus to learning; the child learns by understanding and manipulating the model (in this case story structures) represented in the program and by discussing their experiences with a teacher or peers.

PHRASEBOOKS and BOXES (Sharples 1985) are two additions to the LOGO environment that allow children to classify words, create their own dictionaries and phrasebooks, devise a quiz, write a program that will converse in natural language or build their own 'adventure games'. The aim being, to provide a large user-friendly modelling kit for language.

2.3.2. Organising thoughts

Organising one's thoughts is an important activity of writing. Expert writers often build up an outline framework of the essay to be written before actually starting to write. The next section discusses work done in prompting children to build on ideas before and during writing. There are however, several commercially available programs which are suitable for storing, controlling and organising thoughts.

FRAMEWORK (Ashton-Tate) is an integrated package (database, spreadsheet, graphics and word-processor) which includes a facility for constructing outlines. The word-processor creates empty, numbered outline structures which the user can fill in before commencing writing. These outlines can then be used as headings and titles within the main text. An item in the outline can also be any

framework structure, such as a spreadsheet or a graph. Related programs, such as THINKTANK (Living VideoTex Inc) are now available which are marketed as 'idea processors'. These programs support such processes as 'brainstorming', outlining and grouping of different data sources. Simple outlining facilities are now common in most high-specification word-processing packages.

The SPIRIT system (van der Geest, 1986) specifies the components of a package to support creative writing for secondary education. The main focus of this system is a planning tool which helps the pupil to choose an appropriate subject for the text, generating ideas about the subject, and making an organisational plan for the text structures.

The Writer's Wordbench (Newman, 1989), which was developed by the College of Education in New York is a specialist software product which enables the writer to prepare different parts of a document and then integrate them into a single format, which can then be edited. This package is commercially available. It includes an 'Outliner' which enables the writer to put headings and sub-headings in any required order, a 'Notetaker' which allows the creation of electronic notecards which can be linked to Outliner headings, a 'Reference' tool for recording sources, a 'Format' tool for enhancing text and formatting, and a 'Writer' which is the word-processing application. However, there is a danger that the use of tools such as the Writer's Wordbench will require a more disciplined and organised approach to creative writing, which initially may add an additional overhead to the task of writing itself.

Hypertext is a technology which allows a user to create units of information and link them to other units. Related notes can be joined by creating buttons and/or icons which establish the link and the relationship between items. The NOTECARDS system (Xerox Parc, Halasz et al, 1987) includes a multi-window display that allows a writer to create individual notes that can be linked to other notes. This technique was used in the STORYSPACE (Bolter & Joyce, 1987) program, a hypertext system for constructing interactive stories. The program allowed the user to build episodes of a story with links to other possible episodes. The person 'reading' the story could then

interactively traverse the story, making decisions as he went which would determine the next episode to be read.

Hypertext is a powerful technique for controlling and browsing large amounts of data. The advent of HyperCard (Apple Computer, 1987) on the Apple Macintosh, which is a powerful hypertext system and programming language, has lead to far more hypertext applications becoming available.

The term 'Hyper-Media' is currently in vogue. This refers to the integration of Hypertext and different multi-media input and output devices and computer systems. The aim is to access and integrate different data objects such as video, speech, graphics, text, within a single application. The end product of which is the creation of an electronic 'book' environment combining multi-media and hypertext capabilities. The impact of this type of environment on written composition and the teaching of writing is still unclear, but it should prove to be an interesting and fruitful research area. See Yankelovich & Meyrowitz (1985) for a discussion of the type of facilities available in an electronic book environment.

2.3.3. Stimulating invention

Computer-based writing aids can offer assistance to a writer, or provide a different medium for composition. Word-processors are a form of writing aid which have already been discussed. However packages have been produces which attempt to extend the functions of word-processors, be it giving advice on spelling, punctuation or style, or complete tutoring systems.

The word-processor can be used to stimulate invention. Marcus (1988) describes the use of a word-processor to store written assignments on a disk. A student can then load the assignment as a text file, read it from the screen, and follow directions contained in the file that direct the student in a guided manner to add text to the file. The design of the interface, ie use of margins, placement of text, colour, 'ideas' per screen page, etc; and the design of the '*innerface*',

the qualities of the activity itself are highlighted as areas which need further research.

The WRITER'S ASSISTANT (Levin et al, 1983) was based on a Pascal text editor with additional commands that allowed a child to check the spelling, experiment with word combinations and to merge sentences into a paragraph. The system was used to create classroom newspapers. Dynamic support within the writing environment was provided in several ways. Some sections of the newspaper provided considerable structure, requiring only that the children fill in the blanks. Other sections provided only partial support. A story was started but left for students to finish, or a question was posed and students inserted their own replies.

Much work has been done on generating ideas for student's writing. Burns and Culp (1980) describe a system which encouraged a student to 'brainstorm' a particular subject. The student would choose a subject and the computer would prompt the student to write about that subject. The feedback given by the system was based on word length clues, answer length clues, clarification strings and a brief list of direct commands.

A similar technique was used for idea generation in the SEEN program (Schwartz, 1982), which would lead students through prewriting exercises specifically tailored to literary topics. The general format follows (Rodrigues R J & D W 1984):

(1) pick an X
(2) create a hypothesis X=Y
(3) argue that X=Y for different kinds of evidence
(4) consider conflicting evidence

For example in developing a thesis about characterisation, the computer begins (student's responses are in italics):

Name a fictional character X in a literary work.

Satan in Paradise Lost

Describe the character X by completing the following: X is ....

Satan is tricky

Provide evidence to show that X is Y: What does Satan do that shows Satan is tricky ?

He enters the body of a serpent to disguise himself.....

Text analysis techniques were used in the CAC and CAC2 programs (Woodruff et al 1981) to offer children advice on composing persuasive text. The CAC program initially acts as a simple word-processor. However, if the child presses a 'help' key or the terminal is inactive for more than 20 seconds then the program prompts with a help menu. If 'help' is chosen then the program can offer advice on such things as 'following an argument plan' or 'producing the next sentence'. The guidance offered by CAC is based on the text most recently typed by the child. For example, if the child asks for advice on producing the next sentence then the program searches the last complete sentence for a keyword such as 'believe', 'reason' or 'example'. On finding the word 'reason', for example, the computer would print 'Let's say more about your reasons so the reader will understand'. Different tutoring strategies were tried with the CAC2 program. For example, instead of waiting for the child to ask for help, the program would interrupt after each sentence, presenting a question such as 'Do you have an opinion on this topic' or 'Have you mentioned any facts to support your reason'. Each question is determined by the child's response to previous questions and they are intended to emulate those an expert writer might ask himself while composing. However, the CAC programs were limited in that they were based on simple key-word analysis of the written text and the use of stored canned phrases.

2.3.4. Text analysis and support systems

The simple word-based analysis of text applied by CAC was an attempt to provide a computer system that could apply simple text understanding in a tutoring environment. However, the analysis of written text by computer has for a long time been a major goal of the computer industry, the aim eventually being to automate the writing process completely. The production and handling of vast amounts of paper-based documentation particularly by the business sector has for a long time been a costly exercise. The advent of business computers and office automation systems has increased the flow of written text and dispelled the promise of the so-called paper-less office.

It is worth examining some past approaches to text analysis and support systems. These range from simple style formulas based on word/sentence ratios, to complex grammar checkers and text summarization systems.

The simplest method used to advise the writer on the readability of his text is to analyse the word and sentence length, and compare the results against a pre-determined readability scale. Several different scales of readability have been devised (eg. Gunning Fog index, Flesch-Kincaid readability formulae) to measure such concepts as the reading/writing age that the text is suitable for, the degree to which the meaning of the text is being obscured (so-called FOG index), and the degree to which the sentence length varies.

As computer systems are ideally suited to applying number based formulas to large amounts of input data (ie. text), several of these techniques have been implemented as 'style' analysers (eg. Readability Plus for the IBM PC) and as add-ons for word-processing packages. The general usefulness of this type of support seems to be very low. A readability index does not help one to highlight problem areas in one's writing style, or suggest remedies or improvements. Also it is not possible to apply the same index to all types of written work (eg. a technical report will have a completely different style and content to a newspaper article).

The Bell Laboratories in the USA carried the concept of readability indices a stage further with the development of the Unix Writer's Workbench (Frase, 1983 & MacDonald, 1983). The main aim of this system was to improve the quality of the large amounts of technical documents produces by Bell.

It was recognised that the review and evaluation of technical documents was costly and time-consuming. The Writer's Workbench system provides a simple set of commands that deliver many of the assessments needed in documentation work. These include editorial comments on punctuation, word use, spelling, and text abstractness, and include an analysis of the grammatical parts of speech and calculations of overall text readability.

On the completion of a piece of text, a writer could submit it for examination to the Writer's Workbench system. The Writer's Workbench consists of the following separate programs each of which examine different aspects of the text:

SPELLWWB: conventional spell checker. Allows the user to interactively correct mis-spelled words and remember words in a personal dictionary.

PUNCT: searches for simple punctuation errors and recommends changes. For example, move commas and periods to the left of double quotes, capitalise the first letter of sentences and balance quotation marks.

DOUBLE: identifies consecutive occurrences of the same word.

DICTION: searches a text file for phrases that writing experts have classified as wordy or frequently misused. It also highlights those phrases which may have a sexual bias. As technical documents tend to legitimately include such 'wordy' phrases such as 'terminate' (instead of 'stop' or 'end') there is also the facility for the creation of personal dictionaries of allowable 'wordy' phrases.

SPLITINF: uses a 'part-of-speech' analysis program to find infinitives that are split by adverbs.

STYLE: uses a readability index to identify information on the average length of words and sentences, the distribution of sentence lengths, the grammatical types of sentences, the percentage of nouns, and the number of sentences that begin with expletives.

PROSE: provides an interpretation of the results obtained from STYLE. The best technical documents written in Bell Labs were used to statistically produce a guide-line for good technical style (based on the indices of the STYLE program). The input text is then compared against these guide-lines and if there is a statistical difference then the program explains why this may make the text hard to comprehend. For example, it was found that the passive verb form was harder to comprehend than the active form; if a document had more than the standard 28.6 % of passive verbs then a tutorial message on the use of passive verbs was given.

FINDBE: finds all occurrences of the form 'to be', as this often indicates a passive sentence.

ABST: a list of 314 words rated as abstract are matched against the input text. If the percentage of abstract words is over 2.3 % (derived from the 'good' documents used to develop the PROSE standard) then the program suggests that more concrete examples be introduced.

ORG: is a tool for viewing the organisational structure of the text, by displaying only the headings and paragraph boundaries.

The Writer's Workbench is limited by the text features that the programs can recognise and the validity of these text features as indicators of reading difficulty. The programs do not use linguistic or

semantic parsing techniques and as such cannot understand the 'subject' and 'object' of sentences, or the underlying meaning. Without a parser for English, or some other way of interpreting the meaning of text, the programs cannot give feedback on the quality of the content and organisation.

The Epistle text-critiquing system developed by IBM (Heidorn et al, 1982, & Jenson & Heidorn, 1982) addressed the issue of grammar and style checking of texts written in English, and was based on syntactic analysis using an augmented phrase structure grammar (Heidorn, 1975).

Epistle uses a technique called 'fitted-parse' to detect grammatical errors. This consists of the following three steps:

1. Parse the sentence by applying grammatical rules to it. These rules are similar to Backus-Naur Form BNF, and attempt to build a syntactic representation of the sentence.

2. If the sentence was not parsed in the first step, then try again, but this time relax some of the conditions and apply some additional rules.

3. If the sentence is parsed in the second step, then make note of what condition had to be relaxed and where in the sentence the problem occurred, and pass this information to the error handler for display to the user.

The Epistle critiques do not cover all possible grammatical errors in English, instead it diagnoses the following five classes of errors:

1) Subject-verb disagreement eg. 'have' instead of 'has'
2) Wrong pronoun case eg. 'him' instead of 'he'
3) Noun-modifier disagreement eg. incorrect use of plural form
4) Nonstandard verb forms eg. 'wrote' instead of 'written'
5) Nonparallel structures eg. 'crediting' instead of 'credit'

Epistle also included checks for the following types of style errors:

1) Word-level critiques eg. identify 'business-ese' and words with bad connotations such as 'hate'
2) Phrase-level critiques eg jargonistic phrases
3) Sentence-level critiques eg. sentence too long, too many negatives
4) Paragraph-level critiques eg. too many passive sentences

The diagnosis of style errors is done by a set of encoding rules which are applied to the parse tree. These style critiques are similar to those provided by the Writer's Workbench, but because the Writer's Workbench does not use a parser it cannot identify grammatical errors such as where the subject-verb distance is too great.

The Epistle system provides a syntactic analysis of business letters. However it cannot handle the more 'difficult' kinds of errors associated with the semantic information contained in the sentences. For example, assessing the continuity across sentences and paragraphs, or understanding the purpose of the document and suggesting appropriate text for it.

Semantic analysis of text has been a major area of work within the Artificial Intelligence community. The main emphasis being the analysis of text for summarization rather than for writing support systems. Such work includes MARGIE (Schank et al, 1973), FRUMP (DeJong, 1979), IPP (Lebowitz, 1983), and BORIS (Lehnert et al, 1983).

The MARGIE system was the first major implementation of the semantic representational schema 'Conceptual Dependency' (Schank, 1981). It parsed natural language into conceptual dependency and then made inferences about characters and actions within the text.

The FRUMP system skimmed news stories to gain a semantic representation of it. The parser was made up of a 'predictor' which selected a likely script for the text, and a 'substantiator' which

attempted to confirm the chosen script by filling in its slots. The IPP system also attempted to parse news-wire articles. It used a set of semantic primitives to describe the events associated with news about terrorist incidents. The parsing process combined top-down predictions with bottom-up heuristics.

The BORIS system was tailored to recognise emotions within text. It used this information to understand the goals of the story protagonists and to answer questions about the text. One area that BORIS was used was to understand the motives of characters involved in a situation about marital divorce. The system was pre-loaded with expected plans, goals and motives for characters in this situation.

All of the above work on the semantic analysis of text had the common property of using a large knowledge base in order to understand a small well defined semantic world. This need for a large semantic model makes it very difficult to construct a general purpose parser for all domains.

The developers of the ALEXIS system (Jansen et al, 1986) considered that text understanding systems were too 'unintelligent' to take over the feedback tasks of the teacher. Instead they produced a system which could help the teacher to comment on the products of his students. The system translated simple codes introduced by the teacher into feedback texts which would inform the student about what was wrong with their texts and/or what should be done to improve their next paper. ALEXIS contains a total of 1100 commentary texts and it has received favourable results when used in a university teaching environment. This type of system may be the current practical limit for implementing text understanding systems in the real world.

2.3.5. Hypertext, collaborative writing, and interactive fiction

The technology called 'Hypertext' (and HyperMedia) is currently very much in vogue and recently it has been applied to creative

writing, and in particular the areas of collaborative writing, and interactive fiction. Hypertext systems typically allow a user to create units of information and link them to other units. Related notes can be joined by creating buttons and/or icons which establish the link and the relationship between items. This ability to represent information and associative relationships, with an intuitive user interface, provides an ideal environment for representing the relationships between units of texts as part of a creative writing system.

The WE system (Smith et al, 1988) is a hypertext writing environment which helps writers transform loose associative networks of ideas into a hierarchical structure and then write a document in accord with that structure. It is a tool for organising and manipulating ideas into a coherent document. The developers intend to link WE with an object-oriented database which can serve as a repository for relevant structural information. The WE system can be viewed as a merge between writing and database (hypertext, object-oriented) technologies.

Usually more than one author is involved in creating larger documents. This makes the task of creative writing far harder, as an extra level of management and communication is needed to organise the individual authors. The MUCH project (Rada 1988; Rada & Keith 1988) is an investigation into providing computer systems to support multiple users creating hypertext. Part of the project has looked at secondary school pupils working together to produce a group document. Here the children have produced an 'alternative' travel guide about places they had individually visited. They soon had to standardise the format and agree on headings and subheadings. This particular exercise used a single word-processor, yet the MUCH project hopes to address the issues involved in supporting multiple users interacting across a local area network, and will examine the type of problems raised by such work. They hope to use hypertext to create a semantic net to represent the relationships between the multiple users and the individual units of text produced.

Hypertext has also been used as a medium for a kind of flexible, interactive fiction. Storyspace (Bolter & Joyce, 1987) is a hypertext system which implements a scheme of episodes, and decision points or links. It has two modes: one for the author and one for the reader. The author creates his fiction as a series of textual episodes, using a structural editor. The reader sees the contents of each episode and may then reply by typing a string or pressing a button in order to branch to the next episode. This type of (adventure) authoring system can provide an exciting environment for creative writing, especially if used as part of a cohesive teaching strategy.

Hypertext is really an enabling technology for handling chunks of information and for navigating through it. Much of the work described above is relatively new and has not been specifically applied to education. However, collaborative writing is quite common as part of English education and systems which can support it, will be of benefit.

2.4. Where next?

Several different approaches to composition support systems have been described.

Daiute (1983 and 1985) outlines the advantages to be gained from using the word-processor as composition tool. However it is clear that the computer must be properly integrated within the curriculum. Work by Candy (1986) has shown that the use of a computer without an appropriate strategy can be unproductive and may even be 'detrimental to the development of writing skills'. Petersen et al (1984) offers seven criteria to consider when examining software designed for use in composition courses, and gives examples of the types of questions a teacher should be able to answer before using such software; most of which are concerned with how the software is going to be integrated with the existing curriculum.

The stimulus to discussion which occurs during the use of word-processors can place an increased demand on teacher attention rather than the reduced role which is sometimes imagined (Candy, 1986). There is a need for systems which can support the whole writing process, from the marshalling of initial ideas and thoughts, through to composition and interactive support, to revision and analysis of the text.

Hertz (1983) puts forward several objections to the type of syntactic analysis of text performed by the Writer's Workbench and Epistle systems, and denies any benefits from using them to analyse student texts. It is true that the syntactic analysis of these systems would be too rigid and inflexible to actually mark the text, in place of the teacher. However their use in a support role during composition has yet to be explored. Also these systems have yet to utilise a semantic understanding component.

Sharples and O'Malley (1986) argue that the current word-processing environments (ie. text-editor/spell-checker/outliner) suffer from two limitations; that is, 'they do not appear to be derived from an explicit (cognitive) model of the writing process', and 'as a result, they are

limited in their ability to represent and satisfy the constraints involved in writing'. Sharples and O'Malley put forward a framework for a writer's assistant which (i) allows the writer to specify explicit constraints, (ii) allow the writer to switch easily between any of the writing strategies, and, (iii) provide multiple views of the material at different levels of focus and from different perspectives. This framework is currently being used to produce a prototype of a writer's assistant (Sharples, Goodlet & Pemberton, 1988) that will offer the writer different views of the emerging document. The design of the system has been tested (Pemberton, 1988) using a HyperCard (™ Apple Computer) mock-up and will be implemented in Poplog (Sussex University). The eventual system could provide a good test environment for future cognitive theories.

We will return to the notion of a framework for a writers assistant later in the thesis when we consider the context of an Expert Writing Model.

Chapter 3. RATIONALE FOR A COMPOSITION SUPPORT SYSTEM

There is a growing understanding today in the educational world that it is the process of writing, as distinct from the end product, which must be the focus of pedagogical attention. However, very little has been done to provide appropriate computer based support for pre-writing and writing activities as part of an educational curriculum.

3.1. Aims and objectives

Work by Edmonds and Candy (1982) and Candy (1983 & 1985) undertook close monitoring of pupils between the ages of 14 and 16 years in their use of a software package. A choice of activities, including a personalized spelling bank, reading extension through Cloze procedure and a simple word-processor for writing were provided. They observed that the use of the word-processor promoted changes in the pupil's attitude to the act of writing itself in a number of different ways. For example, otherwise reluctant writers wrote more than usual, attention to detail was more evident in that the ease with which correction could be made was applied readily (the preference with hand written work was to leave it untouched even where errors were identified), and there was a clear preference for discussion with the teacher or fellow pupils during the actual composition process itself. These are recorded impressions by the teacher/researcher who was comparing notes with her normal experience of teaching writing in the classroom.

It was the observation that more demands were placed upon the teacher for advice and support during writing with a word-processor that gave rise to the notion that a computer system which could have knowledge of the writer might prove a useful learning tool. It is not the intention to provide the kind of tutorial support for grammatical/syntactical errors such as offered by the Epistle system described in Miller et al (1981) but to work at the development of early ideas for story and character creation, albeit at a very basic level.

Current educational, and writing systems do not support the writer in selecting the building blocks for a piece of text. Word-processors provide outline processors which the writer can use to plan a document, but the writer must still decide on what the outline headings should be. Text analysers can check a document for readability, style, and grammar but cannot tell the writer if a document is complete, or if sections are missing. Commercial systems have concentrated on the syntax and grammar of text as an adjunct to word-processor packages. Various research projects have produced representative computer systems that can semantically

analyse text for a given domain. But these systems have been too restricted or artificial to be transferred to real world applications.

With the premise that computer support for pre-writing and writing skills will benefit English education, a fundamental task of this research is build an experimental creative writing support system that can assist the writer throughout the writing process. This will provide the foundation for an analysis of the Expert Writing Model for story composition.

In order to achieve this objective an existing package called STORYMAKER (Candy and Schoenfeld, 1983) was used as a basis for a creative writing support system. The STORYMAKER package is a *non-computer based* system devised by teachers in a local school to provide children of a wide range of ability with springboards for writing. The STORYMAKER notes provide a format for children to select information about possible story types, situations, and characters, which can then be used as a basis for various teaching exercises including story writing.

The support system that was created for this experiment was called MULTISTORY which is documented more fully in the next chapter. Briefly, MULTISTORY uses the STORYMAKER package as a basis for pre-selecting the criteria for a story. This is then used by a compositional support system during the writing process. The following sections describe the rationale behind the MULTISTORY system in terms of the composition support, the ICAI criteria and the necessary natural language processing work needed to implement the system.

3.2. Computer supported creative composition.

Computer supported creative writing is a controversial subject. We have visions of computers teaching a class of students without the need for human teachers. This is mainly the province of science fiction. Alternatively we could see computer systems rigidly marking written text purely based on inadequate style indexes and stock stereo-typical phrases. This vision may be nearer to today's reality, and as such provokes the question 'why should computers be used to examine creative writing and clearly it must be impossible'.

By forcing the pupil to make decisions about the components of a story before writing it should be possible to use this information to assist in providing relevant suggestions on request. The premise being that if we roughly know what sort of story the pupil is writing, and we know the situation and the main character then we will also know what sort of suggestions we should be providing. Students in this way can be restricted to a pre-defined set of possible story parameter combinations. However, there is a danger that a small number of combinations can very quickly amalgamate to produce a large number of possible story permutations. For example, with 8 story types and 8 situations per type, and several character combinations per story type and situation, we will quickly produce a very large number of possible stories. It would not be feasible to produce an expert model for each possible story combination. However, for the purposes of this experiment it would be sufficient to support only one story type, situation and character combination to prove the concept.

It is noted that the support system is designed to determine at what stage the child is at in writing the story and provide a suggestion on the next stage/section of the story. There is a danger with this type of system that it could lead to pupils who are using the same story parameters to produce very similar stereo-typical stories leaving them little scope for exhibiting their own imagination. Undoubtedly the type of support offered by the MULTISTORY system would be too simplistic and constraining for pupils of average to good ability. The suggestions would tend to be viewed as offering information which

was pretty obvious anyway. The value of the MULTISTORY suggestions would best be found in supporting those pupils who were of low ability and imagination, and as such could benefit from successive prodding and ideas when writing the story. Some of the conclusions of the work by Candy (1985) into evaluating the usefulness of word-processors in English education, were that there was a need for structured activities which are integrated with other aspects of the classroom, and the use of a word-processor can place an increased demand on teacher attention. It is felt that a computer system like MULTISTORY (used to support the less able pupils) would be of benefit.

Would a pupil using MULTISTORY write a good story and what is a good story?

Is it one with a beginning, middle, and end; is it one which builds up the expectations of the reader and then makes an unexpected twist at the end; is it one which follows a well used formula (such as a Mills and Boon novel); does it start at the end and recount a story in hindsight; is it recounted from several different characters perspectives, does it use elegant vocabulary or stylistic flourishes? As we can see this is a very difficult question to answer. But it is usually fairly obvious what a badly written story is. There is often no overall writing strategy, or a simple *what-next* strategy is used which is a simple sequential account of events based on conversational heuristics. Bad stories usually have no plan which leads to a tangled story structure containing irrelevant information, or missing vital links.

Sharples (1985) suggests that inexpert adult writers difficulties begin with devising a story plan. They may be aware that one is needed but cannot produce one to suit the task. One method of imposing form on inexpert writers is to set 'milestones', which are incidents which must be included. Pupils of average ability and above would probably not benefit from MULTISTORY. However, the lower ability pupils could benefit from MULTISTORY in that it would provide them with an initial rough plan for a story, and could prompt them with ideas during the writing of the story. The use of MULTISTORY would not

lead to any masterpieces, and would tend to produce fairly similar single concept stories, using a simple story plan, but it is proposed that it would provide a springboard for the lower ability writers to improve their creative composition. It is the notion of how we represent the Expert Writing Model that is fundamental to this research.

3.3. ICAI criteria

Some of the claims put forward for ICAI systems were:

- that they should allow exploration by the student
- be controlled by both the student and the underlying system
- give hints, demonstrations and explanations
- answer questions and suggest challenges

The rationale for MULTISTORY against the criteria for ICAI systems is discussed in the following sections.

3.3.1. User interface.

The MULTISTORY user interface would need to be closely bound up with several external constraints, such as the age group of the users of the system, the type of task the system was trying to fulfil, the need to include additional support facilities (such as text editing), and the nature of the support provided.

MULTISTORY is aimed at low ability, early secondary school pupils. The user interface must be easy to learn and to use. It also has to be interesting/exciting and should brighten up a subject, which might otherwise be a dull and routine activity. A colour graphical user interface was deliberately chosen for this purpose, based on recognised guide-lines for the use of colour user interfaces (see Alderson & DeWolf, 1985).

A menu based interface can be used for the selection of story criteria during the story outline phase because the task is very structured and relatively easy to follow.

A simple word-processing mode will be needed for the actual writing of stories. Features such as 'cut', 'copy' and 'paste' for editing text, and the use of a mouse for selecting text and moving the cursor. The text-editor will provide a free-format natural language interface to the story support system ie. after selecting the help option, the entire

story text from the text editor could be examined as part of the analysis process.

The natural language processing component would be the weakest part of the system, and would be just sufficient to demonstrate the suitability/feasibility of this approach for story composition. Commercial natural language systems such as NATURAL (Adabas Inc) and NLMENU (™ Intellicorp) are constrained to understanding simple database queries using a pre-defined format (eg. 'Find the employees who joined this company before 1/1/87'). Their dictionaries are limited to relatively small commercial domains so that queries should easily be translated into a correct database query. Artificial intelligence research into semantic natural language processing has similarly constrained itself to small domains (eg. the 'divorce' domain with the BORIS system by Lehnert et al, 1983; and military aircraft information for the PLANES database query system by Waltz, 1978). Even with this constraint the natural language systems have required very large knowledge bases to contain the semantic and pragmatic knowledge representations of the problem domain. The 'real-world' knowledge required to understand some of the simplest human actions is of a vast scale. The problem of finding suitable generic structures for knowledge representation, and methods of applying these structures, is still one of the major problems of artificial intelligence research.

The MULTISTORY system would need to translate and understand free-format, unconstrained children's writing. This task could be assisted by forcing the pupil to first decide on a story type, situation, and character before starting, which would allow the natural language parser to be restricted to only containing knowledge about the possible stories that could be written for the chosen story parameters. Even so, to develop a parser which could fully 'understand' just one story situation would be beyond the realms of this research.

The text parser within MULTISTORY is required to obtain a machine understandable representation of the story text. The parser could have several limitations such as only recognising simple

transitive and intransitive statements with a single main noun-phrase and verb-phrase, and it could have a limited dictionary of about 240 words. This would be acceptable for the purposes of this research

3.3.2. Embedded student model

The embedded student model is the representation of the state of the student's expertise at a particular moment. This information is essential if a support system is to provide any form of personalized help that is relevant to the student's current level of expertise.

An approach that is often taken, is to create an expert model of a task and then construct a student model from a subset of the expert model (see chapter 1). MULTISTORY would use a similar approach to build part of its student model.

An initial 'Selection' phase of MULTISTORY would build an initial coarse model by forcing the student to choose a story type, situation, character, and selected character attributes from restricted lists. Each story that a student writes will have its own associated set of decisions that are made in this way. These decisions form a pact between the pupil and the system ie. the pupil elects to write a story based on the chosen criteria, and the system would attempt to provide support based on these decisions.

The MULTISTORY system would also need to use the full text of the pupil's story to augment the student model. This becomes apparent, when considering the nature of the problem, that is, the system would be trying to provide suggestions about the possible plot alternatives that could be available for a particular story (piece of text). Unless the system could fully analyse and 'understand' the concepts being expressed in the text, then it could not provide adequate relevant suggestions, or at least provide some extra useful input to the composition process.

The story text, would be parsed into an appropriate representation, which would then form a major part of the current student model. This model could then be compared against the relevant 'expert' model for the chosen story type/situation/character/etc.

3.3.3. Domain knowledge

The domain knowledge component of the system would need to represent the expertise required to write a story. In this context, the expertise should be of the form 'Ok, I can see what you have written, why don't you try this.....'. MULTISTORY will ideally need to recognise what has been written, relate that to its own internal model of what should be a good story for the chosen criteria, and make a suggestion to the writer based upon the difference between the input text and this expert/story model, its so-called domain knowledge.

The input to the expert model will be the student model, that is the parsed story text, and the chosen story criteria.

A first stage MULTISTORY system would be constructed where the expert model would only examine the story criteria (ie. the decisions taken by the student prior to writing), and no analysis would be made on the story text. This model would consist of a rule-base of story suggestions and their associated firing conditions. These rules would test the story criteria, and the resulting suggestions would be ordered from those most likely to suit the beginning of the particular story, through the various plot developments, through to the end of story suggestions.

The second stage MULTISTORY support system would need to create a student model by parsing the story text into an appropriate form. The expert model would consist of several components. A semantic analysis phase is required to relate the syntactic structure obtained from the parser to recognised events and actions within the story. Once this semantic representation has been obtained it should be possible to compare the events/actions against expected events/actions within the expert model. The omission of such could

indicate a possible suggestion point. The presence of additional structures could indicate an important event, which could be used to stimulate dialogue between the student and the support system. It is the contents and architecture of this Expert Writing Model which is the basis of this research.

3.3.4. Tutoring strategy

For the purposes of this experiment the MULTISTORY support system would be a passive system which would not be interrogated until asked by the student writing the story. The support system would act as a consultant, servicing requests from the user to the expert model (reactive). The passive 'consultant' type mode would be acceptable given the aims of this research. However, further work may be needed to investigate more sophisticated multi-mode pro-active dialogue strategies.

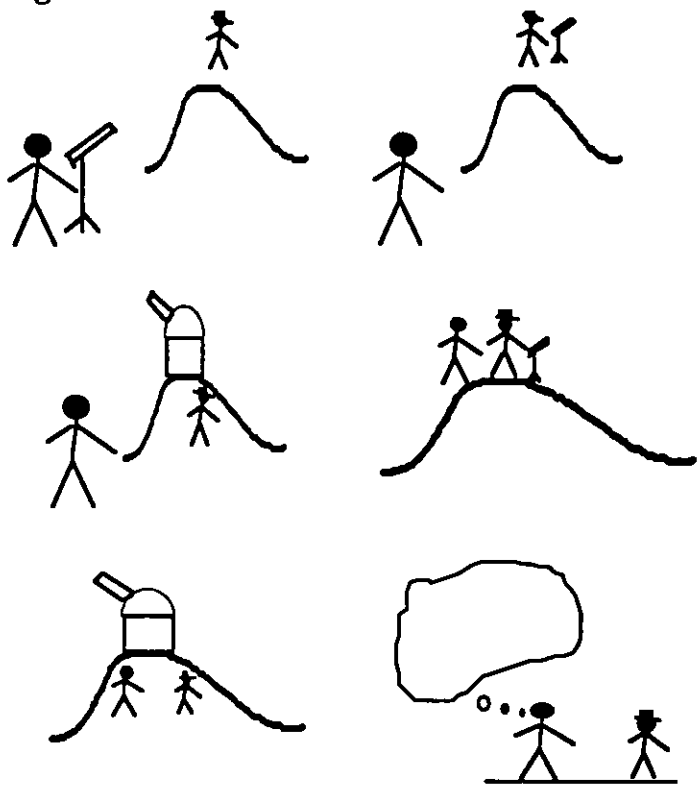
It should also be possible for a student to write a complete story without once asking for a suggestion from the support system. Also, the system would not attempt to mark any written work. This would be left entirely up to the teacher.

3.4. Natural language processing for text analysis

The natural language processing component would be a major part of the MULTISTORY system as it would be used to extract the student model in the form of a parsed representation of the story. This could then be used by the expert model to devise an appropriate suggestion. However, for this experiment the natural language component was purely a necessary task which was needed to be completed in order to prove the overall thesis.

Natural language processing is a hard problem. We use language as a short-hand notation for transferring knowledge. The coding of language by the writer and the subsequent decoding of the text by the reader involves the application of many different knowledge sources, yet we manage to communicate quite successfully. The main problems occur because language is ambiguous and ill-defined. For example the sentence "*I saw the man on the hill with the telescope*" could mean any one of the following situations (figure 2):

Figure 2.



ETC...

This problem becomes more difficult when we start combining sentences into paragraphs, chapters, stories and other combinations.

Early work on natural language processing (1953 onwards) concentrated on the machine translation of text between different languages. This work was based on pure syntactic analysis involving dictionary look-up and substitution and grammatical re-ordering. After a lot of work there were still no cost effective systems. The main problem was that these systems made no attempt to capture the meaning of the text. For example, the translation of the following sentence from English into Russian and back again (using only syntactic translation):

Before: *"The spirit is willing but the flesh is weak"*
After: *"The vodka is strong but the meat is rotten"*

Much more knowledge needs to be applied. Broadly speaking the following basic linguistic distinctions can be made.

Syntax: determining the grammatical structure of sentences usually in the form of linguistic knowledge. For example, a sentence is made up of a noun-phrase followed by a verb-phrase; a noun-phrase can consist of a determiner followed by a noun; and so on. Augmented Transition Networks (ATNs) can be constructed to represent the different possible routes through a defined grammar.

Morphology: word inflection. Linguistic knowledge. For example, 'being', 'was', 'am',' is' and 'were' could all be forms of the 'be' verb.

Semantics: establishing the 'meaning' of text. The correspondence between inner-mental models and the 'world'. Typically a province of 'artificial intelligence', many different methods of knowledge representation have been used to map text onto knowledge structures. eg. 'semantic nets' (Quillian 1968), predicate calculus logic, conceptual dependency (Schank, 1972), 'case grammars' (Fillmore, 1968), 'preference semantics' (Wilks, 1975).

Pragmatics: this is concerned with 'everything else', such as 'common sense', 'speech acts', 'beliefs', metaphor, irony, and so on. Higher level knowledge structures are used to represent stereotype text structures and intentions. For example, 'scripts' (Schank and Abelson, 1977) were combinations of conceptual dependency structures describing stereotype actions and events; PAM (Wilensky, 1981) was a system that tried to explain the reasons for the actions of story protagonists. This work included representations for the plans a character might construct to satisfy underlying goals, the 'themes' of the text, 'explanations' for actions, and 'predictions' of future actions. The BORIS (Dyer, 1981) system attempted to model the role of 'affect' in narrative. This was concerned with representing the moods and feelings of story protagonists eg. affection, disgust, hope, relief, etc.

The relative importance of each of these units to text understanding has been the subject of much work and is outside of the province of

this thesis. Some linguists have tended to opt for pure grammatical/linguistic analysis, while some A.I researchers have only considered the semantic representation. However, each domain has its own contribution to make to text understanding. Rather than create separate distinct components, future systems will need to combine these separate knowledge sources into a single coherent structure.

Where does this place MULTISTORY in terms of implementing a natural language component? A solution was provided in the form of a predicate calculus parser (Hinde, 1986) which could provide grammatical text analysis using a declarative clause grammar based on a context-free grammar. This program would parse the text non-deterministically by using a tree searching algorithm. While traversing the parse tree a predicate-calculus representation of the sentence is built up. If a particular grammar rule is unsuccessful, then the system back-tracks to the previous node of the tree, un-building any resultant knowledge structures and a new route is taken. This process is repeated until all of the input text is successfully parsed. The parse will fail if all possible routes are traversed unsuccessfully.

This program is described more fully in the next chapter. Although, not representing the state of the art in terms of natural language processing it would provide a viable solution for implementing a text-processing component as part of the larger ICAI system and a platform for investigating the Expert Writing Model (MULTISTORY).

Chapter 4. MULTISTORY: DEVELOPMENT OF AN ICAI SYSTEM FOR STORY COMPOSITION

It is proposed that computer support for pre-writing and writing skills will benefit English education. Using appropriate technologies an experimental creative writing support system will be developed that can assist the writer throughout the writing process and which will provide assistance in making plot level decisions. This will provide the foundations for an investigation into an Expert Writing Model for story composition.

4.1. Introduction

The previous chapters have described Intelligent Computer Assisted Instruction (ICAI) and the background to composition support systems. This was followed by a discussion of the rationale for a composition support system MULTISTORY.This chapter describes MULTISTORY; a system to support story composition for secondary school children aged between 11 and 15 of low to middle ability. This includes a description of the two main components of the system: the user interface and the support system.

4.2. MULTISTORY - system design

An initial exploration of composition support systems was carried out by Gardner (1984). This work was based on a 'Storymaker' package (Candy and Schoenfeld, 1983) which is a non-computer based system devised by teachers in a local school to provide children of a wide range of abilities with springboards for writing. This method was adapted into a computer system which could provide limited tutorial support (STORYWRITER) which was well received during discussions with teachers.

Briefly, the STORYWRITER system was implemented on an ATARI 800 micro-computer and was based on the 'Storymaker' package. It allowed the user/student to select the criteria for a story and then write a story based on these decisions. However an extra element was added, which was the notion of providing computer based support during the writing process. The premise was that if the computer knew the basic facts about the story, then it would be possible to use this information to help the user write a story. This was implemented by linking the ATARI to a GEC4090 mini-computer. Then by creating a Prolog database on the GEC4090 of the decisions made by the user during the initial phase, it was possible to examine the data using Prolog rules and provide suggestions during the writing phase.

STORYWRITER suffered from several drawbacks, such as: it needed a more powerful system (poor performance); it had a poor user interface; there was an unwieldy set-up phase; it was not fully developed/tested; and it had an insufficient knowledge/rule base with no means of establishing the proper context for a suggestion.

One of the aims of the MULTISTORY system was to overcome many of the limitations of STORYWRITER by developing a complete integrated system to support the activities before and during writing. The interface would need to be easy to learn and use, and the suggestions should be useful and relevant. The major problem with STORYWRITER was that the suggestions could not be based on what the child had actually written. The system could only base its

suggestions on the criteria known before writing started. It did not contain an Expert Writing Model.

The hardware/software base of the STORYWRITER project was clearly inadequate. What was needed was a powerful stand-alone micro-computer. A Research Machines Nimbus was chosen to fill this role. (See Appendix 4 for technical specification). The Nimbus had the following advantages (at the time):

- Powerful 16-bit processor.
- Sophisticated colour graphics chip and software libraries
- Mouse (and touch-screen) support.
- The 'Next generation' educational computer.
- Stable software/technology base.

The MULTISTORY system design was conveniently split between the software modules for the user interface and the modules required to provide support in the form of meaningful suggestions during the writing phase. The Prospero ProPascal™ package was chosen as the development language for the user-interface and file handling and this was augmented with assembler code where necessary. Robust software libraries were available for this language which would minimise the effort needed for the necessary graphical and peripheral control. The compositional support system and text analyser was implemented in Prolog. This is an ideal language in which to develop rule based systems, and it is particularly suited for the implementation of text parsers. The following sections describe these components followed by an overview of how they were integrated into a coherent system. The MULTISTORY system design was originally completed as part of a proposal to the government funded MEP project and is more fully described elsewhere (see Gardner, Edmonds and Candy, 1985).

4.3. User Interface.

One of the aims of this work was to develop a coherent, easy to use (and learn) user interface for story composition. This was particularly necessary, as the system was aimed at young children. A graphical colour interface loosely based on the WIMP (Windows, Icons, Mouse, Pull-down menus) technology was developed. (See Appendix 2 for the main User Interface and control program listings and for a graphical overview of the system design, and Appendix 3 for sample MULTISTORY screens).

A set of icons are displayed across the top of the screen. These represent the currently available options. The user can escape from whatever they are doing at any time by selecting one of these icons. Below this is the work area (analogous to the 'desktop' metaphor used by commercial WIMP systems) which is used to view text, make selections and edit a story. Each user will be given their own floppy-disk on which to create stories. The MULTISTORY system does all the file handling on each disk and creates a username on the disk by which to identify each user. The user is completely shielded from the underlying operating system.

The user has to insert their computer disk before being allowed to proceed.

First the user has to select information which will be used as a basis for writing the story (select the 'Create' icon). In this way a knowledge base of information about the actual story being written is created on the users disk and the system will not allow the user to write a story unless this information is found. A list of the current story files on the floppy disk is displayed and the user has to enter a new story name. Once this has been done the criteria on which to base a story must be chosen. To do this the user must first choose a story type by clicking on a story type from the list displayed. Currently this can be one of 'Revenge', Love', 'Fantasy/Horror', 'Animals', 'Growing up' and 'Conflict'.

The user then selects a story situation (see Appendix 2.10 for a list of the story situations used by MULTISTORY). We currently store eight situations per story type, but there is no limit on this number other than imposed by practical constraints. The system displays the text for the chosen situation and story type. The user can either decide to use it, or they can ask for a different situation until a choice is finally made.

Next, a main character is chosen (see Appendix 2.11 for a list of the characters used by MULTISTORY). Each story situation contains a list of characters which can be used with it. We have a database of about 30 character descriptions some of which are applicable to a certain situation, others can be applied to several story types and situations. The system displays the text describing one of the available characters. The user can either accept it or ask for a different character for the situation. This is repeated until a choice is made.

The user then determines the main character's profile (see Appendix 2.12 for a list of the main character attributes used by MULTISTORY). A list of different character attributes are displayed on the screen. Alongside each attribute are the numbers from 1 to 5 and a '*'. The user can click on a number to select an appropriate description, or they can click on the '*' to randomly select a description for the attribute. The description that is given is appropriate to the number chosen. eg. a '1' for 'health' would make the character 'weak and sickly' while a '5' is 'extremely fit'. This process is repeated until the user has chosen as many attribute descriptions as they want for the particular story being created.

The story 'create' mode is now ended and a new story information file is created on the user's floppy-disk. At this point the user has decided on the foundations for a story and is ready to start writing a story based on it.

To write a story the user must select the 'Word Process' option. The system will then display the available created story files which are found on the user's floppy-disk. The user must select one of these

files as a basis for a story. When a file is selected by clicking on it, the currently written story (blank if starting a new story) is loaded and displayed. Scroll bars are placed along one side of the work area and the user can now write the story. There are limited word-processing features, such as scrolling, insert/overwrite, cut/paste and save and exit. A new icon is also displayed in the top icon area. This can be used to activate the support system when an appropriate suggestion for the story being written is needed. The support system should give the user a suggestion on the development of the plot or possible alternative approaches to the story. This suggestion, supplied by the support system, is displayed in a window across the screen. It is up to the user whether they decide to take notice of the suggestion or ignore the advice given.

4.4. Support system.

The aim of the support system is to provide relevant and useful suggestions to a user writing a story. The role of the system being similar to the scenario of a teacher overseeing a class of pupils writing stories. The teacher will have set a topic for the class to base their ideas on, and writing will have started. During the session, some of the pupils (probably the less able) are likely to ask for assistance. The teacher will need to know the basic facts about the story and how much has been written before help can be given. Using this information the teacher will offer a suggestion to the user on a possible plot sequence for the story being written.

MULTISTORY forces the user to make all the decisions about the story type, situation and main character before writing starts. The main issues for the support system were:

•*Could a rule-based support system be constructed to use this information to provide useful suggestions ?*

•*Could a text and plot analyser be constructed, to scan the text so far written, form a representation of the story , and use this to advise the support system on an appropriate suggestion to give ? This would require the development of an Expert Writing Model for story composition.*

The methods used by MULTISTORY to address these issues are described in the following sections. (See Appendix 2.7 for a listing of the rule-based support system and Appendix 1 for listings of the text analyser support system).

4.4.1. Simple rule based support system.

The first step was to construct a rule based support system that could use the information chosen by the writer (the story information file) to construct an appropriate suggestion.

The following is an example information file that a person using the system might create (see Appendix 2 for full program listings). The data is represented as Prolog predicates:

```
story_type(revenge).
story_situation(sit_3).
story_character(char_14).
char_att_1(2).
char_att_4(1).
char_att_5(2).
char_att_7(3).
char_att_10(5).
char_att_12(2).
```

This file describes a story of type 'revenge' with situation number 3:

> *"A child, often beaten by a rather unloving dad, leaves home as soon as he can. Sometime later he returns to his home town intending to pay his father back for his years of suffering. But he finds the father more to be pitied than blamed"*

The character is number 14; which is:

> *"DYLAN. Male 11. An orphan brought up by a series of foster parents, some good and kind, others mean and nasty: none of them permanent. Like many in his position he wants to know who his real parents are and is prepared to spend effort locating them."*

The character has been set with the following attribute settings:

Attractiveness--"Average features, and dull"
Friendliness--"Unsociable and hostile"
Health--"Always has coughs and colds"
Intelligence--"Average ability, common sense"
Self-confidence--"Opinionated, bigoted, cocksure"
Truthfulness--"Unreliable, no principles"

The program RUNSUGS is the main controlling program.

```
retractall(X):-retract(X),fail.
retractall(X):-retract((X:-Y)),fail.
retractall(_).
check_consult:-clause(storytype(_),_).
check_consult:-consult('f:cstoryf.pro'),
          storytype(St),storysit(Sit),cons_sug(St,Sit,File),
          conscl(File),setup.
cons_sug(1,Sit,File):-concat(File,['c:',rev,Sit,'.pro']).
cons_sug(2,Sit,File):-concat(File,['c:',lov,Sit,'.pro']).
run:-exec(storym),check_consult,
    do_sugs.
do_sugs:-
          trysug,do_sugs.
trysug :-    sug(X,Y),retractall((sug(X,_))),
          send_to_pas(Y).
trysug :-   send_to_pas(['No more suggestions.']).
send_to_pas(Mes):-open('f:helpfile.tmp',w),
               name56(N),writef('f:helpfile.tmp',N),
               nlf('f:helpfile.tmp'),
               records_out(Mes),
               close('f:helpfile.tmp'),
               exec('storym').
records_out([]).
records_out([H|T]):-writef('f:helpfile.tmp',H),
               nlf('f:helpfile.tmp'),
               records_out(T).
```

This program:

•runs the Pascal MULTISTORY program (the user-interface).
•when a suggestion is required, MULTISTORY is exited and control. is returned to RUNSUGS.
•on return, the program checks whether this is the first time a suggestion has been asked for. If so, then it consults the relevant story information file. It then Checks what story situation is being

used and consults the appropriate suggestion file (eg. REVONE.PRO contains the story suggestions for a 'revenge' story using situation number 1. There will be 8 different suggestions files for the different situations for the chosen story type). The set-up predicates for this file are then initiated.

- see if a suggestion rule fires. If so save the text in a temporary file.
- return to the MULTISTORY user interface.

Briefly, this program controls the main session. It will start up the MULTISTORY user-interface and handle the switching between the user-interface and the Prolog support system.

The suggestions files contain set-up predicates which test for the story character chosen. In it suggestion rules contain the text of the suggestions to give and conditions which are required before they can be fired. For example the following file REVONE.PRO contains suggestions for story type 'revenge' using situation number 1:

```
setup:-storychar(5),asserta(name56('Jake')).
setup:-asserta(name56('Dylan')).
sug(1,['The story should be retrospective.',
    'i.e. The character is looking back',
    'on a past childhood.']).
sug(2,['Decide whether Jake has a mother',
     'and a father, or just a father.']):-name56('Jake').
sug(3,['Describe life at home.',
    'Was the character badly treated ?']):-att1(X),X>2.
sug(3,['Describe life at home.',
    'Was the character badly treated ?',
    'There is a low health rating.',
    'This could be due to a poor upbringing.']):-att1(X),X<3.
sug(4,['Describe in what ways Dylans',
     'step parents are mean to him.']):-name56('Dylan').
sug(5,['Jake is very disruptive at home.',
   'His father beats him to try and control him.',
    'Jake has a high self-confidence rating.']):-name56('Jake'),
                                att10(X),X>3.
sug(5,['Jake tends to be disruptive at home.']):-name56('Jake').
```

```
sug(6,['Decide whether the father is always working',
      'earning a living and running the house; or',
      'has your character had to do all the house',
      'chores from an early age (more likely if',
      'determination is high).']).
sug(7,['Decide whether Jake s father is a rocker.',
     'This will affect the sort of life that',
      'he led. eg. "parties", motorbike rallies.']):-name56('Jake').
sug(8,['Describe the events leading up to leaving',
      'home. Was it well planned or on the spur ',
      'of the moment.']).
sug(9,['Dylan could run away to find out who his ',
      'real parents are.  His step parents will not',
      'tell him.']):-name56('Dylan').
sug(10,['Describe the parents reaction to ',
      'your character leaving.']).
sug(11,['Describe your characters life in the',
      'big wide world.']).
sug(12,['Your character meets and makes many friends.',
      'High friendliness and attractiveness rating.']):-att5(X),X>2,
                                        att2(Y),Y>2.
sug(13,['Your character finds it easy to get a job.',
      'High intelligence and skilfulness rating.']):-att7(X),X>2,
                                        att11(Y),Y>2.
sug(14,['Your character struggles against set-backs.',
      'High determination rating.']):-att4(X),X>3.
sug(15,['Describe events leading up to your characters',
      'return home. Has he/she made good !']).
sug(16,['Dylan could either find out the truth about',
      'his parents or return home in desperation.']):-
                                        name56('Dylan').
sug(17,['Jake roars into town on his motorbike,',
      'intent on paying his father back']):-name56('Jake').
sug(18,['Why is your characters father to be pitied ?',
      'Possibilities : father himself was beaten when a lad,',
      '              the father has repented his ways,',
      '              the father has many problems.',
       'Your characters  high kindness increases the',
```

```
        'possibility of forgiveness.']):-att9(X),X>2.
sug(18,['Why is your characters father to be pitied ?',
        'Possibilities : father himself was beaten when a lad,',
        '                the father has repented his ways,',
        '                the father has many problems.']).
sug(19,['Think of an ending. What does your character',
        'resolve to do ? Does Dylan find out who his ',
         'real parents are ?']):-name56('Dylan').
sug(19,['Think of an ending. What does your character',
        'resolve to do ?']).
```

An example rule in this knowledge base is:

Rule 12 states IF the attractiveness and friendliness ratings are above 2 THEN use the text of the rule as a suggestion ('Your character meets and makes many friends').

The suggestions in the file are tested in the order that is most applicable to the story. That is, the rules that are tested first have suggestions which are more applicable to the beginning of a story, and the later rules apply to further on in the story, and so on. After a rule has fired, it is removed from the rule-base for the duration of that particular story writing session.

In conclusion, the support system, checks to see what story type and situation is being used. It then loads the appropriate rule base for that type and situation. These rules supply suggestions based on conditions which apply to the story situations and character parameters. As there are 6 story types and 8 situations per type then the complete system will require 48 separate suggestions files/rule bases. Each suggestion file contains roughly 25 rules, so the complete system will have approximately 1200 rules which can give possible story suggestions.

4.4.2. Text analysis support system.

The aim of this work was to produce a text analyser which could feed a description of the story (so far written) into the story suggestions chooser and make an appropriate suggestion based on an analysis of the plot structure (the Expert Writing Model).

The main decisions to take were: what method should be used to parse the text, and what knowledge structures should be used to represent the parsed text? A solution was provided in the form of a parser which was based on software provided by Hinde (1986), using the logic programming language Prolog (Clocksin and Mellish, 1982). See appendix 1 for the listings of the complete system. The natural language processing component of MULTISTORY was a necessary task that was required in order that the overall objectives could be met, and it is not a major component of this research. However, the software did require considerable modification in order that it could be used effectively within the MULTISTORY environment.

Prolog offers an ideal language for parser construction due to its ability to use a built in grammar rule notation as a shorthand for representing natural language context-free grammars. This notation is then translated into ordinary Prolog code by the system, or a special form of Prolog 'consult' can be used.

The declarative clause grammar representation was used to implement a context-free grammar (see Appendix 1, file ENGLISH.PRO for the Prolog grammar rules).

This grammar parses the text non-deterministically by using Prolog's built-in tree searching algorithm. While traversing the parse tree a predicate-calculus representation of the sentence is built up. If a particular grammar rule is unsuccessful, then Prolog back-tracks to the previous node of the tree, un-building any resultant knowledge structures and a new route is taken. This process is repeated until all of the input text is successfully parsed. The parse will fail if all possible routes are traversed unsuccessfully.

The file PRO1.GO is the first file to be consulted by Prolog. This contains the list of files that need to be consulted to run the grammar.

The file OPS.PRO defines the predicate calculus operators used by the file ENGLISH.PRO. These operators need to be declared prior to consulting the grammar file.

PRETHING.PRO is used by the grammar to test a list for whether some of its components are members of a given grammatical class.

CHECKSIM.PRO checks whether the two arguments are subsets of each other. This file also contains the definition for the 'member' predicate (checks whether an atom is a member of a list).

APPEND.PRO will append two lists together. This is an integral part of the parser as it is used to decompose lists into its separate components and feed them into the parser.

TIDY.PRO is used to assert the items in a list into the temporary floating vocabulary.

DCGS.PRO is used to read in Prolog grammar rules and convert to standard rule format (for use by Prolog systems which do not support grammar rule notation).

GENSYM.PRO creates a new atom starting with a root provided and finishing with a unique number. Used to create new, unique atoms.

ENG_AD_M.PRO is used to analyse adverbs formed by words ending in 'y', 'ly' and 'ily'.

ENG_V_M.PRO forms the verb according to its type ie. tense, Person, plural/singular, Gender and Regularity.

OURVAR.PRO is used to mark the subject and object nouns, especially when used with transitive verbs.

ASSERTBT.PRO contains a set of tests which are used by the grammar when asserting temporary structures into the database.

WORDS.PRO contains all the items of the English vocabulary known to the system classified according to their syntactic use.

PARSER.PRO consults a file containing several sentences and processes them using the following grammar. The output from the parser is asserted into the database.

ENGLISH5.PRO contains the grammar rules used to analyse a sentence.

Briefly, the predicate calculus representation used by the parser enabled us to represent the form of arguments in such a way that it is possible to check in a formal way, whether or not they are valid (see Mendelson, 1964). We can express propositions about the world such as a Predicate calculus representation of natural language text, and then make tests on the validity of the logic represented.

The listing TEMPSTRY.PRO in appendix 1 contains a sample story based on the situation and character described in the section above. Each sentence is packaged into a Prolog list ready for analysis by the parser. The listing OUTPUT1.PRO in appendix 1 contains the resulting predicate calculus representation of each sentence after processing by the parser. As can be seen the output of the parser is a simple grammatical representation of each input sentence based around the main verb-phrase with additional character and temporal information. There is no semantic analysis of the sentences ie. no decomposition of the actions described, and no analysis between sentences. However, it does provide a minimal representation of the story text which could be used by an Expert Writing Model to give compositional support.

4.5. Integration of the user interface and support systems

The user interface (written in Pascal) and support systems (written in Prolog) described above were implemented separately on a Research Machines Nimbus micro-computer.This section describes how they were integrated into the MULTISTORY system.

The MS-DOS (MicroSoft Disk Operating System) environment on the Research Machines Nimbus micro-computer is a single-user/single-task operating system without inter-process communication (eg. Remote Procedure Call) or piping facilities (such as found on UNIX machines). To allow the user-interface and support systems to communicate a solution was devised through the use of temporary files and program spawning (see appendix 2 for program listings and top-level design chart).

A separate RAM disk was set up on the Nimbus, which could be used as a fast data area for creating temporary files. The Prolog support system was loaded first. This automatically executed the Pascal program which ran the MULTISTORY user-interface. The user would now stay within the Pascal program during the 'create' and 'word-process' stages. It is only when there was a request for a suggestion that a return would be made to the Prolog support system. Temporary files containing word-processing pointers, story settings and the current story being written, are first created before returning to Prolog. The Prolog support system then consults the temporary files, chooses a suggestion, and then creates a new temporary file containing it. Control is then passed back to the Pascal MULTISTORY environment, the temporary files are read in and the suggestion is displayed on the screen. This process is completely transparent to the user. The screen continues to display the MULTISTORY environment and the user is unaware of the program switching that is occurring underneath.

4.6. Current state of development of MULTISTORY

This chapter has described the development of a system to analyse and support children's story composition. A user interface was developed in Pascal with an underlying rule-base implemented in Prolog.

The aim of the system is to provide an aid to the stimulation of formulative ideas at a first stage, as well as immediate and appropriate responses during the act of composition itself in the form of advice or suggestions to the child writing. The support system attempts to provide a level of response which is directly related to the activity itself as distinct from stylized and rigid stock responses. A need to provide this facility for individual writers has been identified.

At the technical level there are some limitations with the current system design. The use of two separate programming languages does pose some problems for system integration, and a solution has been described above. Surprisingly, using this approach there was not a long delay when switching from the user-interface to the support system. But for practical implementation a more robust method would be required. In addition the use of temporary files for program communication does severely restrict the level of communication between systems. Ideally the user-interface and support systems should be implemented in a common programming environment and this is discussed in the next chapter.

To re-visit the relationship between the MULTISTORY system and the ICAI architecture described in chapter 1. In the MULTISTORY system a student model was constructed by parsing the text of the story being written. An initial simple expert-model was constructed based around a simple rule-based description of events in one story. The need for a more comprehensive expert model has been raised particularly in relation to mapping the student model to a representation of predicted story events and this is discussed in the next chapter.

The prototype MULTISTORY system currently consists of the following components:

•an advanced colour WIMP based environment for story composition. The system supports the creation of new story outlines, and a text editor for story composition.

•Embedded within the text editor is an expert system which attempts to provide suggestions for the story currently being written. The development of the expert system followed two stages. Stage 1 was a simple rule based system which examined the current story outline to decide on a suggestion; and stage 2 was a predicate calculus parser which translated an input story into a grammatical representation. Currently, the output of the parser is not fed into the suggestions chooser.

The development of the MULTISTORY system has provided the framework for assessing the benefits of computer supported creative writing and the basis on which to develop an Expert Writing Model for compositional support.

Chapter 5. A CRITIQUE OF *MULTISTORY*

The previous chapters have described the development of a system called MULTISTORY to support story composition based on an evaluation of Intelligent Computer Assisted Instruction (ICAI) criteria and an analysis of composition support systems. This chapter provides an overview of this research with a discussion of the main issues addressed by MULTISTORY, and some of the new questions it raises and future research ideas.

5.1. Research Overview

Story understanding research in the past has examined various different aspects of text understanding, including sentence level syntax and grammar (Booth, 1983), syntax of stories themselves (Beaugrande & Colby, 1979), semantic components (Schank, 1972), plots (Schank and Abelson, 1977), themes (Dyer, 1981), and character's emotions (Dyer, 1983).

Work has also been done on enabling computers to write stories based on predefined templates using a mixture of canned sequences and functions which can represent varying sequences. Different approaches have been used, including story grammars (Rumelhart, 1980), rules of expressiveness (Dreizin et al, 1978), and the more well known work by Meehan (1981) on the TALE-SPIN program which devised stories based on an initial problem through to the solution by assigning goals to characters in the story and constructing plans by which to solve them.

However, very little has been done in applying story analysis techniques to the problem of supporting the writer in creating stories, and in particular the task of making decisions about the different plot alternatives of a story, whilst it is being written. There is no clear understanding of the usefulness of attempting to provide interactive support for the writer making these decisions.

Previous writing support systems such as the Unix Writer's Workbench (Frase, 1983) and the IBM Epistle system (Heidorn et al, 1982) concentrated on syntactic analysis and support. O'Malley and Sharples (1986) have defined several tools to be included as part of a 'Writers Assistant', based on a cognitive model of the tasks involved in writing. Commercial products aimed at supporting writers such as the Wordbench (Addison-Wesley, 1989) for the IBM PC are now appearing. This package includes a suite of tools including an outliner, spelling checker, thesaurus, a note-taker, a reference tool, a viewer for examining outlines and citations while working on a document, and a brainstorming tool. MULTISTORY is aiming to support the writer in an area which as yet is unexplored:

With the premise that computer support for pre-writing and writing skills will benefit English education the aim of this research is to use appropriate technologies to build an experimental creative writing support system that can assist the writer throughout the writing process. This will provide the foundation for an analysis of the Expert Writing Model for story composition.

The following steps were necessary in order to complete the research:

- to build a story writing support system by applying natural language processing techniques and a rule-based assistant.
- to determine what the components of such a system are.
- to test the usefulness and feasibility of such a system.
- to make recommendations on the requirements and direction of future composition support systems and the Expert Writing Model.

So how have these aims been met?

A story writing support system called MULTISTORY was constructed. The user-interface and story selection phase was implemented in Pascal and Assembler, and a natural language parser and rule-based support system was built using Prolog. The system components were based on the criteria laid out in chapter 1 for so-called 'Intelligent Tutoring Systems', that is: user interface, expert model, student model, and a tutoring strategy.

A 'user-friendly' colour WIMP (Windows Icon Mouse Pointer) user-interface was used to make the system as attractive and as easy to use as possible. This interface was developed using the conventional programming languages Pascal and Assembler, and required a great deal of programming effort (see appendix 2).

A student model was constructed by forcing the user to make decisions about the story type, situation, and main character before writing a story. In addition to this a predicate-calculus type parser was constructed to parse the story text, and the resulting parse would form a detailed student model. The accuracy of the student model

produced by the parser, is dependent on the power of the parser itself, ie. the size of it's lexicon, the sophistication of the grammar used, it's ability to recognise ambiguity, identify pronoun references, and recognise the underlying meaning of the text.

As has been stated, Natural Language Processing is a hard problem. The parser used in MULTISTORY was limited in many ways. It could only recognise simple noun-phrase/verb-phrase combinations, there was limited morphological processing, and the predicate-calculus representation had little semantic value. However the aim of this research was not to build a bullet-proof natural language parser. The parser had only to be good enough to test whether it could be used as part of a story writing support system.

A simple expert model was constructed by devising a rule-base of suggestions which would match the story criteria against the conditions of a series of rules. When a rule fired, then an appropriate suggestion would be given. A more sophisticated expert model is required to properly utilise the the student model obtained from the natural language parser (discussed in the next chapter).

5.2. Research Issues

This work has proved that it is possible to construct a creative story writing support system based on what is becoming generally known as the Hartley-Sleeman four-component model (Ford & Yazdani, 1988) architecture for Intelligent Tutoring Systems.

The natural language processing component has proved to be the weakest link of the system. However, it was a necessary component in order to fulfil the overall research objectives. There has generally been a significant decline in research work into the application of A.I. theories for natural language text understanding. The interest in this area shown in the late 70's and early 80's mainly in the United States, has provided some interesting theories and small scale systems, but failed to furnish any really usable or practicable methodologies for semantic analysis of text. Fortunately, we are now starting to see general purpose Natural Language Processing (NLP) toolkits such as SRI's Core Language Engine (CLE), a system (Moore, 1987) for translating natural language (English) sentences into formal representations of their literal meanings. This system includes a lexical lookup, syntactic parsing, semantic interpretation, and quantifier scoping to resolve ambiguities. The CLE is built around the Prolog programming language and it should allow for the construction of far more robust and usable natural language systems.

The development of the MULTISTORY system has highlighted the need for advanced software tools such as user interface libraries, knowledge engineering toolkits, and rapid prototyping environments. For example, systems such as KEE (™ Intellicorp) can provide a rich rapid-prototyping development environment. Formally these systems were restricted to expensive A.I. workstations, which were impractical for typical schools and colleges. However, with the continued decrease in the price/performance ratio illustrated with the availability of 32-bit and high-performance reduced instruction set (RISC) computers (eg. the Sun SparcStation) they may well point the way to the type of educational computer system we will see in the future.

The four component Hartley-Sleeman model for Intelligent Computer Assisted Instruction does appear to be valid but the distinction between some of the components can become blurred. In software engineering terms it is quite feasible and often more practicable to combine some of the components together rather than design a system from scratch as four distinct modules. For example in the MULTISTORY system part of the student model is formed from the parsed story text, whilst the expert model comprises the semantic analysis of the parse. The long held belief in the ICAI community that the student model is usually the most important in an intelligent tutoring system does seem to hold true, although in this context it is viewed as being the input to the expert model. MULTISTORY stands or falls on the quality of the parser which forms the student model. But as the parsing process itself will involve a large amount of 'expert' knowledge then the student and expert models can be viewed as being very much inter-related.

Early ICAI systems (eg. WEST, GUIDON) formed the student model as a distinct subset of a bigger expert model. MULTISTORY does not make this formal assumption. The expert model in MULTISTORY will need to clearly define what it is that identifies a 'good' writer, whereas the student model will at best only intersect with the expert model, in that it will contain some facets which the expert model will recognise, and a large amount that could be good, bad or indifferent. The expert model should try to describe the cognitive processes and decisions which need to be made in good writing, whereas the student model will be a snapshot of one particular collection of ideas at any one time. The need for a proper cognitive model of writing is one that has already been identified by O'Malley & Sharples (1986).

5.3. New research

What is an expert model of writing? This is a difficult question. A.I. researchers have attempted to apply semantic analysis to story understanding. For example the BORIS program (Lehnert, 1983 and Dyer, 1983) was an attempt to identify the plans and goals of characters in a narrative by modelling their emotions and resulting motives. This program was relatively successful for the very small domain to which it was applied (divorce narratives). If we take this approach a step further, then we could apply it as a form of expert model which identifies a character's plans & goals, and made suggestions for the next piece of narrative based on the underlying motives of the character. This sounds promising, but could lead to stereotypical/boring stories, consisting of a logical sequence of events one after the other. However this is an essential step which needs to be addressed before any real story-writing support can be provided.

The question now should be:

'*assuming that all the natural language problems are solved, including proper semantic analysis of characters plans, goals, themes, motives, emotions, etc (an unattainable goal at present) then what problems are still present for a story writing support system?*'.

The main problem must be in determining how to define an expert writing model which can properly use the rich student model derived from this mythical future text parser/analyser. This expert model would have to encompass knowledge about all aspects of story writing, including: writing styles (eg. spy thriller, romantic, pulp-fiction, intellectual, who-done-it, moralistic stories, etc), and the degree to which the writer wants the text to conform to a particular defined writing style; each character's development within the text; first/second/third person narratives; the level of unpredictability of the plot; the use of techniques such as humour, flash-backs, simile', etc; and so on. This subject is discussed in the next chapter.

Then assuming we can encapsulate a suitable expert writing model, the next problem is how we apply this knowledge in a tutoring

environment ie. the coaching strategy. The MULTISTORY system only gave a suggestion when the student asked for it. This method was used because it was the easiest, the computer configuration would not support the multi-tasking necessary for a more sophisticated coaching strategy, and the system could not be absolutely sure that the suggestions it gave would be timely and appropriate. However, if a future writing support system had a far richer student model, expert model and computer system then it would be feasible to implement more sophisticated tutoring strategies and it would be far more likely that the suggestions would be relevant and useful.

What different tutoring strategies can we envisage? The ideal situation could be that the support system continuously monitored the student's input and gave suggestions in a separate output window whenever the expert model identified a possible decision point, fault, or opportunity. The student could choose whether to use the suggestion or not. However, this could lead to the situation where students only followed the computer's suggestions and did not apply any of their own thoughts and ideas. To overcome this the tutor program would have to monitor how many of its suggestions were being used, and vary the support it gave to encourage the student to contribute more to the story. If the tutor realised that the student was precisely following its suggestions, then the tutor could start to phrase the suggestions in a more socratic style, asking the pupil questions rather than only providing easy solutions. The level of intelligence displayed by the tutor would largely be dependent on the sophistication of the underlying expert writing model.

Self's (1988) argument that ICAI systems should learn at the same time as the student may point the way to future composition support systems. The ability of a future system to improve over time through the use of self-learning/rule-induction algorithms (such as Quinlan's ID3 1979) could provide a better means of representing the student model. The system could learn the style of each individual student by monitoring their work over several different story writing exercises. In this way it could induce common mistakes and tailor the advice to more fully support the students needs.

These are some of the problems that need to be addressed if the natural language bottleneck is overcome. The main research area being to develop a more advanced expert writing model and tutoring strategy, assuming that a suitable natural language component can provide a more detailed student model. Clearly the hardware/software platform used for the current MULTISTORY system is inadequate for the implementation of a future more advanced story writing support system. The following minimum requirements would be necessary:

- Multi-tasking operating system (probably Unix)
- Large screen (about 19")
- Multi-window environment
- Advanced software tools (eg. KEE, HyperCard, NextStep)
- Fast micro-processor (eg. RISC technology)

In addition to developing the Expert and Tutoring models there are several other research areas that are then feasible. The following are some ideas for future research:

A mechanism for teachers to modify and author their own expert writing models to suit a particular teaching session or example.

Additional practical evaluation of this type of computer aided writing support system is needed in a classroom environment.

How could this type of support system be integrated into a complete 'writing' environment, such as the writer's assistant being developed by O'Malley and Sharples et al (1986) which includes 'brainstorming', 'outlining', and other writing tools? A proper cognitive model of 'writing' is needed as the basis of the research (discussed in chapters 6 & 7).

How can the domains of Hypertext (McAleese, 1989) and Interactive Fiction (Howell, 1989) be applied to a story writing support system. The common concept behind both Hypertext

and Interactive Fiction is that the reader can make choices on possible routes through a narrative text, and can choose whether to expand a particular piece of information or change direction altogether. The authoring of Hypertext and Interactive Fiction is still relatively unknown, and the notion of providing 'intelligent' support may prove to be an interesting area.

Collaborative writing of text opens up a whole new set of problems, and is especially relevant in the classroom situation. Research is needed into how we can provide systems which support this type of work. See Rada et al (1989).

An investigation into the use of expert system shells in education. What sort of explanation facilities are needed? How can they be used as part of a coherent teaching strategy. See Valley (1989).

There are probably many other areas of work which could be mentioned, but the above list provides a good starting point. MULTISTORY can be seen to have carried out the necessary first phase of this research. It has highlighted the main problem areas, given us a clearer idea on how such a system might be implemented, and has suggested further areas of research. The next chapter discusses the notion of the 'expert writing model', which has proved to be the main stumbling block for 'intelligent' story writing support systems.

Chapter 6. THE 'EXPERT WRITING MODEL'

The previous chapter critiqued the MULTISTORY system and provided an overview of the research with a discussion of the main issues addressed, and some of the new questions raised.

The main question that has resulted from this work is: How can we define an Expert Writing Model for MULTISTORY which can use the student model derived from the text parser/analyser and provide appropriate support to the writer?

This chapter defines the components of and the architecture for an Expert Writing Model for story composition.

The scope of the Expert Writing Model in this context is concerned with the domain knowledge and heuristics necessary to provide appropriate plot or event/episode based suggestions (ie. advice on the 'well-formedness' of the story) in the MULTISTORY environment previously described. We are not concerned with modelling the expertise for other writing activities or functions (such as correct grammar, writing style, etc) which is adequately covered elsewhere. Neither is this section concerned with the mechanics of the underlying parser, it is assumed that the natural language processing problems can be solved, including the analysis of a character's plans, goals, motives, emotions, etc (an unattainable vision at present).

6.1. The Components of the Expert Writing Model

As described in chapter 3 the domain knowledge component of MULTISTORY should represent the expertise required to write a story. In this context, the expertise should be of the form 'OK, I can see what you have written, why don't you try this.....'. The Expert Writing Model (EWM) will ideally need to recognise what has been written, relate that to its own internal model of what should be contained in a good story for the chosen criteria, and make a suggestion to the writer based upon the difference between the input text and the expert/story model, its so-called domain knowledge. The input to the expert model will be the student model (the parsed story text), and the previously chosen story criteria.

This corresponds to the bug-construction method of student modelling described in chapter 1 where the expert model generates a search tree for the student model derived from its own internal domain knowledge.

For the provision of plot level support it is proposed that the following components should be included in the EWM. It is assumed that a sufficiently powerful text parser can supply the EWM with an appropriate representation of the story (discussed in chapter 3):

- Top-down Story Grammar
- Bottom-up AI planner/simulator

The functionality of the individual components will now be described, followed by a description of the architecture of the EWM.

6.1.1. Top-down Story Grammar

Just as simple sentences can be said to have an internal structure, so too can stories be said to have an internal structure. Rumelhart (1975) was one of the first proponents of a story grammar which could account for many of the salient facts about the structure of simple stories.

The general structure of this grammar consists of the following re-write rules (a * indicates one or more units, and a | indicates mutually exclusive units):

Rule 1: Story -> Setting + Episode
Rule 2: Setting -> (State)*
Rule 3: Episode -> Event + Reaction
Rule 4: Event -> {Episode | Change-of-state | Action | Event + Event}
Rule 5: Reaction -> Internal Response + Overt Response
Rule 6: Internal Response -> {Emotion | Desire}
Rule 7: Overt Response -> {Action | (Attempt)*}
Rule 8: Attempt -> Plan + Application
Rule 9: Application -> (Preaction)* + Action + Consequence
Rule 10: Preaction -> Subgoal + (Attempt)*
Rule 11: Consequence -> {Reaction | Event}

For each syntactic rule there is an associated semantic interpretation:

Rule 1: ALLOW (Setting, Episode)
Rule 2: AND (State, state, ...)
Rule 3: INITIATE (Event, Reaction)
Rule 4: CAUSE(Event^1, Event^2) or ALLOW(Event^1, Event^2)
Rule 5: MOTIVATE(Internal-response, Overt Response)
Rule 6: semantically constrained
Rule 7: THEN(Attempt^1, Attempt^2,.....)
Rule 8: MOTIVATE(Plan, Application)
Rule 9: ALLOW(AND(Preaction, Preaction,...),
{CAUSE | INITIATE | ALLOW} (Action, Consequence))
Rule 10: MOTIVATE [Subgoal, THEN (Attempt,.....)]
Rule 11: semantically constrained

Where the semantic relationships have the following definitions:

AND	conjunction of a number of arguments
ALLOW	relationship between an event which made possible, but which did not directly cause a second event

INITIATE relationship between an external event and the reaction to it

MOTIVATE relationship between an internal response and the actions resulting from that internal response

CAUSE relationship between two events in which the first is the physical cause of the second

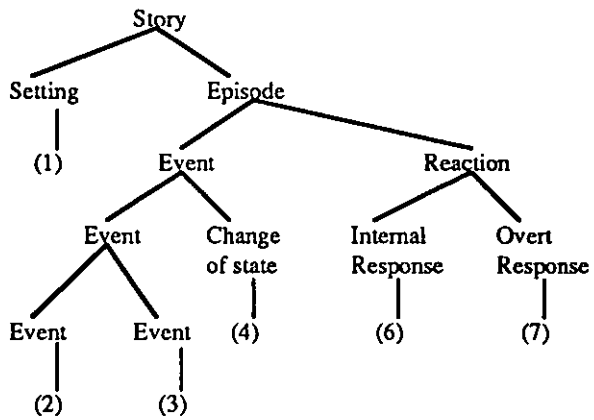
THEN relationship between temporal events

These semantic relationships attempt to provide a set of simple descriptors for the global structure of the story. This is a similar approach to the Conceptual Dependency (CD) theory developed by Schank (1972) and Schank & Abelson (1977), but CD provides a more bottom-up approach to text analysis, whereas Rumelhart's grammar describes the top-down global structure.

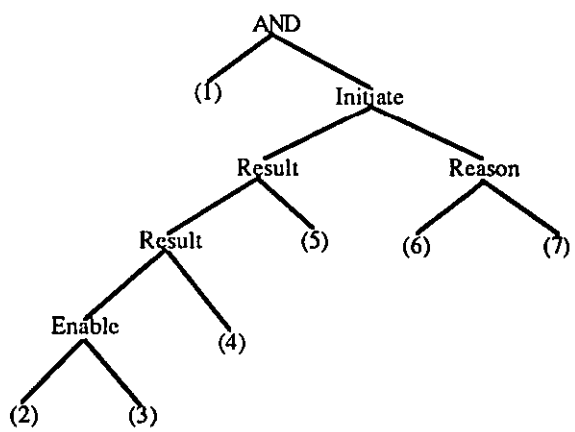
The following example illustrates the application of the above grammar to a simple story (the example assumes that the text can be successfully parsed):

Units of the example story:
(1) Margie was holding tightly to the string of her beautiful new balloon.
(2) Suddenly, a gust of wind caught it
(3) and carried it into a tree.
(4) It hit a branch
(5) and burst.
(6) [sadness] --> inferred from the text
(7) Margie cried and cried.

The syntactic structure derived from the story (based on the story grammar). Notice that action 5 is disregarded in the syntactic structure.



The semantic structure derived from the story (based on the story grammar).



It can be seen that from this grammar it is possible to build a syntactic representation for a story and also a semantic (loose) representation for the relationships between events in the story based on a constrained set of semantic relations or primitives. A deeper level semantic analysis is required for some of the more complex relationships (eg. rules 6 & 11).

This simple grammar was found (Rumelhart, 1975) to be adequate for the analysis of most folk tales and simple fables. In addition, Rumelhart developed a set of summarisation rules which could be applied to the semantic structures built by the grammar to produce adequate summaries of simple stories. However, the grammar does

have difficulty handling more complex multi-protagonist stories, and they are best suited to stories about a single major character striving to reach a single overall goal.

Other general story grammars have been proposed by Thorndyke (1977), Mandler & Johnson (1977), and Stein & Glenn (1979) which take a similar approach to Rumelhart's grammar.

Story grammars are not intended to define the set of stories, nor are they intended to be the sole component of a story comprehender. Rather, their purpose is to serve as one of the many sources of knowledge necessary in story comprehension and, in doing so, to produce a story representation out of story-element representations (ie. the parsed text). They are therefore an important component of the EWM and an example of their possible application will be given later in the chapter.

6.1.2. Bottom-up AI planner/simulator

The previous section described a schema for recognising the top-level global structure from story texts. This approach is different to that taken by Black & Wilensky (1979) (and the typical 'A.I' view) who advocate that to determine the 'well-formedness' of a story presupposes an understanding of the story itself. Since the purpose of the grammatical structure of a story is to aid in understanding the story, there is no reason to determine the structure because we must have understood the story before we can discover the structure in the first place (a 'chicken and egg' situation). This is an important point, in that the story grammar approach pre-supposes a powerful enough parser (both syntactic & semantic) to build an appropriate representation of the story. This parser would be based on a representation of the typical knowledge people use to understand stories. The A.I view is that if you have already understood the story what benefit is there from then applying a set of top-level story grammar rules. However, in actuality the story grammar approach, far from being an alternative to 'investigating the knowledge people use to understand stories', can be used to encode the intended aspects

of that knowledge. ie. it can be viewed as an additional knowledge source (see Frisch & Perlis, 1981, and De Beaugrande & Colby, 1979). This is particularly appropriate when applied to the domain of MULTISTORY (discussed later).

The A.I bottom-up approach to story understanding has focused on the generation of stories by computer, but as will be shown most of these ideas directly map onto the MULTISTORY paradigm. The majority of the A.I work was based on the simple theory that 'a story is about a problem and how it gets solved'. TALE-SPIN (Meehan, 1981) was one of the main systems that was developed to generate stories based on this theory.

TALE-SPIN writes stories by simulating a world, assigning goals to some characters and saying what happens when these goals interact with events in the simulated world. The user must first supply much of the information about the initial state of the world, such as the choice of characters and the relationships between one character and another. From then onwards the story generation is a report of the problem solver. Accordingly, the stories TALE-SPIN produces are essentially accounts of what happens during the course of solving one or more problems (see figure 3 - sample output from TALE-SPIN).

Figure 3. Sample output from TALE-SPIN.

ONCE UPON A TIME GEORGE ANT LIVED NEAR A PATCH OF GROUND. THERE WAS A NEST IN AN ASH TREE. WILMA BIRD LIVED IN THE NEST. THERE WAS SOME WATER IN A RIVER. WILMA KNEW THAT THE WATER WAS IN THE RIVER. GEORGE KNEW THAT THE WATER WAS IN THE RIVER. ONE DAY WILMA WAS VERY THIRSTY. WILMA WANTED TO GET NEAR SOME WATER. WILMA FLEW FROM HER NEST ACROSS THE MEADOW THROUGH A VALLEY TO THE RIVER. WILMA DRANK THE WATER. WILMA WASN'T THIRSTY ANYMORE.

GEORGE WAS VERY THIRSTY. GEORGE WANTED TO GET NEAR SOME WATER. GEORGE WALKED FROM HIS PATCH OF

GROUND ACROSS THE MEADOW THROUGH THE VALLEY TO A RIVER. GEORGE FELL INTO THE WATER. GEORGE WANTED TO GET NEAR THE VALLEY. GEORGE COULDN'T GET NEAR THE VALLEY. GEORGE WANTED TO GET NEAR THE MEADOW. GEORGE COULDN'T GET NEAR THE MEADOW. WILMA WANTED TO GET NEAR GEORGE. WILMA GRABBED GEORGE WITH HER CLAW. WILMA TOOK GEORGE FROM THE RIVER THROUGH THE VALLEY TO THE MEADOW. GEORGE WAS DEVOTED TO WILMA. GEORGE OWED EVERYTHING TO WILMA. WILMA LET GO OF GEORGE. GEORGE FELL TO THE MEADOW. THE END.

The approach taken by TALE-SPIN to generate a story follows:

- Identify a CHARACTER out of a pre-defined set.
- Give that character a PROBLEM out of a pre-defined set.
- Create a MICRO-WORLD out of a pre-defined set.
- Input the above to a problem-solving simulator.
- Either STOP or GOTO the start again.

The problem-solving simulator employed by TALE-SPIN was based on the planning program PAM developed by Wilensky (1981) which itself was based on Schank's Conceptual Dependency (CD) semantic representation for natural language. However, TALE-SPIN could only handle situations where there was only one character solving one problem at a time (as is illustrated in the above example). It is difficult to see how TALE-SPIN could deal with more complicated situations without having to change the theory that the stories are just about problem solving of characters. To do this would require the program to have plans of its own and it would also need to plan the story in advance based on a set of pre-defined criteria. What are these criteria?

Beaugrande and Colby (1979) have formulated a basic set of STORY-TELLING RULES which introduce recursion, failure (in addition to success) of the goal and multi-character situations to the scenario above. These rules follow:

(1) Identify at least one character.
(2) Create a PROBLEM STATE for that CHARACTER.
(3) Identify a GOAL STATE for that CHARACTER.
(4) Initiate a PATHWAY from the PROBLEM STATE leading towards the GOAL STATE.
(5) Block or postpone attainment of the GOAL STATE.
(6) Mark one STATE TRANSITION as a TURNING POINT.
(7) Create a TERMINAL STATE which is clearly marked as MATCHING or NOT MATCHING the GOAL STATE.

This basic rule set was also augmented to account for such character roles as protagonist and antagonist, each having a state-action track. These type of STORY-TELLING RULES allow for some means of setting a stories initial criteria and controlling the planner/simulator that will produce the story. Beaugrande and Colby also considered issues concerned with the structure and 'interestingness' of a story, based on identifying 'non-obvious' conditions which can change an action or event into a goal state. However, a limitation with both of these approaches is that they ignore the intentions of the writer. Other work has continued to expand on the simulation/planning approach developed above, notably Dehn (1981) with the AUTHOR system which attempted to account for the authors intentionality, and Yazdani (1983 & 1986) with the ROALD system which allowed for more than one 'active' agent in the world of stories. These approaches seem promising as they provide a means for extending the capabilities of the 'bottom-up' approach.

Similar to the top-down Story Grammar technique the bottom-up AI planner/simulator approach for story generation is very much dependent on the power of the underlying text parser, but in this case the parser is being used to generate rather than interpret text, and it is more tightly bound with the whole process (ie. the planner can be a significant part of the parsing process itself).

The story generator is an important part of the EWM as it will provide a process for identifying the plans and goals of characters within the story world and also a means of generating possible stories from the initial parameters supplied by MULTISTORY. If

needed, the MULTISTORY pre-writing tasks could be expanded to provide additional character and world information required by the story generator. Again, the story generator can be viewed as one of the many sources of knowledge necessary in story comprehension and in producing a story from the initial story-elements they are an important component of the proposed EWM.

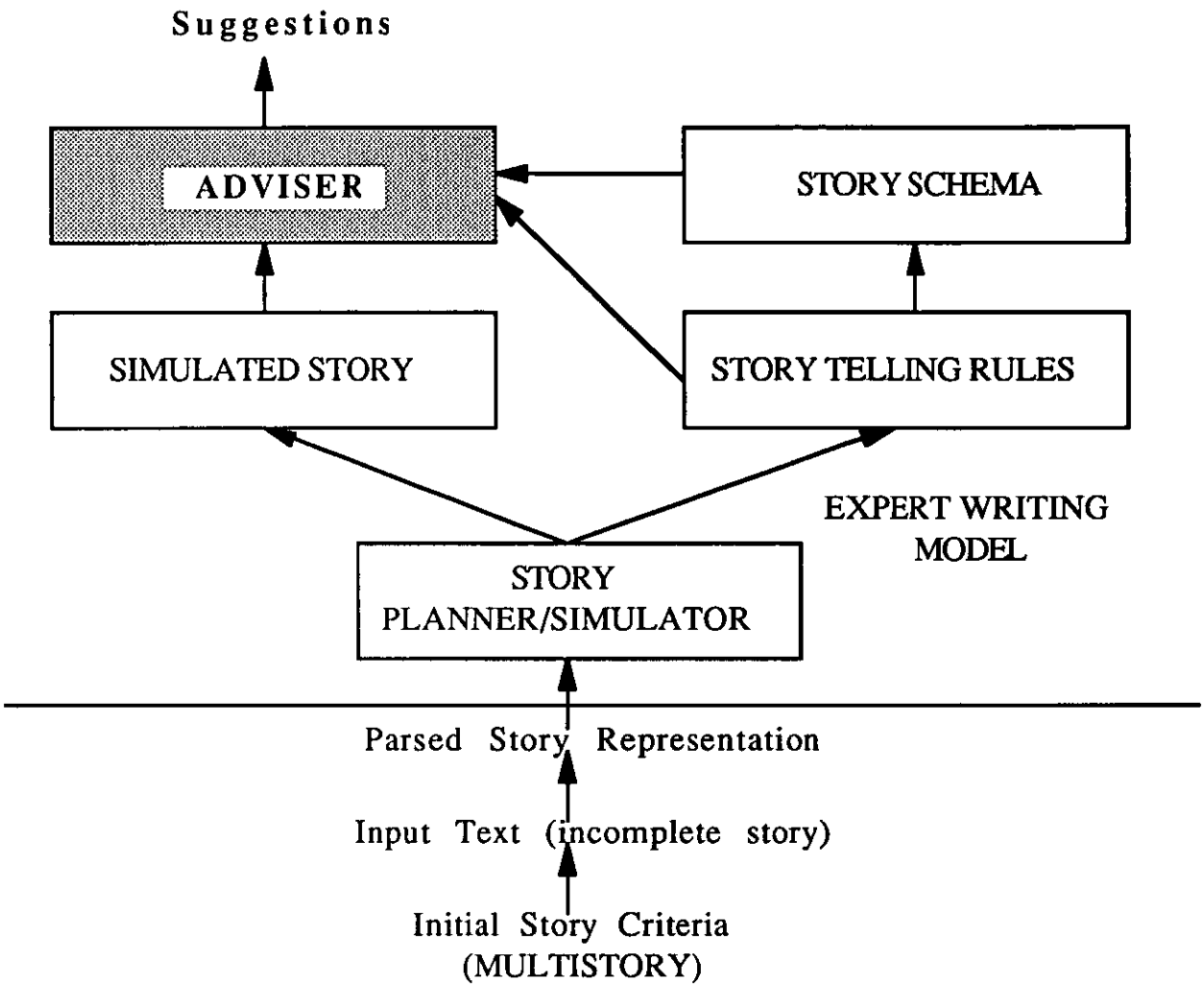
6.2. The EWM Architecture

We can formalise the model of story writing proposed by Meehan (1981) and Beaugrande and Colby (1979) in the following way:

> *A story is an account of goal directed behaviour of a set of characters - their interactions with each other and with the objects of the world.*

The Story grammar (schema) approach is largely reticent about how a schema-based, top-down understander actually gets the structure out of the surface text. The STORY-TELLING RULES developed by Beaugrande and Colby (1979) suggest a solution to this dilemma, especially the noticing of value assignments (Process 4), motivational statements (Process 5), markers of boundaries of events, actions, and states (Process 6), and indicators of time, place, and resources (Process 7). The planning/simulation method of story generation developed by Meehan (1981) provides a means for identifying the different states within the STORY-TELLING RULES (ie. the PROBLEM STATES, GOAL STATES, STATE TRANSITIONS, etc) and the problem-solving PATHWAYS between PROBLEM and GOAL STATES. Also it provides a means of generating a story given an initial set of condition statements (ie. the micro-world). Given these different knowledge sources the following architecture is proposed for the EWM (figure 4):

Figure 4. EWM architecture



The aim of the ADVISER is to determine the well-formedness (or conversely the incompleteness) of the story and to supply an appropriate suggestion to the writer. The strategy employed by the ADVISER is dependent on the Tutoring Strategy employed by MULTISTORY and is discussed elsewhere. However, the ADVISER will need to arbitrate between the knowledge sources contained in the Expert Writing Model (EWM) to choose the most appropriate suggestion.

The initial story criteria will be supplied by MULTISTORY and will be used to set-up the micro-world of the PLANNER/SIMULATOR based on the decisions made by the writer prior to the commencement of writing.

The Input Text will be the current state of the story when the writer asks for advice.

The Parsed Story Representation will be the output from the natural language parser and it will be in an appropriate syntactic and semantic representation for the EWM.

The EWM itself will have the following agents:

STORY PLANNER/SIMULATOR - the planner will determine the actors (main character's) goals, will establish the sub-goals that will lead to the main goal, and will match the actor's actions with the associated plans. This will form a snap-shot of the current story and will be the input into the STORY TELLING RULES. The simulator will also take the established micro-world and generate a story using the processes described above. This will produce the SIMULATED STORY.

SIMULATED STORY - will provide a knowledge source for giving advice to the writer which will represent a possible version of the story given the initial criteria. The SIMULATED STORY agent will match the simulated story against the snap-shot produced by the planner and will determine the current (episodic) state of the story compared to the simulation. This will then enable the agent to select the next event within the simulation as a suggestion for the writer. The content of the simulated story will range from being very close to the actual story being written to completely different. However, it should provide a valuable additional source of ideas for the writer (discussed later).

STORY TELLING RULES - will provide a knowledge source for determining whether the plans and goals identified by the planner meet the guide-lines imposed by the STORY TELLING RULES. These rules will define a general structure for well-formed and 'interesting' narratives which will enable the placement of a character's actions within an episodic framework. The rule sequence will be recursive, so that it may recognise either a minimal story or

only an episode in a longer story. The suggestion given by this agent will depend on the rule that fires within the rule set (there are 7 core rules in Beaugrande and Colby's STORY TELLING RULES) and the instantiation of the characters PROBLEM and GOAL STATES from the planning agent. It may be appropriate to have different sets of STORY TELLING RULES for the different possible story types, so for example, a particular rule set could specialise in REVENGE type stories and another set in ADVENTURE stories (see next section).

STORY SCHEMA - the STORY TELLING RULES will provide advice based on identified episodes within the story text. As discussed above the STORY TELLING rules can also be used to recognise the components of the STORY SCHEMA such as value assignments, motivational statements, markers of boundaries of events, actions, and states, and indicators of time, place, and resources. This information will serve as the input to the STORY SCHEMA agent which will be used to construct partial syntactic and semantic structures for the story which can be used to give advice on the global structure of the story. These suggestions will be based on the level of incompleteness of the story as described by the syntactic and semantic structures (ie. the non-recognition of a sub-structure within the schema can be used to generate a suggestion for the writer).

ADVISER - this agent simply collates the suggestions from the other agents and passes them onto the MULTISTORY tutoring component. It may be appropriate to prioritise the suggestions from the other agents according to some criteria, but this is difficult to specify at present. Also the functionality of the agent will depend to a large extent on the tutoring strategy employed by MULTISTORY.

The architecture that has been proposed for the EWM allows for independent cooperating knowledge sources each working on a different representation of the text. The advantage of this organisation are those generally associated with the modularisation of knowledge, although a full blackboard style system (as used by Erman & Lesser, 1980) would not be appropriate as there is direct communication between the individual knowledge sources.

It is possible that other knowledge sources would want to be included in this architecture, such as an agent to identify affect (emotions) in narratives such as proposed by Dyer (1983), or an agent to apply stereotypical scripts which can be used to predict events in story understanding as proposed by Schank and Abelson (1977). Careful attention would be needed to ensure that additional knowledge sources did not require changing the underlying theory of the EWM that stipulates that a story is an account of goal directed behaviour of a set of characters and their interactions with each other and with the objects of the world.

6.3. An Example

So far we have discussed the theoretical framework for an Expert Writing Model (EWM) to provide appropriate plot or event/episode based suggestions (ie. advice on the 'well-formedness' of the story) in the MULTISTORY environment. This EWM consists of a variety of cooperating knowledge sources consisting mainly of a top-down story schema and a bottom-up planner/simulator with story telling rules. An example of how these knowledge structures could be applied to a fictional story situation will now be given.

This example will use the following components based on the above EWM architecture (the story criteria and characterisation information is taken directly from the MULTISTORY database):

Initial Story Criteria:
STORY SITUATION - REVENGE - 'A child often beaten by a rather unloving dad, leaves home as soon as he can. Sometime later he returns to his home town intending to pay his father back for his years of suffering. But he finds the father more to be pitied than blamed'.

STORY MAIN-CHARACTER - DYLAN - 'Male, aged 17. An orphan brought up by mean foster parents. Like many in his position he wants to know who his real parents are and is prepared to spend effort locating them'.

STORY CHARACTER ATTRIBUTES - see chapter 4 for a description of how the character attributes are selected.

Example input text (ie. the current state of the story):
'Dylan was very badly treated as a child. His step-father beat him frequently. Dylan also had to cook all the meals and clean the house. He promised himself that one day he would pay his father back for his torment and he began to formulate a plan. When Dylan was 15 years old he decided to run away from home. One night he sneaked past his drunken father and ran from the house as fast as his two legs would carry him. For 5 years he stayed away....'.

Parsed representation:
We assume that the parser can produce a representation suitable for the the following EWM agents.

STORY PLANNER/SIMULATOR:
The simulator will generate a story given the above criteria (micro-world) and any other necessary information. This will produce the SIMULATED STORY. As this will be generated in real-time an example cannot be given, but it will be based on the single-character goal-driven story theory stipulated above. Using the parser output the planner will determine the actors (main character's) goals, will establish the sub-goals that will lead to the main goal, and will match the actor's actions with the associated plans. This will form a snap-shot of the current story and will be the input into the STORY TELLING RULES.

SIMULATED STORY:
This will provide a knowledge source for giving advice to the writer which will represent a possible version of the story given the initial criteria. Again, as this will be generated in real-time an example cannot be given, however the use of the simulation will be very similar to the rule-base described in chapter 4 for the simple rule-based support system component of MULTISTORY. The SIMULATED STORY agent will match the simulated story against the snap-shot produced by the planner and will determine the current (episodic) state of the story compared to the simulation. This will then enable the agent to select the next event within the simulation as a suggestion for the writer. Using the following example of the first few rules from the simple rule-base (chapter 4):

```
sug(1,['The story should be retrospective.',
    'i.e. The character is looking back',
    'on a past childhood.']).
sug(2,['Decide whether Jake has a mother',
     'and a father, or just a father.']):-name56('Jake').
sug(3,['Describe life at home.',
    'Was the character badly treated ?']):-att1(X),X>2.
```

```
sug(3,['Describe life at home.',
      'Was the character badly treated ?',
      'There is a low health rating.',
      'This could be due to a poor upbringing.']):-att1(X),X<3.
sug(4,['Describe in what ways Dylans',
       'step parents are mean to him.']):-name56('Dylan').
sug(5,['Jake is very disruptive at home.',
     'His father beats him to try and control him.',
     'Jake has a high self-confidence rating.']):-name56('Jake'),
                                    att10(X),X>3.
sug(5,['Jake tends to be disruptive at home.']):-name56('Jake').
sug(6,['Decide whether the father is always working',
       'earning a living and running the house; or',
      'has your character had to do all the house',
      'chores from an early age (more likely if',
      'determination is high).']).
....
.... etc
```

The agent will be given the current state or episode from the planning agent which it can then use to select the correct suggestion rule from the above rule-base (for example, suggestion 5 may be selected). There may indeed be scope for the simulator to contain a series of different simulations based on different initial criteria and also other hard-coded scripted templates from which it could select the most appropriate suggestion.

STORY-TELLING RULES:

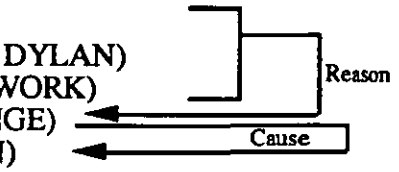
The suggestion given by this agent will depend on the rule that fires within the rule set and the instantiation of the characters PROBLEM and GOAL STATES from the planning agent. Given the above story criteria and the expected output from the planning agent we could expect the STORY-TELLING RULES to generate the following episodic information:

EPISODE 1:

MAIN CHARACTERS: DYLAN, STEP-FATHER

PROBLEM: BAD TREATMENT
GOAL: FORMULATE-PLAN

(1) STATE (DYLAN, CHILD)
(2) STATE (DYLAN, BADLY-TREATED)
(3) STATE (BEATEN-BY STEP-FATHER, DYLAN)
(4) STATE (DYLAN, DOES-ALL-HOUSEWORK)
(5) ACTION (PROMISE, DYLAN, REVENGE)
(6) STATE (DYLAN, FORMULATE-PLAN)

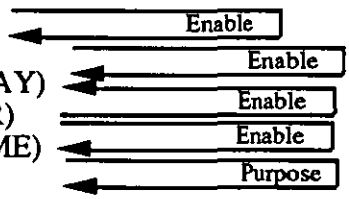


EPISODE 2:

MAIN CHARACTERS: DYLAN, STEP-FATHER

PROBLEM: CARRY-OUT PLAN
GOAL: COMPLETE PLAN

(7) ACTION (DYLAN, GETTING OLDER)
(8) STATE(DYLAN, 15 YEARS OLD)
(9) ACTION (DECIDE, DYLAN, RUN-AWAY)
(10) STATE (GET-DRUNK, STEPFATHER)
(11) ACTION (RUN-AWAY, DYLAN, HOME)
(12) STATE (DYLAN, LEFT HOME)



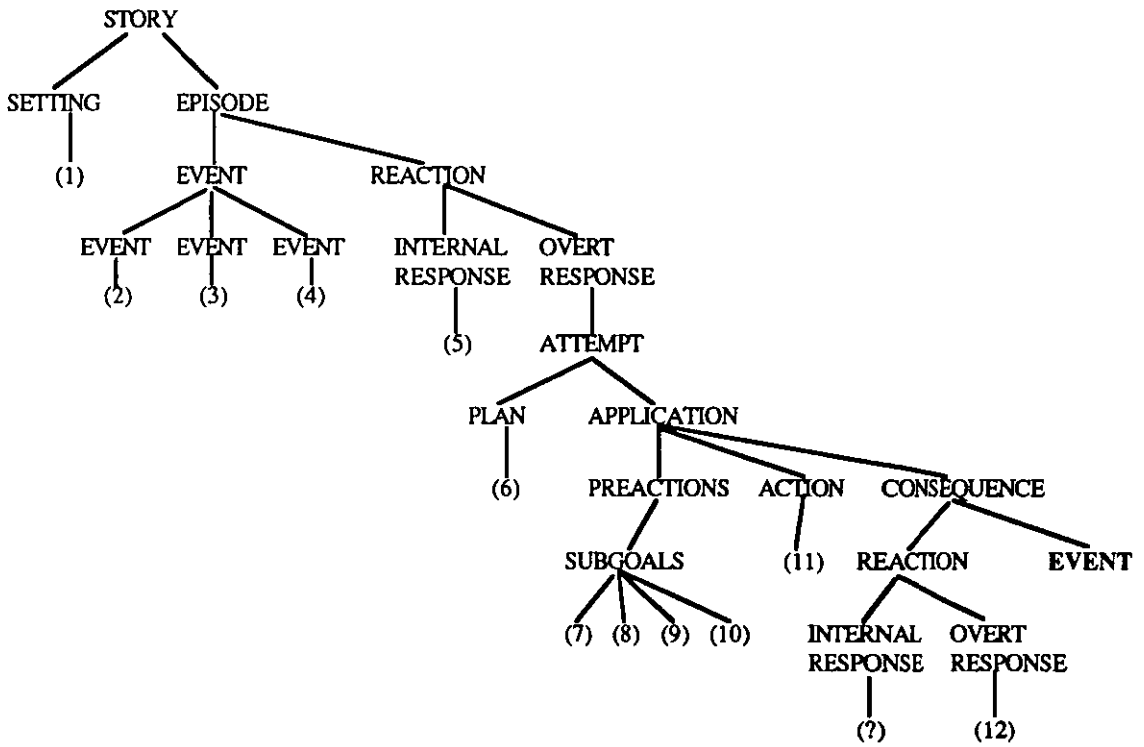
The first operation which should be performed is to divide the story into EPISODES by noticing when the set of story-telling rules had run one cycle, eg. from a problem state to a turning point followed by some terminal state. The above text has 2 episodes: an initial scene setting episode, and a problem to goal sequence with a leaving-home problem. There are clear MOTIVATIONAL STATEMENTS (as would be expected in a revenge story) with the bad treatment of Dylan motivating his desire for revenge. The agent will now be aware that 2 episodes have occurred in the story so far and that there is an unresolved goal state which is to carry out the plan of revenge (in the second episode). The agent will now be in a position to provide a suggestion based on the above extrapolated episodic structure, or indeed based on the lack of a proper causal link between identified states and actions within the structure.

STORY SCHEMA:

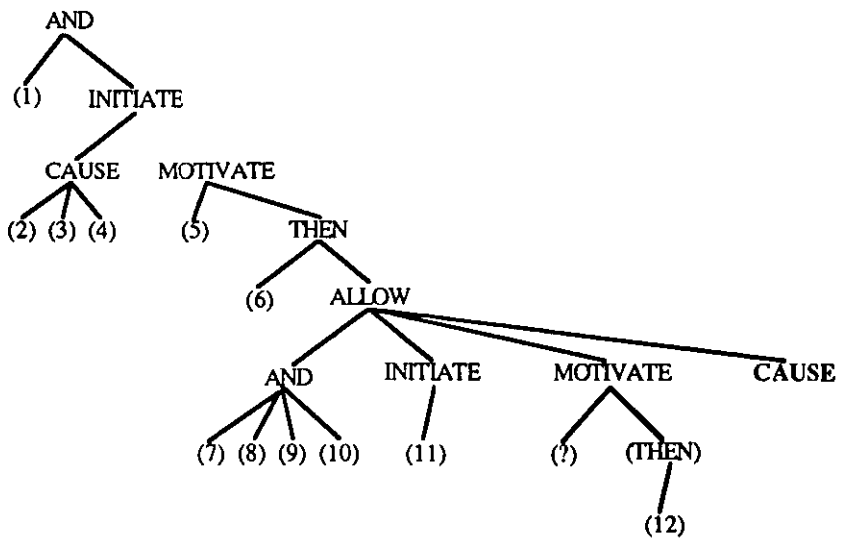
The noticing of value assignments, motivational statements, markers of boundaries of events, actions, and states, and indicators

of time, place, and resources by the STORY TELLING RULES will provide a structure for the STORY SCHEMA. Given the example story texts and the episodic units derived above, the following syntactic and semantic structures can be built from the schema rules (the leaf-node numbers refer to the events identified by the STORY-TELLING-RULES):

Syntactic structure of the story:



Semantic structure of the story:



The semantic structure appears to add little extra information to the syntactic tree, this is probably due to the fact that it is a simple re-write from the syntactic representation. From its global perspective of the story, the schema picks out the fact that there is very little 'setting' to the story (just one statement), also that there is a planning cycle in the story with associated pre-events and consequences. The schema was unable to fill the slot for the character's 'internal response' to the fact that he ran away from home. This could provide a valuable clue to selecting an appropriate suggestion for the writer. Both trees have a non-terminal node indicating that the story is unfinished (ie. a planning sequence is not terminated), and this will be where the remaining story structures will be built.

6.4. Summary

This chapter has provided an overview of an Expert Writing Model (EWM) based on a Top-down Story Grammar, a Bottom-up AI planner/simulator, and an Adviser. The structure of each of these components has been described followed by an example of how they could be applyed to a story in the context of the MULTISTORY system. The EWM does admittedly have some drawbacks, most noticeably the fact that it uses a fairly simple model for the type of stories it will recognise, ie: non-complex, single character/goal stories. However, it does provide a reasonably flexible architecture for adding additional knowledge sources (agents) to the EWM.

Chapter 7 will discuss the framework in which the the EWM can be placed in relation to other writing support environments.

Chapter 7. THE FRAMEWORK FOR AN EXPERT WRITING MODEL

The scope of the Expert Writing Model (EWM) defined in the previous chapter is concerned with the domain knowledge and heuristics necessary to provide appropriate plot or event/episode based suggestions (ie. advice on the 'well-formedness' of the story) in a story compositional environment. It is not concerned with modelling the expertise for other writing activities or functions (such as correct grammar, writing style, etc) which is adequately covered elsewhere. Neither is it concerned with the mechanics of the underlying parser. It is also assumed that the natural language processing problems can be solved, including the analysis of characters plans, goals, motives, emotions, etc.

As discussed in previous chapters, the possible scope of an Expert Writing Model is very large and this thesis has concentrated on a distinct subset of the whole. Other work, particularly by Sharples et al (1988 & 1990) on the Writer's Assistant project addresses many of the cognitive issues involved when people are creating complex documents. In this project a software tool known as the Writer's Assistant was developed which is based on a cognitive framework for writing. This chapter will discuss the relationship between the Writer's Assistant and the Expert Writing Model and will show that the Expert Writing Model can fit cleanly into the architecture of the Writer's Assistant.

The Writer's Assistant aims to assist the writer throughout the writing process, from the generation and capture of ideas to the production of a connected piece of prose, combining the facilities of a text editor, an 'outliner', an 'ideas processor', and a 'structure editor'. The system will offer the writer three separate, but mutually consistent views of the emerging text, in the form of an ideas net, a string of words and a layout. The writer can move between them all at will and, by altering one view, know that the others will remain in step. Chapter 2 discussed the range of composition support systems, including the Writer's Assistant.

The purpose of the Writer's Assistant is: first, to show the writer the structure of a text and its underlying ideas; second, to allow the writer to create and manipulate both ideas and text, moving amongst different writing strategies; and third, to allow the writer to specify constraints on text. As the text is created the system should attempt to satisfy the constraints and, where it is unable to do so, should present the constraint conflict to the writer.

A simple example of a constraint might be a rule within a sentence object to notify the user of repeated adjacent words. For example (O'Malley & Sharples, 1986):

?word1 ^word1 -> tell_user

The '?word1' matches a single word in the sentence and '^word1' specifies the same adjacent word. When the pattern occurs the 'tell_user' procedure is fired.

Constraint satisfaction is carried out whenever an operation is performed on a view. Some constraints, such as those involving formatting, can be satisfied by the system without involving the writer. Others, such as resource violations (eg. a word cannot be found in the dictionary), or attribute violations (eg. a section contains too many words) do pose the additional problem of deciding whether to interrupt the writer or to automatically resolve the problem. This falls into the domain of the tutoring strategy as discussed in chapter 1 (Intelligent Computer Assisted Instruction).

The constraint handling employed by the Writer's Assistant deliberately concentrates on the surface features of the different textual views. However, O'Malley & Sharples do identify the need to be able to take the linear stream of text and itemise it to represent the underlying meaning (ie. to get below the surface features) but they do not say how this can be done. I propose that the Expert Writing Model described in the previous chapter can provide a means for representing and managing the complex constraints involved with the plot and episodic structure within the Writer's Assistant, and it

can also provide a means of representing the underlying structure of the text from a story telling point of view.

This means that the type of support offered by the Expert Writing Model for story composition could be integrated with the Writer's Assistant through the use of constraint satisfaction. Also the deep-level knowledge structures that are extracted by the Expert Writing Model could be used to identify 'chunks' within the text (eg. episodes, 'ideas') and could also assist the Writer's Assistant in switching between the different views of the text.

The mechanisms employed by the Writer's Assistant to switch between the linear view and the network view are based on applying heuristics to the surface structures within the text (eg. footnotes). It is feasible that the Expert Writing Model would be able to provide a more accurate networked view based on the underlying episodic structure of the text. The episodic framework of a story could then be displayed to the writer as an additional support feature. The suggestions provided by the Expert Writing Model would be handled through the constraint satisfaction component of the Writer's Assistant employing a suitable tutoring/interruption strategy.

In summary, the Expert Writing Model appears to be fully compatible with the architecture of the Writer's Assistant. The deeper level knowledge structures of the Expert Writing Model could be made available to the Writer's assistant through the use of constraint satisfaction, and it could also provide a more powerful mechanism for moving between the different views of the text. In return, the Writer's Assistant would provide the story writer with a powerful environment for all stages throughout the writing process, from the generation and capture of ideas through to the production of a finished story.

In conclusion, this thesis has proposed that inadequate attention has been paid to supporting the writing process, particularly in providing plot-level support for story composition. A case has been made for developing such systems and an experimental writing support system has been described (MULTISTORY). From this work an

initial definition for an Expert Writing Model (EWM) is given, where the EWM is the domain knowledge that describes 'story structure' in terms of episodes in a story and the character's plans and goals. Plot level support can then be given in terms of this domain knowledge. Finally the EWM is put in context of a Writer's Assistant (Sharples et al, 1988 and 1990) which is derived from a cognitive model of the writing process.

The main achievement of this research is a definition of the domain knowledge and heuristics necessary to provide appropriate plot or event/episode based advice for a writing support environment. This is based on a rigourous investigation of the fields of Intelligent Computer Assisted Instruction (ICAI), Composition Support Systems, and the opportunities afforded by various A.I. techniques. A major part of this work has been the development of an experimental system which formed the basis for the definition of the Expert Writing Model itself.

REFERENCES

1. Alderson G & DeWolf M (1985), 'Guide to Effective Screen design'. Computers in the Curriculum.

2.Anderson,J. R, Boyle,C. F & Reiser,B. J (1985), 'Intelligent Tutoring Systems'. Science vol 228, No.4698.

3. Anderson,J. R, Boyle,C. F & Yost,G (1985), 'The geometry tutor', Proc IJCAI, 1985.

4. Barr,A & Clancey,W. J (1982) 'Application-oriented AI research: Education', in 'The Handbook of Artificial Intelligence', Vol II by Barr,A & Feigenbaum,E.A (eds). Pitman.

5. Bates,M and Wilson K (1982) 'ILIAD: Interactive Language Instruction Assistance for the Deaf', (BBN Report No.4771), Cambridge Ma: Bolt Beranek and Newman Inc.

6. Beaugrande R & Colby B(1979) 'Narrative models of action and interaction', Cognitive Science 3.

7. Bereiter, C & Scardamalia M (1982) 'From Conversation to Composition: the Role of Instruction in a Development Process' in 'Advances in Instructional Psychology', vol 2, Glaser R (Ed), Lawrence Erlbaum Associates.

8. Black J B & Wilensky (1979) 'An Evaluation of Story Grammars', Cognitive Science, vol 3.

9. Bolter, J D & Joyce M (1987) 'Hypertext and Creative Writing', Hypertext '87.

10. Booth A W (1983) 'The resolution of ambiguities and the correction of errors in the automatic transcription of palantype', PhD Thesis, Leicester Polytechnic.

11. Brecht B & Jones M (1988), 'Student Models: the genetic graph approach', IJMMS, Vol 28, No 5.

12. Brown,J. S & Burton,R (1975) 'Multiple representations of knowledge for tutorial reasoning' in 'Representation and Understanding' by Bobrow,D & Collins,A, (eds).

13. Brown,J. S (1977) 'Uses of artificial intelligence and advanced computer technology in education' in 'Computer & Communication: Implications for Education', Academic Press.

14. Brown,J. S & Burton,R. R (1978) 'Diagnostic models for procedural bugs in basic mathematical skills'. Cognitive Science, 2.

15. Brown,J. S & Burton,R. R (1982) 'An investigation of computer coaching for informal learning activities' in 'Intelligent Tutoring Systems' by Sleeman and Brown (eds). Academic Press.

16. Bruce, B (1986). 'Information Technologies and Written Expression', Report for the Centre for Educational Research and Innovation.

17. Bumbaca F (1988), 'Intelligent computer assisted instruction: a theoretical framework', IJMMS, Vol 29, No 3.

18. Burns, H L & Culp G H (1980) 'Stimulating Invention in English Composition through Computer Assisted Instruction', Educational Technology August 1980.

19. Burton,R (1976) 'Semantic grammar'. PhD Thesis. University of California, Irvine.

20. Burton,R. R (1982) 'Diagnosing bugs in a simple procedural skill' in 'Intelligent Tutoring Systems' by Sleeman,D & Brown,J. S (eds), Academic Press.

21. Candy,L. (1983) 'A project which investigates using a computer in English language teaching'. CAL News vol. 23.

22. Candy,L and Schoenfeld,D. (1983) The source of 'storymaster'. Wreake Valley College, Syston, Leicester.

23. Candy,L. (1985) 'The computer and the individual child: an investigation by case study'. HCIRU internal report no.6.

24. Candy, L (1985) 'Innovation with Microcomputers: A Strategy for School Based Action in English', MPhil thesis, Leicester Polytechnic (CNAA), 1985.

25. Carbonnell,J (1970) 'AI in CAI: An artificial-intelligence approach to computer assisted instruction' in IEEE Trans. on Man-Machine Systems, vol MMS 11, No.4.

26. Card, S, Moran T, & Newell A (1980), 'Computer Text-Editing: an Information-processing Analysis of a Routine Cognitive Skill', Cognitive Psychology 12, January 1980 32-74.

27. Clancey,W. J (1974) 'Tutoring rules for guiding a case method dialogue'. IJMMS 11.

28. Clancey,W. J & Letsinger,R (1981) 'NEOMYCIN: reconfiguring a rule based expert system for application to teaching'. 7th IJCAI 1981.

29. Clancey,W. J (1987) 'Knowledge Based Tutoring: the Guidon program', the MIT press.

30. Clocksin,W.F and Mellish,C.S. (1981) 'Programming in Prolog'. Springer-Verlag.

31. Colbourn,M. J (1984) 'Expert systems in education'. Canadian Info.Proc.Soc. 'Images of fear, images of hope: expert systems in education'. Calgary.

32. Cummings, B (1988), 'Using computers to teach English composition: what are the results ?', Proc. 5th Int. Conf on Technology & Education, Edinburgh.

33. Daiute, C A (1983) 'The Computer as Stylus and Audience', College Composition and Communication Vol 34, No 2, May 1983.

34. Daiute, C A (1985) 'Writing and Computers', Addison-Wesley,1985.

35. Dehn N (1981) 'Story generation after TALE-SPIN', International Joint Conference on Artificial Intelligence, 8.

36. DeJong, G (1979) 'Prediction and Substitution: A New Approach to Natural Language Processing', Cognitive Science, 3.

37. Dreizin F et al (1978) 'Towards a computerised generation of sacred legends ', technical report no.1, Focus Project.

38. Dyer,M.G. (1981) 'The role of TAUs in narratives' in Proc. 3rd. Annual Conf. of Cognitive Science Society.

39. Dyer M (1983) 'The role of affect in narratives', Cognitive Science, 1983.

40. Edmonds,E.A and Candy,L. (1982) 'A study in the use of a computer as an aid to English teaching'. Int.J.Man-Machine Studies, 16.

41. Edmonds, E. A (1982) 'The man-computer interface: a note on concepts and design', IJMMS, 16.

42. Ennals, R (1983) 'Beginning Micro-Prolog'. Ellis-Horwood.

43. Erman L D & Lesser V R (1980) 'The HEARSAY-II speech understanding system: A tutorial', in Lea W (Ed) 'Trends in speech recognition', Prentice-Hall.

44. Fillmore, C (1968), 'The Case for Case', in Universals in Linguistic Theory, E Bach and R T Harms (Eds), Holt, New York.

45. Flower, L S & Hayes J R (1979) 'A Process Model of Composition', ERIC Report ED 218 661.

46. Ford, L (1984), 'Intelligent Computer Aided Instruction' in 'Artificial Intelligence, Human Effects' by Yadzani,M & Narayanan,N (eds), Ellis Horwood.

47. Ford, L (1987) 'Teaching strategies and tactics in intelligent computer aided instruction', Artificial Intelligence Review,1.

48. Ford, L & Yazdani M (1988) 'Tutoring systems: the state-of-the-art in Great Britain', Expert Systems, November 1988, Vol 5, No 4.

49. Frase, L T (1983) 'The Unix Writer's Workbench Software: Philosophy', The Bell System Technical Journal, Vol 62, No 6.

50. Frisch A M & Perlis D (1981) 'A Re-Evaluation of Story Grammars', Cognitive Science, 5.

51. Gardner,M.R. (1984) 'The use of Prolog in Computer Aided Learning'. BSc(Hons) final year project report, Leicester Polytechnic.

52. Gardner M R, Edmonds E A & Candy L (1985), 'A system to stimulate and advise childrens' writing', MEP project proposal, HCIRU, Leicester Polytechnic.

53. Goldstein,I.P & Carr,B (1977) 'The computer as coach: an athletic paradigm for intellectual education' in Proc.An.Conf.Assoc. for Comp.Mach. , Seattle.

54. Goldstein,I. P (1982) 'The genetic graph: a representation for the evolution of procedural knowledge' in 'Intelligent Tutoring Systems' by Sleeman,D and Brown,J. S (eds), Academic Press.

55. Granny's Garden. 4MAT Educational Software (1984).

56. Grignetti,M. C, Hausmann,G & Gould,L (1975) 'An intelligent on-line assistant and tutor - NLS-SCHOLAR'. Proc.Nat.Comp.Conf. San Diego.

57. Halasz, F G, Moran T P & Trigg R H (1987) 'Notecards in a nutshell', Proc.ACM Conf. on Human Factors in Computer Systems, Toronto, April 1987.

58. Hayes-Roth,F, Waterman,D. A & Lenat,D. B (1983) (eds) 'Building expert systems'. Addison-Wesley.

59. Heidorn, G E (1975) 'Augmented phrase structure grammars', in Theoretical Issues in Natural Language Processing, Nash Webber B & Schank R C (eds), Assoc Comp Linguistics.

60. Heidorn, G E, Jenson K, Miller L A, Byrd R J & Chodorow M S (1982) 'The EPISTLE text-critiquing system', IBM System Journal, Vol 21, No 3.

61. Hertz, R M (1983), 'Problems of Computer-Assisted Instruction in Composition', The Computing Teacher, September 1983.

62. Hinde, C (1986) 'Predicate Calculus Prolog Parser', Loughborough University of Technology.

63. Howell G (1989) 'IF: journal of interactive fiction', Herriot-Watt University.

64. Jansen, C J, Looijmans P J, Pilot A A, Schrauwen D P & Steehouder M F (1986), 'ALEXIS: Computer-assisted feedback on written assignments', Proc. of EURIT 86, First European Conference on Education and Information Technology.

65. Jensen, K & Heidorn G E (1982) 'The Fitted Parse: 100% Parsing Capability in a Syntactic Grammar of English', IBM Research Report RC9729, Yorktown Heights Research Centre.

66. Johns, T (1983) 'Generating Alternatives', in Exploring English with Micro-computers, Chandler D (Ed), Council for Educational Technology, London.

67. Johnson,W. L & Soloway,E (1984) 'Intention-based diagnosis of programming errors'. Proc.Am.Assoc.A.I.

68. Langley,P & Ohlsson,S (1984) 'Automated Cognitive Modelling'. Proc. AAAI 84.

69. Lebowitz, M (1983) 'Memory Based Parsing', Artificial Intelligence, 21.

70. Lehnert, W G (1983) 'BORIS - An Experiment in In-Depth Understanding of Narratives', Artificial Intelligence, 20.

71. Levin, J A, Boruta M J & Vasconcellos (1983) 'Microcomputer-based Environments for Writing' in Classroom Computers and Cognitive Science by Wilkinson A C (ed), Academic Press.

72. Levy,A. H (1983), 'Experiences with PLATO in medical education' in 'Meeting the challenge: Infomatics and Medical Education' by Page's,J. C, Levy,A. H, Gre'my,F & Anderson,J (eds). Elsevier-Science.

73. MacDonald, N H (1983) 'The Unix Writer's Workbench Software: Rationale and Design', The Bell System Technical Journal, Vol 62, No 6.

74. McAleese R (1989) 'Hypertext: theory into practice', Intellect Ltd.

75. Mandler J M & Johnson N S (1977) 'Remembrance of things parsed: Story structure and recall', Cognitive Psychology.

76. Marcus, S (1988), 'Designing word processor based writing activities', Proc. 5th Int. Conf. on Technology & Education, Edinburgh.

77. Meehan J (1976) 'The metanovel: writing stories by computer', PhD thesis, Yale University.

78. Meehan J R (1981) 'TALE-SPIN', in 'Inside Computer Understanding', by R C Schank & C K Riesbeck (Eds), Erlbaum.

79. Mendelson E (1964) 'Introduction to Mathematical Logic', Van Nostrand Reinhold.

80. Miller,L et al. (1981) 'Text-critiquing with the Epistle system: an authors aid to better syntax' in AFIPS Conf. Proc., Vol 50, Arlington, Va, 649-655.

81. Minsky,M (1975) 'A framework for representing knowledge' in 'The psychology of computer vision' by Winston,P (ed). McGraw-Hill.

82. Moore, J. L & Sleeman D (1988) 'Enhancing PIXIE's tutoring capabilities', IJMMS, Vol 28, No 6.

83. Moore R (1987) 'The SRI Core Language Engine Project: Overview', in Natural Language Processing, Unification and Grammar Formalisms, Proc of Alvey/SERC sponsored workshop, University of Stirling, 1987.

84. Newman, R (1989) 'Writer's Wordbench', Personal Computer World, April 1989.

85. O'Malley C & Sharples M (1986) 'Tools for management and support of multiple constraints in a writer's assistant', HCI 86.

86. O'Shea,T (1982) 'A self-improving quadratic tutor' in 'Intelligent Tutoring Systems' by Sleeman and Brown (eds), Academic Press.

87. O'Shea,T & Self,J (1983), 'Learning and teaching with computers'. Harvester.

88. Papert, S (1980) 'Mindstorms: Children, computers and powerful ideas', Harvester Press.

89. Pemberton, I (1988) 'Using HyperCard to test the design of the Writer's Assistant', Cognitive Science Research Paper, School of Cognitive and Computing Sciences, University of Sussex, September 1988.

90. Petersen, B, Selfe C, & Wahlstrom B (1984) 'Computer-Assisted Instruction and the Writing Process: Questions for Research and Evaluation', College Composition and Communication, Vol 35, No 1.

91. Quillian,R (1968) 'Semantic Memory' in 'Semantic information processing' by Minsky,M (ed), MIT Press.

92. Quinlan J R (1979) 'Discovering rules by induction from large collections of examples', in D Michie (ed) 'Expert systems in the micro-electronic age', Edinburgh University Press.

93. Rada, R & Keith B (1988) 'Collaborative writing of text and hypertext', Project MUCH, Report CS-MUCH-3-88, Department of Computer Science, University of Liverpool.

94. Rada, R (1988) 'Writing and reading hypertext: an overview', Project MUCH, Report CS-MUCH-2-88, Department of Computer Science, University of Liverpool.

95. Rada, R (1988) 'Guidelines for multiple users creating hypertext: SQL and HyperCard experiments', Project MUCH, Report CS-MUCH-1-88, Department of Computer Science, University of Liverpool.

96. Rada R et al (1989) 'MUCH project: multiple users creating hyperdocuments', Liverpool University.

97. Raphael,B (1968) 'A computer program for semantic information retrieval' in 'Semantic information processing' by Minsky,M (ed), MIT Press.

98. Readability Plus. Scandinavian PC Systems.

99. Reiser, B. J,Anderson J. R and Farrell (1985) 'Dynamic student modelling in an intelligent tutor for Lisp programming', IJCAI '85.

100. Rich,E (1983) 'Part II - Knowledge representation in artificial intelligence' in 'Artificial Intelligence'. McGraw-Hill.

101. Rodrigues, R J & Rodrigues D W (1984) 'Computer-based Invention: Its Place and Potential' College Composition and Communication, Vol 35, No 1, Feb 1984.

102. Rumelhart,D.E. (1975) 'Notes on a schema for stories' in Representation and Understanding, Studies in Cognitive Science, D G Bobrow and A Collins (Eds), Academic Press.

103. Sacerdoti, (1977) 'A structure for plans and behaviour' in 'The Artificial Intelligence Series. Elsevier North-Holland.

104. Schank,R. (1972) 'Conceptual Dependency: A theory of Natural Language Understanding'. Cognitive Psychology, 3, 552-631.

105. Schank, R C, Goldman N, Rieger C J, & Riesbeck C (1973) 'MARGIE: Memory Analysis, Response Generation and Inference in English', Proc. 3rd Int.Joint. Conf. on Artificial Intelligence.

106. Schank,R. C & Rieger,C. J (1974) 'Inference and computer understanding of Natural Language'. Artificial Intelligence 5.

107. Schank,R. C & Abelson,R. P (1977) 'Scripts, plans, goals and understanding'. Erlbaum.

108. Schank, R C & Riesbeck C K (1981) 'Inside Computer Understanding', Lawrence Erlbaum Associates.

109. Schwartz, H (1982) 'A computer program for invention and feedback', Conf.on College Composition and Communication, San Fransisco, March 1982.

110. Self,J (1985) 'A perspective on Intelligent Computer Assisted Learning'. Journal of Computer Assisted Learning.

111. Self, J (1988) 'Knowledge, belief and user modelling', in Artificial Intelligence III: methodology, systems, applications. T O'shea & V Sgurev (Eds), North-Holland.

112. SERC/DoI, (1983), 'Intelligent systems: a brief overview', for SERC study of Architectures of IKBS Workshop 3, 16-17 March 1983.

113. Sharples, M (1985). 'Cognition, Computers and Creative Writing', Ellis Horwood.

114. Sharples, M & O'Malley C (1986) 'Tools for management and support of multiple constraints in a writer's assistant', HCI '86.

115. Sharples, M, Goodlet J & Pemberton L (1988) 'Developing a writer's assistant', draft of paper for Proc. 1st Conf. on Computers and Writing, Sheffield.

116. Sharples M, Pemberton L & Goodlet J (1990) 'Writer's Assistant Project - Research Aims', University of Sussex, School of Cognitive and Computing Sciences, internal report.

117. Shortliffe,E.H (1976) 'Computer based medical consultations: MYCIN', Elsevier.

118. Sleeman,D & Smith,M. J (1981) 'Modelling student's problem solving', Artificial Intelligence, 16.

119. Sleeman,D & Brown, J.S (1982), 'Intelligent Tutoring Systems'. Academic Press.

120. Smith, J B, Weiss S F & Ferguson G J (1988), 'A hypertext writing environment and its cognitive basis', Hypertext '87, March 1988.

121. Soloway,E.M, Woolf,B, Rubin,E & Barth,P (1981) 'MENO-II: an intelligent tutoring system for novice programmers'. 7th IJCAI 81.

122. Stein N L & Glenn C G (1979) 'An analysis of story comprehension in elementary school children', in 'New directions in discourse processing', R Freedle (Ed), Norwood.

123. Stevens,A.L & Collins (1977) 'The goal structure of a socratic tutor'. Proc. 1977 An.Conf.Assoc for Comp.Mach, Seattle.

124. Thorndyke P W (1977) 'Cognitive structures in comprehension and memory of narrative discourse', Cognitive Psychology.

125. Valley, K (1989) 'Realising the potential of expert system shells in education', DAI Research Paper No. 432, Dept of AI, Univ of Edinburgh, 1989.

126. Van der Geest, T (1986) 'Teaching writing skills with computers: the development of a writing aid for secondary education', Proc. of EURIT 86, First European Conference on Education and Information Technology.

127. VanLehn,K (1983) 'Human procedural skill acquisition: theory, model and psychological validation'. Proc.Am.Assoc.A.I, Los Altos, Ca.

128. VanLehn, K & Soloway E (1985) 'AI and education'. Conference Tutorial No.10, IJCAI 85.

129. VanLehn, K (1987) 'Learning one sub-procedure per lesson', Artificial Intelligence, 31.

130. Walsh, T (1988) 'PLATO: Predicate Logic Advisory TOol', Proc. 5th Int. Conf. on Technology and Education, Volume 1.

131. Waltz, D L (1978) 'An English Language Question Answering System for a Large Relational Database', Communications of ACM 21.

132. Watson, D (1986) 'Generating Language Learning with CAL', Comput. Educ., Vol 10, No 1, pp181-187.

133. Wexler,J.D (1970), 'Information networks in generative computer-assisted instruction'. IEEE Transactions on Man-Machine Systems, vol MMS 11, no.4.

134. Wilensky, R (1981), 'PAM', in Inside Computer Understanding, R C Schank and C K Riesbeck (Eds), Erlbaum.

135. Wilks, Y (1975) 'Preference Semantics', in Formal Semantics of Natural Language, E L Keenan (Ed), Cambridge University Press.

136. Wilson, N (1986) 'Designing user interfaces for educational software', in Proc. EURIT86 'Developments in educational software and course-ware', Moonen J & Plomp T (Eds), Pergamon Press.

137. Woodruff, E, Bereiter C & Scardamalia M (1981) 'On the road to Computer Assisted Composition', Journal of Educational Technology Systems, 10, 2.

138. Wordbench (1989) Addison-Wesley software.

139. Yankelovich, N & Meyrowitz N (1985) 'Reading and Writing the Electronic Book' Computer, Vol 18 Pt10.

140. Yazdani M (1983) 'Generating events in a fictional world of stories', Research Report R-113, Department of Computer Science, University of Edinburgh.

141. Yazdani (1986) 'A Process-based Model of Text Generation', 3rd International Language Generation Workshop, Nijmegan, The Netherlands, August.

142. Yazdani, M (1986) 'Intelligent tutoring systems survey', in Artificial Intelligence Review,1.

143. Yazdani M (1988) 'Tutoring systems: the state-of-the-art in Great Britain', Expert Systems, November 1988, Vol 5, No 4.

APPENDIX 1.

Predicate Calculus Prolog Listings and Sample Output.

OPS.PRO

```
/* OPS                          */
/* ---                          */

/* This file defines the type of all      */
/*  operators used during the translation. */
/*-----------------------------------------*/

?-op(255,xfx,:).
?-op(225,xfx,<=>).
?-op(225,xfx,=>).
?-op(200,xfy,&).
/* ?-op(200,xfy,ú). */
/* ?-op(200,fx,$). */
/* ?-op(30,fx,~). */
/* ?-op(15,xfx,^) */
```

OPS1.PRO

```
/* This file defines the type of all      */
/*  operators used during the translation. */
/*-----------------------------------------*/

?-op(255,xfx,':').
?-op(225,xfx,'<=>').
?-op(225,xfx,'=>').
?-op(200,xfy,'@').
/* ?-op(200,xfy,'ú'). */
/* ?-op(200,fx,'$'). */
/* ?-op(30,fx,'~'). */
/* ?-op(15,xfx,'^') */
```

PRETHING.PRO

```
/*   PRE_THING                    */
/*    ---------                   */

pre_thing(Fname,[[X] | Args2]):-
      var(X),
      floating_vocab(X),
       Fun=..[Fname | [X | Args2]],
      Fun.
pre_thing(Fname,[[X | T] | Args2]):-
      not(var(X)),
       Fun=..[Fname | [X | Args2]],
      Fun.
pre_thing(Fname,[[X | T] | Args2]):-
      not(var(X)),
       pre_thing(Fname,[T | Args2]).
```

CHECKSIM.PRO

```
/*   CHECKSIM                              */
/*    --------                             */
/*     checks whether the two arguments are      */
/*     subsets of each other.                */
/*-----------------------------------------------*/

checksim(P,P):-
      !.

checksim(P,Q):-
      (atomic(P),
      symbol(Q)),
      !.

checksim([P],Q):-
      member(P,Q),
      !.
```

```
checksim(P,[Q]):-
      member(Q,P),
      !.

checksim(P,Q):-
      P=..[Pf|Pargs],
      Q=..[Pf|Qargs],
       checksimargs(Pargs,Qargs).

checksimargs([],[]):-
      !.

checksimargs([Parg|Pargs],[Qarg|Qargs]):-
       checksim(Parg,Qarg),
       checksimargs(Pargs,Qargs).

member(X,[]):-
      !,
      fail.

member(X,[X|Y]):-
      !.

member(X,[Y|Z]):-
      member(X,Z).
```

CHECKSM1.PRO

```
/*   CHECKSIM                              */
/*    --------                             */
/*     checks whether the two arguments are      */
/*     subsets of each other.                */
/*-----------------------------------------------*/

checksim(P,P):-
      !.
```

```
checksim(P,Q):-
      (atomic(P),
      symbol(Q)),
      !.

checksim([P],Q):-
       member(P,Q),
      !.

checksim(P,[Q]):-
       member(Q,P),
      !.

checksim(P,Q):-
       P=..[Pf|Pargs],
       Q=..[Pf|Qargs],
        checksimargs(Pargs,Qargs).

checksimargs([],[]):-
      !.

checksimargs([Parg|Pargs],[Qarg|Qargs]):-
        checksim(Parg,Qarg),
        checksimargs(Pargs,Qargs).
```

APPEND.PRO

```
append([],L,L).
append([X|L1],L2,[X|L3]):-append(L1,L2,L3).
```

BEGINSV.PRO

```
beginsv(X):-
       name(X,List1),
       vowel(V),
        name(V,Vint),
       append(Vint,Z,List1).
vowel(a).
```

```
vowel(e).
vowel(i).
vowel(o).
vowel(u).
al(je,j).
al(le,l).
al(H,H).
```

TIDY.PRO

```
/*  TIDY                    */
/*  ----                    */

tidy([]) :-
        !.
tidy([X | Y]):-
        assert(floating_vocab(X)),
        tidy(Y).
```

DCGS.PRO

```
/* Reading in Prolog grammmar rules
```

Main predicates provided:

g X — (operator) Carries out a 'consult' of a file that may contain grammar rules, converting any grammar rules into normal clauses

re_g X — As above, but with a 'reconsult' instead of a 'consult'

It is assumed that grammar rules use the infix operator --> to separate the LHS from the RHS. It is also assumed that calls to predicates that are not true 'non-terminals' are enclosed within curly brackets {...}. Conjunctions and disjunctions within curly brackets must be signalled by an extra level of brackets, eg. {(a,b)}. Also, spaces must separate curly brackets from

```
   symbol characters such as '.'.
   These last two details differ from the standard syntax of
   grammar rules used in Dec10 Prolog.
*/


/* Consult a file, converting grammar rules as necessary */

?- op(150,fx,g).
?- op(150,fx,re_g).
?- op(251,fx,{).
?- op(250,xf,}).
?- op(255,xfx,-->).


g X :-
    '$gread'(+,X).

re_g X :-
    retractall('$done'(_)),
    '$gread'(-,X),
    retractall('$done'(_)).

'$gread'(S,X) :-
   seeing(F), see(X),
   repeat, read(T),
    '$gproc'(S,T), !, seen,
    write('Read from '), write(X),
   nl, see(F).

'$gproc'(_,end_of_file) :- !.
'$gproc'(S,A-->B) :- !, '$expand'(A,B,A1,B1), !, '$gass'(S,(A1:-B1)),
fail.
'$gproc'(_,?-Z) :- !, call(Z), !, fail.
'$gproc'(S,L) :- '$gass'(S,L), fail.

    '$gass'(+,L) :- !, assertz(L).
    '$gass'(-,(A:-B)) :- !, '$hddo'(A), assertz(A:-B).
```

```
    '$gass'(-,A) :- '$hddo'(A), assertz(A).

    '$hddo'(A) :- '$done'(A), !.
    '$hddo'(A) :- functor(A,F,N),
      functor(D,F,N),
      asserta('$done'(D)),
      retractall(D).

/* Expand a grammar rule */

'$expand'(P0,Q0,P,Q) :-
       '$dcglhs'(P0,S0,S,P), '$dcgrhs'(Q0,S0,S,Q1),
       '$flatconj'(Q1,Q).

'$dcglhs'((NT,Ts),S0,S,P) :- !,
    nonvar(NT),
    '$islist'(Ts),
    '$tag'(NT,S0,S1,P),
    '$append'(Ts,S0,S1).
'$dcglhs'(NT,S0,S,P) :-
    nonvar(NT),
    '$tag'(NT,S0,S,P).

'$dcgrhs'((X1,X2),S0,S,P) :- !,
    '$dcgrhs'(X1,S0,S1,P1),
    '$dcgrhs'(X2,S1,S,P2),
    '$and'(P1,P2,P).
'$dcgrhs'((X1;X2),S0,S,(P1;P2)) :- !,
    '$dcgor'(X1,S0,S,P1),
    '$dcgor'(X2,S0,S,P2).
'$dcgrhs'({P},S,S,P) :- !.
'$dcgrhs'(!,S,S,!) :- !.
'$dcgrhs'(Ts,S0,S,true) :-
    '$islist'(Ts), !,
    '$append'(Ts,S,S0).
'$dcgrhs'(X,S0,S,P) :- '$tag'(X,S0,S,P).
'$dcgor'(X,S0,S,P) :-
     '$dcgrhs'(X,S0a,S,Pa),
```

```
( var(S0a), S0a \== S, !, S0=S0a, P=Pa;
  P=(S0=S0a,Pa) ).

'$tag'(X,S0,S,P) :-
   X=..[F|A],
    '$append'(A,[S0,S],AX),
   P=..[F|AX].

/* Auxiliary predicates */

'$and'(true,P,P) :- !.
'$and'(P,true,P) :- !.
'$and'(P,Q,(P,Q)).

'$flatconj'(A,A) :- var(A), !.
'$flatconj'((A,B),C) :- !, '$fc1'(A,C,R), '$flatconj'(B,R).
'$flatconj'(A,A).

  '$fc1'(A,(A,R),R) :- var(A), !.
  '$fc1'((A,B),C,R) :- !, '$fc1'(A,C,R1), '$fc1'(B,R1,R).
  '$fc1'(A,(A,R),R).

   '$islist'([]) :- !.
    '$islist'([_|_]).

'$append'([A|B],C,[A|D]) :- '$append'(B,C,D).
'$append'([],X,X).
```

GENSYM.PRO

```
/*Create a new atom starting with a root provided and
  finishing with a unique number */
gensym(Root,Atom):-
        get_num(Root,Num),
        name(Root,Name1),
        integer_name(Num,Name2),
        append(Name1,Name2,Name),
```

```
        name(Atom,Name),!.

get_num(Root,Num):-
        /*this root encountered before */
        retract(current_num(Root,Num1)),!,
        Num is Num1+1,
        asserta(current_num(Root,Num)).
/* first time for this root */
get_num(Root,1):-asserta(current_num(Root,1)).

/* Convert from an integer to a list of characters */

integer_name(Int,List):-integer_name(Int,[],List).
integer_name(I,Sofar,[C | Sofar]):-
        I<10, !, C is I+48.
integer_name(I,Sofar,List):-
        Tophalf is I//10,
        Bothalf is I mod 10,
        C is Bothalf+48,
        integer_name(Tophalf,[C | Sofar],List).
```

ENG_AD_M.PRO

```
/*      ENG_ADVERB_MAK                          */
/*      ------------                            */
/*      2 types of English adverbs i.e. ending with     */
/*      'ly' and 'ily' are analysed or synthesized.     */
/*------------------------------------------------------*/


eng_adverb_make(Person,PL,Gender,W,Z) :-
            name(W,Adjective),
             do_append(Adj,"y",Adjective),
             do_append(Adj,"ily",Adverbform),
            name(Z,Adverbform).
eng_adverb_make(Person,PL,Gender,W,Z) :-
            name(W,Adjective),
             do_append(Adjective,"ly",Adverbform),
```

```
        name(Z,Adverbform).
```

ENG_V_M.PRO

```
/*  ENG_VERB_MAKE                                   */
/*  -------------                                   */
/*  Forms the English verb according to its type    */
/*  i.e. tense, Person, PL, Gender, Regularity.     */
/*  Eng_verb_form ( contained in WORDS ) finds a match  */
/*  for the irregular verb.                         */
/*--------------------------------------------------*/
eng_verb_make([Tense],first,singular,irregular,W,Z):-
        eng_verb_form([Tense],W,Z,Ss,Ts,Fpl,Spl,Tpl).
eng_verb_make([Tense],second,singular,irregular,W,Z):-
        eng_verb_form([Tense],W,Fs,Z,Ts,Fpl,Spl,Tpl).
eng_verb_make([Tense],third,singular,irregular,W,Z):-
        eng_verb_form([Tense],W,Fs,Ss,Z,Fpl,Spl,Tpl).
eng_verb_make([Tense],first,plural,irregular,W,Z):-
        eng_verb_form([Tense],W,Fs,Ss,Ts,Z,Spl,Tpl).
eng_verb_make([Tense],second,plural,irregular,W,Z):-
        eng_verb_form([Tense],W,Fs,Ss,Ts,Fpl,Z,Tpl).
eng_verb_make([Tense],third,plural,irregular,W,Z):-
        eng_verb_form([Tense],W,Fs,Ss,Ts,Fpl,Spl,Z).


eng_verb_make([present],third,singular,Regularity,W,Z):-
       name(W,Infinitive),
        find_last(Verbstem,"e",Infinitive),
        do_append(Verbstem,"es",Verbform),
       name(Z,Verbform).
eng_verb_make([present],Person,PL,Regularity,W,W):-
       name(W,Infinitive),
        find_last(Verbstem,"e",Infinitive).
eng_verb_make([present],third,singular,Regularity,W,Z):-
       name(W,Verbstem),
        do_append(Verbstem,"s",Verbform),
       name(Z,Verbform).
```

```
eng_verb_make([present],Person,PL,Regularity,W,W).

eng_verb_make([infinitive],Person,PL,Regularity,W,W).

eng_verb_make([past_part],Person,PL,regular,W,Z):-
      name(W,Infinitive),
       find_last(Verbstem,"e",Infinitive),
       do_append(Verbstem,"ed",Verbform),
      name(Z,Verbform).
eng_verb_make([past_part],Person,PL,regular,W,Z):-
      name(W,Verbstem),
       do_append(Verbstem,"ed",Verbform),
      name(Z,Verbform).


eng_verb_make([simple_past],Person,PL,regular,W,Z):-
      name(W,Infinitive),
       find_last(Verbstem,"e",Infinitive),
       do_append(Verbstem,"ed",Verbform),
      name(Z,Verbform).
eng_verb_make([simple_past],Person,PL,regular,W,Z):-
      name(W,Verbstem),
       do_append(Verbstem,"ed",Verbform),
      name(Z,Verbform).

eng_verb_make([used_to],Person,PL,Regularity,W,W).

eng_verb_make([pres_part],Person,PL,Regularity,W,Z):-
      name(W,Infinitive),
       find_last(Verbstem,"e",Infinitive),
       do_append(Verbstem,"ing",Verbform),
      name(Z,Verbform).
eng_verb_make([pres_part],Person,PL,Regularity,W,Z):-
      name(W,Verbstem),
       do_append(Verbstem,"ing",Verbform),
      name(Z,Verbform).

eng_verb_make([future],Person,PL,Regularity,W,W):-
```

```
        name(W,Infinitive),
         find_last(Verbstem,"e",Infinitive).
eng_verb_make([future],Person,PL,Regularity,W,W).

eng_verb_make([imperfect],Person,PL,Regularity,W,Z):-
        name(W,Infinitive),
         find_last(Verbstem,"e",Infinitive),
          do_append(Verbstem,"ing",Verbform),
        name(Z,Verbform).
eng_verb_make([imperfect],Person,PL,Regularity,W,Z):-
        name(W,Verbstem),
         do_append(Verbstem,"ing",Verbform),
        name(Z,Verbform).

eng_verb_make([conditional],Person,PL,Regularity,W,W).

find_last(A,B,C):-fl_append(A,B,C),!.
fl_append([],L,L).
fl_append(_,_,[]):-!,fail.
fl_append([X|L1],L2,[X|L3]):-fl_append(L1,L2,L3).
do_append(A,B,C):-append(A,B,C),!.
```

OURVAR.PRO

```
/*      This program helps to mark our nouns,     */
/*      especially in the case of transitive      */
/*      verbs where the subject and object        */
/*      nouns are marked to prevent synthesizing  */
/*      of the incorrect nouns.                   */
/*------------------------------------------------*/

ourvar(X):-
      var(X),
      !,
        gensym('VAR',X).
ourvar(X):-
      atom(X),
        name('VAR',Var),
```

```
        name(X,XX),
        append(Var,Rest,XX).
```

ASSERTBT.PRO

```
/*   ASSERTBT          */
/*    --------         */

assertbtz(symbol([X])):-
        var(X),
        !,
        fail.
assertbtz(X):-
        nonvar(X),
        assertz(X).
assertbtz(X):-
        nonvar(X),
        retract(X).
assertbta(X):-
        nonvar(X),
        asserta(X).
assertbta(X):-
        nonvar(X),
        retract(X).
assertbt(X):-
        nonvar(X),
        assert(X).
assertbt(X):-
        nonvar(X),
        retract(X).
```

WORDS.PRO

```
/*   WORDS - MONOLINGUAL ENGLISH DICTIONARY  */
/*    ------------------------------------- */
/*     All items of English vocabulary known
       to the system are held in this file.
```

```
        They are classified according to their
        syntactic function .
*/
/*-----------------------------------------------*/


eng_adverb(tomorrow,tomorrow).
eng_adverb(today,today).
eng_adverb(yesterday,yesterday).
eng_adverb(stronger,stronger).

eng_number(one,one,singular).
eng_number(two,two,plural).
eng_number(three,three,plural).
eng_number(four,four,plural).
eng_number(five,five,plural).
eng_number(six,six,plural).
eng_number(seven,seven,plural).
eng_number(eight,eight,plural).
eng_number(nine,nine,plural).
eng_number(ten,ten,plural).
eng_number(eleven,eleven,plural).
eng_number(twelve,twelve,plural).

eng_noun(step_father,step_father).
eng_noun(friend,friend).
eng_noun(desire,desire).
eng_noun(day,day).
eng_noun(night,night).
eng_noun(revenge,revenge).
eng_noun(star,star).
eng_noun(cruelty,cruelty).
eng_noun(meal,meal).
eng_noun(station,station).
eng_noun(street,street).
eng_noun(childhood,childhood).
eng_noun(man,man).
eng_noun(newspaper,newspaper).
```

```
eng_noun(office,office).
eng_noun(horse,horse).
eng_noun(woman,woman).
eng_noun(beer,beer).
eng_noun(garden,garden).
eng_noun(neighbour,neighbour).
eng_noun(person,person).
eng_noun(boy,boy).
eng_noun(girl,girl).
eng_noun(cake,cake).
eng_noun(jelly,jelly).
eng_noun(news,news).
eng_noun(hospital,hospital).
eng_noun(zoo,zoo).
eng_noun(job,job).
eng_noun(time,time).
eng_noun(house,house).
eng_noun(noun,noun).
eng_noun(verb,verb).
eng_noun(adjective,adjective).
eng_noun(lodgings,lodgings).
eng_noun(what,what).
eng_noun(which,which).
eng_noun(building,building).
eng_noun(umbrella,umbrella).
eng_noun(ill,ill).
eng_noun(door,door).
eng_noun(man,man).
eng_noun(home_town,home_town).
eng_noun(year,year).
eng_noun(apple,apple).
eng_noun(table,table).
eng_proper_noun(dylan,dylan).
eng_proper_noun(ian,ian).
eng_proper_noun(john,john).
eng_proper_noun(ann,ann).
eng_proper_noun(mary,mary).
eng_proper_noun(chris,chris).
```

```
eng_proper_noun(who,who).
eng_proper_noun(england,england).
eng_proper_noun(X,X):-
      name(X,[A|B]),
     A>64,
     A<91.

eng_pronoun(him,him,third,singular,masc).
eng_pronoun(he,he,third,singular,masc).
eng_pronoun(she,she,third,singular,fem).
eng_pronoun(i,i,first,singular,Gender).
eng_pronoun(you,you,second,singular,Gender).
eng_pronoun(no_one,no_one,third,plural,Gender).
eng_pronoun(we,we,first,plural,Gender).
eng_pronoun(you,you,second,plural,Gender).
eng_pronoun(they,they,third,plural,Gender).
eng_impers_pronoun(it,it,third,singular,Gender).
eng_impers_pronoun(there,there,third,PL,Gender).
eng_interrog_pronoun(who,who,third,PL,Gender).
eng_plural(beer,beer).
eng_plural(boy,boys).
eng_plural(man,men).
eng_plural(woman,women).
eng_plural(person,people).
eng_plural(what,what).
eng_plural(revenge,revenge).
eng_plural(lodgings,lodgings).

eng_possessive(my,my).
eng_possessive(your,your).
eng_possessive(his,his).
eng_possessive(her,her).
eng_possessive(our,our).
eng_possessive(their,their).

eng_adjective(happy,happy).
eng_adjective(red,red).
eng_adjective(big,big).
```

```
eng_adjective(unpleasant,unpleasant).
eng_adjective(several,several).
eng_adjective(beautiful,beautiful).
eng_adjective(small,small).
eng_adjective(little,little).
eng_adjective(green,green).
eng_adjective(sad,sad).
eng_adjective(possible,possible).
eng_adjective(necessary,necessary).
eng_adjective(many,many).
eng_adjective(important,important).
eng_adjective(astonishing,astonishing).
eng_adjective(cold,cold).
eng_adjective(old,old).
eng_adjective(cruel,cruel).
eng_adjective(very,very).

eng_conjunction(if,if).
eng_conjunction(and,and).
eng_conjunction(but,but).
eng_conjunction(before,before).
eng_verb(do,do,transitive,irregular,have,be,do).
eng_verb(like,like,transitive,regular,have,be,do).
eng_verb(drink,drink,transitive,irregular,have,be,do).
eng_verb(love,love,transitive,regular,have,be,do).
eng_verb(eat,eat,transitive,irregular,have,be,do).
eng_verb(sell,sell,transitive,irregular,have,be,do).
eng_verb(meet,meet,transitive,irregular,have,be,do).
eng_verb(open,open,transitive,regular,have,be,do).
eng_verb(finish,finish,transitive,regular,have,be,do).
eng_verb(live,live,intransitive,regular,have,be,do).
eng_verb(arrive,arrive,intransitive,regular,have,be,do).
eng_verb(listen,listen,intransitive,regular,have,be,do).
eng_verb(dance,dance,intransitive,regular,have,be,do).
eng_verb(have,have,transitive,irregular,have,be,do).
eng_verb(be,be,intransitive,irregular,have,be,do).
eng_verb(be,be,transitive,irregular,have,be,do).
eng_verb(beat,beat,transitive,regular,have,be,do).
```

```
eng_verb(want,want,transitive,regular,have,be,do).
eng_verb(regret,regret,transitive,regular,have,be,do).
eng_verb(wait,wait,transitive,regular,have,be,do).
eng_verb(pity,pity,transitive,irregular,have,be,do).
eng_verb(see,see,transitive,irregular,have,be,do).
eng_verb(tell,tell,transitive,irregular,have,be,do).
eng_verb(forgive,forgive,transitive,irregular,have,be,do).
eng_verb(decide,decide,transitive,regular,have,be,do).
eng_verb(grow,grow,intransitive,irregular,have,be,do).
eng_verb(clean,clean,transitive,regular,have,be,do).
eng_verb(cook,cook,transitive,regular,have,be,do).
eng_verb(escape,escape,transitive,regular,have,be,do).
eng_verb(sleep,sleep,transitive,irregular,have,be,do).
eng_verb(run,run,transitive,irregular,have,be,do).
eng_verb(walk,walk,transitive,regular,have,be,do).
eng_verb(make,make,transitive,irregular,have,be,do).
eng_verb(leave,leave,transitive,irregular,have,be,do).
eng_verb(get,get,transitive,irregular,have,be,do).
eng_verb(find,find,transitive,irregular,have,be,do).
eng_verb(return,return,transitive,regular,have,be,do).
eng_verb(pay,pay,transitive,regular,have,be,do).
eng_verb(die,die,intransitive,regular,have,be,do).
eng_verb(rush,rush,transitive,regular,have,be,do).

eng_verb_form([simple_past],forgive,forgave,forgave,forgave,forgav
e,forgave,
forgave).
eng_verb_form([simple_past],pity,pitied,pitied,pitied,pitied,pitied,pit
ied).
eng_verb_form([simple_past],see,saw,saw,saw,saw,saw,saw).
eng_verb_form([simple_past],tell,told,told,told,told,told,told).
eng_verb_form([simple_past],grow,grew,grew,grew,grew,grew,gre
w).
eng_verb_form([simple_past],make,made,made,made,made,made,
made).
eng_verb_form([simple_past],find,found,found,found,found,found,f
ound).
eng_verb_form([simple_past],get,got,got,got,got,got,got).
```

```
eng_verb_form([simple_past],run,ran,ran,ran,ran,ran,ran).
eng_verb_form([simple_past],sleep,slept,slept,slept,slept,slept,slept)
.
eng_verb_form([simple_past],leave,left,left,left,left,left,left).
eng_verb_form([present],do,do,do,does,do,do,do).
eng_verb_form([simple_past],do,did,did,did,did,did,did).
eng_verb_form([future],be,will,will,will,will,will).
eng_verb_form([simple_past],have,had,had,had,had,had,had).
eng_verb_form([simple_past],be,was,were,was,were,were,were).
eng_verb_form([simple_past],drink,drank,drank,drank,drank,dran
k,drank).
eng_verb_form([simple_past],eat,ate,ate,ate,ate,ate,ate).
eng_verb_form([simple_past],sell,sold,sold,sold,sold,sold,sold).
eng_verb_form([simple_past],meet,met,met,met,met,met,met).
eng_verb_form([past_part],meet,met,met,met,met,met,met).
eng_verb_form([past_part],drink,drunk,drunk,drunk,drunk,drunk
,drunk).
eng_verb_form([past_part],eat,eaten,eaten,eaten,eaten,eaten,eaten).
eng_verb_form([past_part],sell,sold,sold,sold,sold,sold,sold).
eng_verb_form([past_part],have,had,had,had,had,had,had).
eng_verb_form([past_part],be,been,been,been,been,been,been).
eng_verb_form([imperfect_be],be,was,were,was,were,were,were).
eng_verb_form([present],be,am,are,is,are,are,are).
eng_verb_form([present],have,have,have,has,have,have,have).
eng_verb_form([present],eat,eat,eat,eats,eat,eat,eat).
eng_verb_form([present],X,X,X,X,X,X,X).
eng_verb_form([future],X,X,X,X,X,X,X).
eng_verb_form([conditional],X,X,X,X,X,X,X).

eng_tense(infinitive,[infinitive]).
eng_tense(pres_part,[pres_part]).
eng_tense(past_part,[past_part]).
eng_tense(perfect,[perfect]).
eng_tense(present,[present]).
eng_tense(future,[future]).
eng_tense(imperfect_be,[imperfect_be]).
eng_tense(imperfect,[imperfect]).
eng_tense(simple_past,[simple_past]).
```

```
eng_tense(conditional,[conditional]).
eng_tense(used_to,[used_to]).

eng_preposition(under,under).
eng_preposition(at,at).
eng_preposition(after,after).
eng_preposition(along,along).
eng_preposition(among,among).
eng_preposition(before,before).
eng_preposition(behind,behind).
eng_preposition(between,between).
eng_preposition(by,by).
eng_preposition(during,during).
eng_preposition(except,except).
eng_preposition(for,for).
eng_preposition(from,from).
eng_preposition(in,in).
eng_preposition(into,into).
eng_preposition(near,near).
eng_preposition(nearly,nearly).
eng_preposition(of,of).
eng_preposition(off,off).
eng_preposition(on,on).
eng_preposition(onto,onto).
eng_preposition(since,since).
eng_preposition(through,through).
eng_preposition(to,to).
eng_preposition(towards,towards).
eng_preposition(via,via).
eng_preposition(while,while).
eng_preposition(with,with).
eng_preposition(within,within).
eng_preposition(without,without).


eng_determiner(all,all,plural,universal).
eng_determiner(the,the,singular,universal).
eng_determiner(the,the,plural,universal).
```

```
eng_determiner(every,every,singular,universal).
eng_determiner(some,some,plural,existential).
eng_determiner(some,some,singular,existential).
eng_determiner(a,a,singular,existential).
eng_determiner(an,an,singular,existential).
eng_determiner(one,one,singular,existential).
eng_determiner(the,the,singular,universal).
```

ENGLISH5.PRO

```
/*      ---------- */

    /* This file contains the set of Prolog grammar rules which are
used to
        analyse an English sentence when English is the source
language,
        or synthesise an English sentence when English is the target
        language. When English is the SL, the resultant P.C. is
        represented by the variable P which is passed to SUBSTITUTE
        for conversion to the TL PC.
    */
    /*-----------------------------------------------------------------*/

    /*    SENTENCE      */
    /*    --------      */

     eng_statement(P)-->
          noun_phrase(Person,PL,Gender,X,P1,P),
          verb_phrase(Person,PL,Gender,X,P1).
     eng_statement(no).

     eng_question(P)-->
          noun_phrase(Person,PL,Gender,X,P1,P),
          verb_phrase(Person,PL,Gender,X,P1).

     eng_question(P)-->
```

```
    i_auxiliary(Person,PL,Gender,X,P).

eng_question(P)-->
    t_auxiliary(Person,PL,Gender,X,Y,P).

eng_conjunct(f(W,P,L))-->
   eng_statement(P),
   rest_of_statement(W,L).

rest_of_statement(W,L)-->
   gconjunction(W),
   eng_statement(L).


/*    CONJUNCTION    */
/*    -----------    */
gconjunction(f(W))-->
    [Z],
      { (pre_thing(eng_conjunction,[W,Z]) ,
       assertbtz(symbol(W))) } .


/*    NOUN_PHRASE    */
/*    -----------    */
  noun_phrase(Person,PL,Gender,X,P1,exists(X,U,P))-->
     { checksim(P1,P) } ,
       ((gproper_noun(Person,PL,Gender,X,U));
       (gpronoun(Person,PL,Gender,X,U));
        (ginterrog_pronoun(Person,PL,Gender,X,U))).

  noun_phrase(Person,PL,Gender,X,P1,P)-->
        gdeterminer(Person,PL,Gender,X,P2,P1,P),
       rest_np(Person,PL,Gender,X,P2).

  noun_phrase(Person,PL,Gender,X,P1,exists(X,U,P))-->
        { checksim(P1,P) } ,
          (gimpers_pronoun(Person,PL,Gender,X,U)).
```

```
noun_phrase(Person,PL,Gender,X,P1,P)-->
        (((gnumber(Person,PL,Gender,X,P2,P1,P));
        (gpossessive(Person1,PL1,Gender1,X,P2,P1,P));
        (gadjective(Person,PL,Gender,P2,P))),
        (gnoun(Person,PL,Gender,X,P2))).


noun_phrase(Person,PL,Gender,X,P1,P)-->
        gpossessive(Person1,PL1,Gender1,X,P2,P1,P),
         adj_phrase(Person,PL,Gender,X,P2).

noun_phrase(Person,PL,Gender,X,P1,P)-->
       { checksim(P1,P) } ,
         gnoun(Person,PL,Gender,X,P).

noun_phrase(Person,PL,Gender,X,P1,P)-->
         prep_phrase(Person,PL,Gender,P1,P).

rest_np(Person,PL,Gender,X,P2)-->gnoun(Person,PL,Gender,X,P2).
rest_np(Person,PL,Gender,X,P2)--
>adj_phrase(Person,PL,Gender,X,P2).
rest_np(Person,PL,Gender,X,P2)-->gnoun(Person,PL,Gender,X,P3),
                    rel_clause(Person,PL,Gender,X,P3,P2).

/*    VERB_PHRASE    */
/*    ----------     */
 verb_phrase(Person,PL,Gender,X,P&L)-->
         intrans_verb(Person,PL,Gender,X,P),
         rest_vp(Person,PL,Gender,X,L).

 verb_phrase(Person,PL,Gender,X,P)-->
         intrans_verb(Person,PL,Gender,X,P).
verb_phrase(Person,PL,Gender,X,P)-->
         pre_trans_verb(Person,PL,Gender,X,Y,P1),
         noun_phrase(Person1,PL1,Gender1,Y,P1,P).

 verb_phrase(Person,PL,Gender,X,P)-->
         pre_trans_verb(Person,PL,Gender,X,P1,P),
```

```
            noun_phrase(Person1,PL1,Gender1,Y,P2,P1),
            prep_phrase(Person1,PL1,Gender1,X,P2).

 rest_vp(Person,PL,Gender,X,L)-->gadverb(Person,PL,Gender,X,L).
 rest_vp(Person,PL,Gender,X,L)--
>prep_phrase(Person,PL,Gender,X,L).
 rest_vp(Person,PL,Gender,X,L)-->g_adj(Person,PL,Gender,X,L).
 rest_vp(Person,PL,Gender,X,L).
/*    ADVERB */
    /*    ------ */
      gadverb(Person,PL,Gender,X1,(f(W,X)&sing(X)))-->
          { checksim(X,X1) } ,
          [Z],
           { (pre_thing(eng_adverb,[W,Z]),
           assertbtz(symbol(W))) } .

      gadverb(Person,PL,Gender,X1,(f(W,X)&pl(X)))-->
          { checksim(X,X1) } ,
          [Z],
           { (pre_thing(eng_adverb,[W,Z]),
           assertbtz(symbol(W))) } .

       gadverb(Person,PL,Gender,X1,(f(W,X)&sing(X)))-->
          { checksim(X,X1) } ,
          [Z],
            { (pre_thing(eng_adjective,[W,V]),
              eng_adverb_make(Person,PL,Gender,V,Z),
            assertbtz(symbol(V))) } .

    /*    NUMBER */
    /*    ------ */
       gnumber(Person,plural,Gender,X,P1,P2,
(exists(X,det(W),(P1t&P2t))))-->
          { (ourvar(X),
           checksim(P1,P1t),
           checksim(P2,P2t)) } ,
          [Z],
            { (pre_thing(eng_number,[W,Z,plural]),
```

```
        assertbtz(symbol(W))) } .

    gnumber(Person,singular,Gender,X,P1,P2,
(exists(X,det(W),(P1t&P2t))))-->

        { (ourvar(X),
         checksim(P1,P1t),
         checksim(P2,P2t)) } ,
        [Z],
          { (pre_thing(eng_number,[W,Z,singular]),
          assertbtz(symbol(W))) } .

/*  PREP_PHRASE    */
/*  ----------     */
 prep_phrase(Person,PL,Gender,X,P)-->
      [Z],
        { (pre_thing(eng_preposition,[W,Z]),
        assertbtz(symbol(W))) } ,
          noun_phrase(Person1,PL1,Gender1,Y,f(W,X,Y),P).

/*  REL_CLAUSE     */
/*  ----------     */
 rel_clause(Person,PL,Gender,X,P1,P1t&P2)-->
      [that],
       { (checksim(P1,P1t),
        assertbtz(symbol(that))) } ,
         verb_phrase(Person,PL,Gender,X,P2).

 rel_clause(Person,PL,Gender,X,P1,P1t&P2)-->
      [who],
       { (checksim(P1,P1t),
        assertbtz(symbol(who))) } ,
         verb_phrase(Person,PL,Gender,X,P2).

 rel_clause(Person,PL,Gender,X,P1,P1t&P2)-->
      [which],
       { (checksim(P1,P1t),
        assertbtz(symbol(which))) } ,
```

```
                verb_phrase(Person,PL,Gender,X,P2).

/*   GDETERMINER    */
      /*   ----------    */
/*          gdeterminer(Person,plural,Gender,X,P1,P2,
(all(X,(P1t=>P2t))))-->
           { (ourvar(X),
             checksim(P1,P1t),
             checksim(P2,P2t)) } ,
            [].
*/
        gdeterminer(Person,plural,Gender,X,P1,P2,
(all(X,det(W),(P1t=>P2t))))-->
           { (ourvar(X),
             checksim(P1,P1t),
             checksim(P2,P2t)) } ,
            [Z],
              { (pre_thing(eng_determiner,[W,Z,plural,universal]),
             assertbtz(symbol(W))) } .

         gdeterminer(Person,singular,Gender,X,P1,P2,
(exists(X,det(W),(P1t&P2t))))-->
           { (ourvar(X),
             checksim(P1,P1t),
             checksim(P2,P2t)) } ,
            [Z],
               { (pre_thing(eng_determiner,[W,Z,singular,existential]),
              assertbtz(symbol(W))) } .

       gdeterminer(Person,singular,Gender,X,P1,P2,(all(X,det(W),
(P1t=>P2t))))-->
           { (ourvar(X),
           checksim(P1,P1t),
           checksim(P2,P2t)) } ,
          [Z],
             { (pre_thing(eng_determiner,[W,Z,singular,universal]),
           assertbtz(symbol(W))) } .
```

```
        gdeterminer(Person,plural,Gender,X,P1,P2,
(exists(X,det(W),(P1t&P2t)))-->
            { (ourvar(X),
            checksim(P1,P1t),
            checksim(P2,P2t)) } ,
            [Z],
              { (pre_thing(eng_determiner,[W,Z,plural,existential]),
            assertbtz(symbol(W))) } .

      /*    GNOUN        */
      /*    -----        */
        gnoun(third,singular,Gender,X,(f(W,X)&sing(X)))-->
            [Z],
              { (pre_thing(eng_noun,[W,Z]),
            assertbtz(symbol(W))) } .

        gnoun(third,plural,Gender,X,(f(W,X)&pl(X)))-->
            [Z],
             { (eng_plural(Y,Z),
              pre_thing(eng_noun,[W,Y]),
            assertbtz(symbol(W))) } .

        gnoun(third,plural,Gender,X,(f(W,X)&pl(X)))-->
            [Z],
            { (not(var(Z)),
             name(Z,Pluralname),
               fl_append(Singularname1,"ies",Pluralname),
              append(Singularname1,"y",Singularname2),
              name(Singularname,Singularname2),
               pre_thing(eng_noun,[W,Singularname]),
            assertbtz(symbol(W))) } .

        gnoun(third,plural,Gender,X,(f(W,X)&pl(X)))-->
            [Z],
            { (not(var(Z)),
             name(Z,Pluralname),
               fl_append(Singularname1,"s",Pluralname),
              name(Singularname,Singularname1),
```

```
          pre_thing(eng_noun,[W,Singularname]),
        assertbtz(symbol(W))) } .

    gnoun(third,plural,Gender,X,(f(W,X)&pl(X)))-->
        [Z],
          { (pre_thing(eng_noun,[W,Singularname]),
          name(Singularname,Singularname1),
          append(Singularname1,"s",Pluralname),
         name(Z,Pluralname),
        assertbtz(symbol(W))) } .

    gnoun(third,plural,Gender,X,(f(W,X)&pl(X)))-->
        [Z],
          { (pre_thing(eng_noun,[W,Singularname]),
          name(Singularname,Singularname2),
          fl_append(Singularname1,"y",Singularname2),
          append(Singularname1,"ies",Pluralname),
         name(Z,Pluralname)) } .

  /*    ADJECTIVAL PHRASE      */
  /*    ----------------       */
   adj_phrase(Person,PL,Gender,X,(P2&P1))-->
         g_adj(Person,PL,Gender,X,P1),
         gnoun(Person,PL,Gender,X,P2).


/*    GADJECTIVE     */
  /*    ----------     */
    g_adj(third,singular,Gender,X,(f(W,X)&sing(X)))-->
        [Z],
          { (pre_thing(eng_adjective,[W,Z]),
        assertbtz(symbol(W))) } .

    g_adj(third,plural,Gender,X,(f(W,X)&pl(X)))-->
        [Z],
          { (pre_thing(eng_adjective,[W,Z]),
        assertbtz(symbol(W))) } .
```

```
gadjective(Person,singular,Gender,X,(f(W,X)&sing(X)))-->
    [Z],
     { (pre_thing(eng_adjective,[W,Z]),
    assertbtz(symbol(W))) } .

gadjective(Person,plural,Gender,X,(f(W,X)&sing(X)))-->
    [Z],
     { (pre_thing(eng_adjective,[W,Z]),
    assertbtz(symbol(W))) } .

/*   GPOSSESSIVE   */
/*   ----------    */
 gpossessive(Person,PL,Gender,X,P1,P2,
(f(X,det(W),(P1t&P2t))))-->
    { (ourvar(X),
    checksim(P1,P1t),
    checksim(P2,P2t)) } ,
    [Z],
     { (pre_thing(eng_possessive,[W,Z]),
    assertbtz(symbol(W))) } .

/*   GPROPER_NOUN        */
/*   ------------        */
  gproper_noun(third,singular,Gender,W,proper_noun(W))-->
    [X],
      { (pre_thing(eng_proper_noun,[W,X]),
    assertbtz(symbol(W))) } .

  gproper_noun(third,plural,Gender,W,proper_noun(W))-->
    [X],
     { (eng_plural(P,X),
       pre_thing(eng_proper_noun,[W,P]),
       not(pre_thing(eng_noun,[Y,P]))) } .

/*   GPRONOUN     */
/*   --------     */
  gpronoun(Person,Plurality,Gender,W,pronoun(W))-->
    [Z],
```

```
        { (pre_thing(eng_pronoun,[W,Z,Person,Plurality,Gender]),
        assertbtz(symbol(W))) } .

    /*    G_IMPERSONAL_PRONOUN   */
    /*    ------------------- */
      gimpers_pronoun(Person,PL,Gender,W,impers(W))-->
        [Z],
        {
(pre_thing(eng_impers_pronoun,[W,Z,Person,PL,Gender]),
        assertbtz(symbol(W))) } .

    /*    INTERROGATIVE PRONOUN   */
    /*    ------------------- */
      ginterrog_pronoun(third,singular,Gender,W,interrog(W))-->
        [Z],
        {
(pre_thing(eng_interrog_pronoun,[W,Z,Person,PL,Gender]),
        assertbtz(symbol(W))) } .

    /*    TRANS_INTERROGATIVES PRESENT TENSE    */
    /*    ---------------------------------- */
      t_auxiliary(Person,PL,Gender,X1,Y1,
(P,Q,(f(Do,V,X,Y)&tense(([present],Do),
        ([infinitive],V)))))-->
        { checksim(X,X1) },
        { checksim(Y,Y1) } ,
        [Z],
        { (pre_thing(eng_tense,[T,[present]]),
    pre_thing(eng_verb,[Do,W,Transitivity,irregular,H,B,D])) },
        { (eng_verb_make([present],Person,PL,irregular,W,Z),
        assertbtz(symbol([Do]))) } ,
        noun_phrase(Person,PL,Gender,X1,P1,P),
        [M],
        { (pre_thing(eng_tense,[U,[infinitive]]),
    pre_thing(eng_verb,[V,N,transitive,Regularity,H,B,D]),
    eng_verb_make([infinitive],Person,PL,Regularity,N,M),
        assertbtz(symbol(V))) } ,
        noun_phrase(Person1,PL1,Gender1,Y1,P2,Q).
```

```
    /*    INTRANS_INTERROGATIVES PRESENT TENSE   */
    /*    ---------------------------------- */
     i_auxiliary(Person,PL,Gender,X1,
(f(Do,V,X)&tense(([present],Do),([infinitive],V))))-->
          { checksim(X,X1) } ,
          [Z],
           { (pre_thing(eng_tense,[T,[present]]),
   pre_thing(eng_verb,[Do,W,Transitivity,irregular,H,B,D])) } ,
            { (eng_verb_make([present],Person,PL,irregular,W,Z),
          assertbtz(symbol([Do]))) } ,
           noun_phrase(Person,PL,Gender,X1,P1,P),
          [M],
           { (pre_thing(eng_tense,[U,[infinitive]]),
       pre_thing(eng_verb,[V,N,intransitive,Regularity,H,B,D]),
       eng_verb_make([infinitive],Person,PL,Regularity,N,M),
          assertbtz(symbol(V))) } .


    /*    CHECK TRANS_VERB          */
    /*    ----------------          */

      pre_trans_verb(Person,PL,Gender,X1,Y1,
(f(V,X,Y)&tense(T1,V)))-->
          { checksim(X,X1) },
          { checksim(Y,Y1) } ,
       trans_verb(Person,PL,Gender,X1,Y1,(f(V,X,Y)&tense(T1,V))).

    /*    TRANS_VERB,SIMPLE_PAST TENSE   */
    /*    ------------------------------ */
  trans_verb(Person,PL,Gender,X1,Y1,(f(V,X,Y)&tense(T1,V)))-->
          { checksim(T1,[simple_past]) } ,
          [Z],
           { (pre_thing(eng_tense,[T,[simple_past]]),
       pre_thing(eng_verb,[V,W,transitive,Regularity,Have,Be,Do]),
       eng_verb_make([simple_past],Person,PL,Regularity,W,Z),
          assertbtz(symbol(V))) } .
```

```
    /*    INTRANS_VERB SIMPLE PAST TENSE  */
    /*    ------------------------------ */
     intrans_verb(Person,PL,Gender,X1,(f(V,X)&tense(T1,V)))-->
         { checksim(X,X1) },
         { checksim(T1,[simple_past]) } ,
        [Z],
          { (pre_thing(eng_tense,[T,[simple_past]]),
      pre_thing(eng_verb,[V,W,intransitive,Regularity,Have,Be,Do]),
      eng_verb_make([simple_past],Person,PL,Regularity,W,Z),
         assertbtz(symbol(V))) } .

    /*    TRANS_VERB, "USED_TO"   */
    /*    ---------------------   */
     trans_verb(Person,PL,Gender,X1,Y1,
(f(V,X,Y)&tense([used_to],V)))-->
         [used],
         [to],
         [Z],
           { (pre_thing(eng_tense,[T,[used_to]]),
      pre_thing(eng_verb,[V,W,transitive,Regularity,Have,Be,Do]),
      eng_verb_make([used_to],Person,PL,Regularity,W,Z),
         assertbtz(symbol(V))) } .

    /*    INTRANS_VERB, "USED_TO" */
    /*    ----------------------- */
     intrans_verb(Person,PL,Gender,X1,
(f(V,X)&tense([used_to],V)))-->
         { checksim(X,X1) } ,
         [used],
         [to],
         [Z],
           { (pre_thing(eng_tense,[T,[used_to]]),
      pre_thing(eng_verb,[V,W,intransitive,Regularity,Have,Be,Do]),
      eng_verb_make([used_to],Person,PL,Regularity,W,Z),
         assertbtz(symbol(V))) } .
```

```
    /*    TRANS_VERB PRESENT TENSE       */
    /*    -----------------------        */
  trans_verb(Person,PL,Gender,X1,Y1,(f(V,X,Y)&tense(T1,V)))-->
          { checksim(T1,[present]) } ,
          [Z],
            { (pre_thing(eng_tense,[T,[present]]),
        pre_thing(eng_verb,[V,W,transitive,Regularity,Have,Be,Do]),
        eng_verb_make([present],Person,PL,Regularity,W,Z),
           assertbtz(symbol(V))) } .

    /*    INTRANS_VERB PRESENT TENSE     */
    /*    -------------------------      */
  intrans_verb(Person,PL,Gender,X1,(f(V,X)&tense([T1],V)))-->
          { checksim(X,X1) },
          { checksim([T1],[present]) } ,
          [Z],
            { (pre_thing(eng_tense,[T,[present]]),
      pre_thing(eng_verb,[V,W,intransitive,Regularity,Have,Be,Do]),
      eng_verb_make([present],Person,PL,Regularity,W,Z),
          assertbtz(symbol(V))) } .


    /*    TRANS_VERB,PERFECT TENSE       */
    /*    -----------------------        */
/*       trans_verb(Person,PL,Gender,X,Y,
(f(V,X,Y)&tense([perfect],V)))-->
            trans_verb(Person,PL,Gender,X,Y,(f(Have,V,X,Y)
           &tense(([present],Have),([past_part],V)))).   */


      trans_verb(Person,PL,Gender,X,Y,
(f(Have,V,X,Y)&tense(([present],Have),([past_part],V))))-->
          [C],
          [Z],
          {
(pre_thing(eng_verb,[V,W,transitive,Regularity,Have,Be,Do]),
            eng_verb_make([present],Person,PL,irregular,Have,C),
            eng_verb_make([past_part],Person,PL,Regularity,W,Z),
```

```
        assertbtz(symbol([Have])),
        assertbtz(symbol(V))) } .

    /*    INTRANS_VERB PERFECT TENSE     */
    /*    ------------------------       */
/*      intrans_verb(Person,PL,Gender,X,
(f(V,X)&tense([perfect],V)))-->
          intrans_verb(Person,PL,Gender,X,(f(Have,V,X)
        &tense(([present],Have),([past_part],V)))).    */

     intrans_verb(Person,PL,Gender,X,
(f(Have,V,X)&tense(([present],Have),([past_part],V))))-->
        [C],
        [Z],
        {
(pre_thing(eng_verb,[V,W,intransitive,Regularity,Have,Be,Do]),

          eng_verb_make([present],Person,PL,irregular,Have,C),
          eng_verb_make([past_part],Person,PL,Regularity,W,Z),
        assertbtz(symbol([Have])),
        assertbtz(symbol(V))) } .

    /*    TRANS_VERB,FUTURE TENSE        */
    /*    -----------------------        */

     trans_verb(Person,PL,Gender,X1,Y1,
(f(V,X,Y)&tense([future],V)))-->
        [will],
        [Z],
          { (pre_thing(eng_tense,[T,[future]]),
      pre_thing(eng_verb,[V,W,transitive,Regularity,Have,Be,Do]),
          eng_verb_make([future],Person,P,Regularity,W,Z),
        assertbtz(symbol(V))) } .

     trans_verb(Person,PL,Gender,X1,Y1,
(f(V,X,Y)&tense([future],V)))-->
        [would],
        [Z],
```

```
        { (pre_thing(eng_tense,[T,[future]]),
      pre_thing(eng_verb,[V,W,transitive,Regularity,Have,Be,Do]),
          eng_verb_make([future],Person,P,Regularity,W,Z),
        assertbtz(symbol(V))) } .

    /*    INTRANS_VERB,FUTURE TENSE     */
    /*    ------------------------      */

     intrans_verb(Person,PL,Gender,X1,
(f(V,X)&tense([future],V)))-->
        { checksim(X,X1) } ,
        [will],
        [Z],
          { (pre_thing(eng_tense,[T,[future]]),
      pre_thing(eng_verb,[V,W,intransitive,Regularity,Have,Be,Do]),
          eng_verb_make([future],Person,PL,Regularity,W,Z),
        assertbtz(symbol(V))) } .

     intrans_verb(Person,PL,Gender,X1,
(f(V,X)&tense([future],V)))-->
        { checksim(X,X1) } ,
        [would],
        [Z],
          { (pre_thing(eng_tense,[T,[future]]),
      pre_thing(eng_verb,[V,W,intransitive,Regularity,Have,Be,Do]),
          eng_verb_make([future],Person,PL,Regularity,W,Z),
        assertbtz(symbol(V))) } .

    /*    TRANS_VERB,IMPERFECT TENSE     */
    /*    --------------------------     */
/*      trans_verb(Person,PL,Gender,X,Y,
(f(V,X,Y)&tense([imperfect],V)))-->
          trans_verb(Person,PL,Gender,X,Y,(f(Be,V,X,Y)
        &tense(([imperfect_be],Be),([pres_part],V)))).   */

     trans_verb(Person,PL,Gender,X,Y,
(f(Be,V,X,Y)&tense(([imperfect_be],Be),([pres_part],V))))-->
        [C],
```

```
        [Z],
        {
(pre_thing(eng_verb,[V,W,transitive,Regularity,Have,Be,Do]),
      eng_verb_make([imperfect_be],Person,PL,irregular,Be,C),
      eng_verb_make([pres_part],Person,PL,Regularity,W,Z),
          assertbtz(symbol([Be])),
          assertbtz(symbol(V))) } .

    /*    INTRANS_VERB IMPREFECT TENSE   */
    /*    ---------------------------   */
/*       intrans_verb(Person,PL,Gender,X,(f(V,X)&tense(T1,V)))-->
          { checksim(T1,[imperfect]) } ,
           intrans_verb(Person,PL,Gender,X,(f(Be,V,X)
           &tense(([imperfect_be],Be),([pres_part],V)))).   */

    /*    INTRANS_VERB IMPERFECT TENSE   */
    /*    ---------------------------   */
      intrans_verb(Person,PL,Gender,X,
(f(Be,V,X)&tense(([imperfect_be],Be),([pres_part],V))))-->
          [C],
          [Z],
          {
(pre_thing(eng_verb,[V,W,intransitive,Regularity,Have,Be,Do]),

      eng_verb_make([imperfect_be],Person,PL,irregular,Be,C),
      eng_verb_make([pres_part],Person,PL,Regularity,W,Z),
          assertbtz(symbol([Be])),
          assertbtz(symbol(V))) } .

    /*    INTRANS_VERB IMPERFECT OF VERB TO BE   */
    /*    ----------------------------------   */
      intrans_verb(Person,PL,Gender,X,
(f(V,X)&tense([imperfect_be],V)))-->
          [Z],
            { (pre_thing(eng_tense,[T,[imperfect_be]]),
      pre_thing(eng_verb,[V,W,intransitive,irregular,Have,Be,Do]),
      eng_verb_make([imperfect_be],Person,PL,irregular,W,Z),
          assertbtz(symbol(V))) } .
```

```
    /*    TRANS_VERB,CONDITIONAL TENSE    */
    /*    ---------------------------    */
     trans_verb(Person,PL,Gender,X1,Y1,
(f(V,X,Y)&tense([conditional],V)))-->
          [would],
          [Z],
           { (pre_thing(eng_tense,[T,[conditional]]),
     pre_thing(eng_verb,[V,W,transitive,Regularity,Have,Be,Do]),
     eng_verb_make([conditional],Person,PL,Regularity,W,Z),
          assertbtz(symbol(V))) } .

    /*    INTRANS_VERB CONDITIONAL      */
    /*    ------------------------      */
     intrans_verb(Person,PL,Gender,X1,
(f(V,X)&tense([conditional],V)))-->
          { checksim(X,X1) } ,
          [would],
          [Z],
           { (pre_thing(eng_tense,[T,[conditional]]),
     pre_thing(eng_verb,[V,W,intransitive,Regularity,Have,Be,Do]),
     eng_verb_make([conditional],Person,PL,Regularity,W,Z),
          assertbtz(symbol(V))) } .
```

PARSER.PRO

```
/* This file consults the story file, parses the story using ENGLISH5
  and asserts the PC into the database. On completion the story file
  is retracted.  */
/*-----------------------------------------------------------------*/
go:-consult('a:tempstry.pro'),
   parse,
   retractall(sent).

parse:-sent(S),do_sentence(S).
parse.
```

```
do_sentence(S):-eng_statement(P,S,T),
        rest_stat(P,S,T),!,fail.

rest_stat(P,S,[]):-assertz(pc_sent(P)).
rest_stat(P,S,[H|T]):-eng_conjunction(H,H),
            assertz(pc_sent(P)),
            eng_statement(P1,T,[]),
            assertz(pc_sent(f(H,P1))).
```

TEMPSTRY.PRO

Example of a simple story with each sentence packaged up into a Prolog predicate.

```
sent([dylan,had,an,unpleasant,childhood]).
sent([his,step_father,beat,him]).
sent([he,cleaned,the,house,and,he,cooked,the,meals]).
sent([dylan,wanted,some,revenge]).
sent([his,step_father,would,regret,his,cruelty]).
sent([dylan,waited,two,years,before,he,escaped,from,his,cruel,step_
father]).
sent([he,left,the,house,on,a,cold,night,and,he,ran,to,the,station]).
sent([dylan,walked,the,streets,during,the,day,and,he,slept,every,nig
ht,under,
the,stars]).
sent([he,got,a,job,after,some,time,and,he,found,some,lodgings]).
sent([dylan,made,many,friends]).
sent([dylan,wanted,the,revenge,towards,his,step_father]).
sent([his,desire,grew,stronger]).
sent([dylan,returned,to,his,home_town,within,a,year]).
sent([his,step_father,would,pay,for,his,cruelty]).
sent([there,was,no_one,in,the,house]).
sent([a,neighbour,told,the,news,to,dylan]).
sent([his,step_father,was,in,the,hospital]).
sent([dylan,rushed,to,the,hospital]).
sent([he,would,have,his,revenge]).
sent([dylan,saw,his,step_father,but,he,pitied,the,old,man]).
```

```
sent([his,step_father,was,very,ill]).
sent([dylan,forgave,his,step_father,before,he,died,in,the,hospital]).
```

OUTPUT1.PRO

Using the above story as input, the following is the output from the parser.

```
pc_sent(exists([dylan],proper_noun([dylan]),exists('VAR2',det([an]),
((f([childhoo
d],'VAR2')&sing('VAR2'))&f([unpleasant],'VAR2')&sing('VAR2'))&f
([have],[dylan],'V
AR2')&tense([simple_past],[have])))) .

pc_sent(f('VAR9',det([his]),(f([step_father],'VAR9')&sing('VAR9'))&
exists([him],p
ronoun([him]),f([beat],'VAR9',[him])&tense([present],[beat])))) .

pc_sent(exists([he],pronoun([he]),all('VAR12',det([the]),f([house],'V
AR12')&sing(
'VAR12')=>f([clean],[he],'VAR12')&tense([simple_past],[clean])))) .

pc_sent(f(and,exists([he],pronoun([he]),all('VAR13',det([the]),f([mea
l],'VAR13')&
pl('VAR13')=>f([cook],[he],'VAR13')&tense([simple_past],[cook]))))) .

pc_sent(exists([dylan],proper_noun([dylan]),exists('VAR15',det([so
me]),(f([reveng
e],'VAR15')&sing('VAR15'))&f([want],[dylan],'VAR15')&tense([simp
le_past],[want]))
)) .

pc_sent(f('VAR22',det([his]),(f([step_father],'VAR22')&sing('VAR22'
))&f('VAR29',d
et([his]),(f([cruelty],'VAR29')&sing('VAR29'))&f([regret],'VAR22','V
AR29')&tense(
```

```
[future],[regret]))))) .

pc_sent(exists([dylan],proper_noun([dylan]),exists('VAR34',det([two
]),(f([year],'
VAR34')&pl('VAR34'))&f([wait],[dylan],'VAR34')&tense([simple_pas
t],[wait])))) .

pc_sent(f(before,exists([he],pronoun([he]),f('VAR54',det([his]),((f([ste
p_father]
,'VAR54')&sing('VAR54'))&f([cruel],'VAR54')&sing('VAR54'))&f([fro
m],f([escape],[h
e],_125)&tense([simple_past],[escape]),'VAR54'))))) .

pc_sent(exists([he],pronoun([he]),f([leave],[he],all('VAR77',det([the])
,f([house]
,'VAR77')&sing('VAR77')=>exists('VAR79',det([a]),((f([night],'VAR7
9')&sing('VAR79
'))&f([cold],'VAR79')&sing('VAR79'))&f([on],[he],'VAR79'))))&tense([
simple_past],
[leave]))) .

pc_sent(f(and,exists([he],pronoun([he]),all('VAR92',det([the]),f([stati
on],'VAR92
')&sing('VAR92')=>f([to],f([run],[he],_116)&tense([simple_past],[run
]),'VAR92')))
)) .

pc_sent(exists([dylan],proper_noun([dylan]),f([walk],[dylan],all('VA
R113',det([th
e]),f([street],'VAR113')&pl('VAR113')=>all('VAR116',det([the]),f([day]
,'VAR116')&
sing('VAR116')=>f([during],[dylan],'VAR116'))))&tense([simple_past
],[walk]))) .

pc_sent(f(and,exists([he],pronoun([he]),f([sleep],[he],all('VAR139',de
t([every]),
f([night],'VAR139')&sing('VAR139')=>all('VAR140',det([the]),f([star]
,'VAR140')&pl
```

('VAR140')=>f([under],[he],'VAR140'))))&tense([simple_past],[sleep])
))) .

pc_sent(exists([he],pronoun([he]),f([get],[he],exists('VAR162',det([a])
,(f([job],
'VAR162')&sing('VAR162'))&exists('VAR164',det([some]),(f([time],'V
AR164')&sing('V
AR164'))&f([after],[he],'VAR164'))))&tense([simple_past],[get]))) .

pc_sent(f(and,exists([he],pronoun([he]),exists('VAR166',det([some]),(
f([lodgings]
,'VAR166')&sing('VAR166'))&f([find],[he],'VAR166')&tense([simple_
past],[find])))))
) .

pc_sent(exists([dylan],proper_noun([dylan]),f([many],'VAR175')&si
ng('VAR175')','(
f([friend],'VAR175')&pl('VAR175'))&f([make],[dylan],'VAR175')&ten
se([simple_past]
,[make]))) .

pc_sent(exists([dylan],proper_noun([dylan]),f([want],[dylan],all('VA
R206',det([th
e]),f([revenge],'VAR206')&pl('VAR206')=>f('VAR213',det([his]),(f([ste
p_father],'V
AR213')&sing('VAR213'))&f([towards],[dylan],'VAR213'))))&tense([si
mple_past],[wan
t]))) .

pc_sent(f('VAR220',det([his]),(f([desire],'VAR220')&sing('VAR220'))
&(f([grow],'VA
R220')&tense([simple_past],[grow]))&f([stronger],'VAR220')&sing('V
AR220'))) .

pc_sent(exists([dylan],proper_noun([dylan]),f([return],[dylan],f('VA
R307',det([hi
s]),(f([home_town],'VAR307')&sing('VAR307'))&f([to],exists('VAR30
9',det([a]),(f([

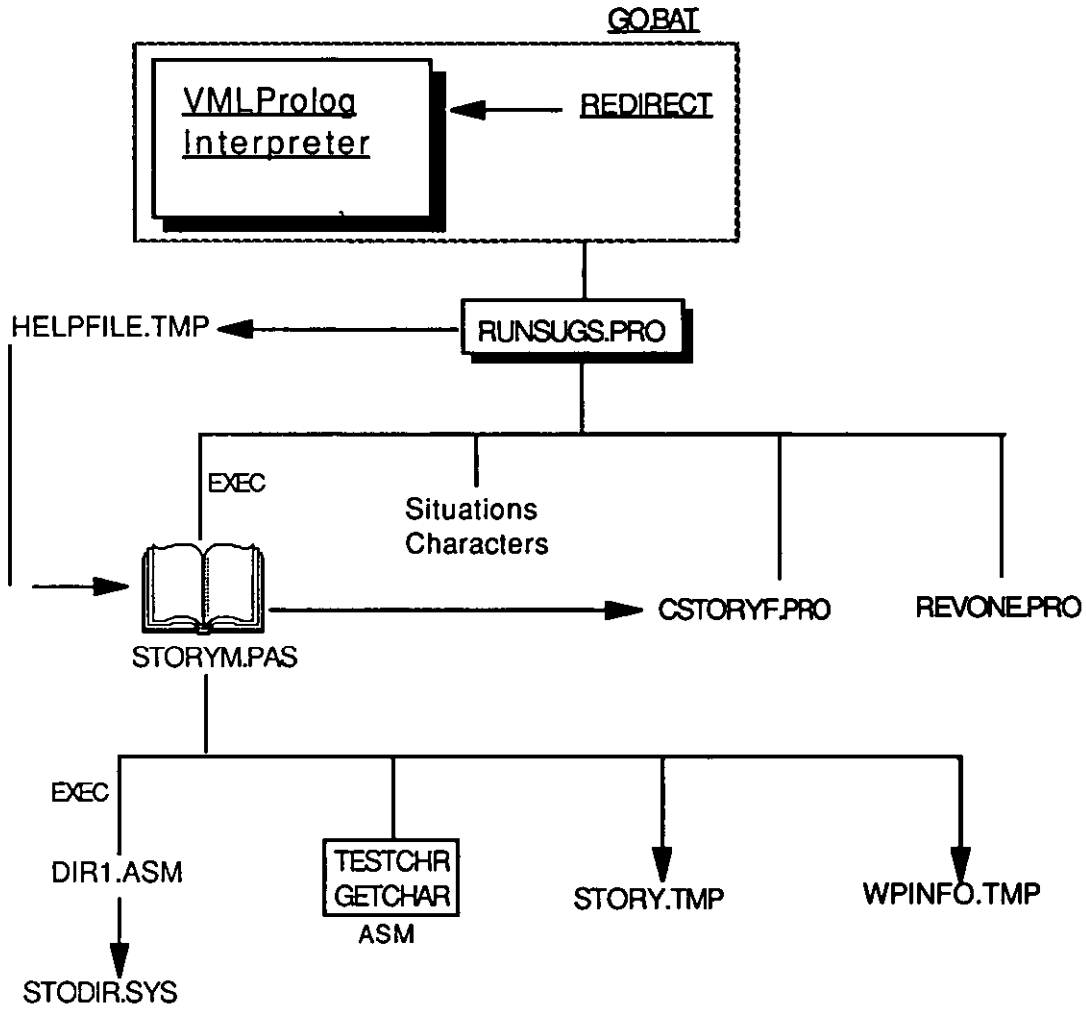
```
year],'VAR309')&sing('VAR309'))&f([within],[dylan],'VAR309')),'VA
R307')))&tense([
simple_past],[return])))  .

pc_sent(f('VAR316',det([his]),(f([step_father],'VAR316')&sing('VAR
316'))&f('VAR33
3',det([his]),(f([cruelty],'VAR333')&sing('VAR333'))&f([for],f([pay],'V
AR316',_11
9)&tense([future],[pay]),'VAR333'))))  .

pc_sent(exists([there],impers([there]),f([be],[there],exists([no_one],p
ronoun([no
_one]),all('VAR360',det([the]),f([house],'VAR360')&sing('VAR360')=
>f([in],[there]
,'VAR360'))))&tense([simple_past],[be])))  .

pc_sent(exists('VAR362',det([a]),(f([neighbour],'VAR362')&sing('VA
R362'))&f([tell
],'VAR362',all('VAR385',det([the]),f([news],'VAR385')&sing('VAR38
5')=>exists([dyl
an],proper_noun([dylan]),f([to],'VAR362',[dylan])))))&tense([simple_
past],[tell]))
)  .

pc_sent(f('VAR392',det([his]),(f([step_father],'VAR392')&sing('VAR
392'))&(f([be],
'VAR392')&tense([simple_past],[be]))&all('VAR395',det([the]),f([hosp
ital],'VAR395
')&sing('VAR395')=>f([in],'VAR392','VAR395'))))  .

pc_sent(exists([dylan],proper_noun([dylan]),all('VAR408',det([the]),f
([hospital],
'VAR408')&sing('VAR408')=>f([to],f([rush],[dylan],_114)&tense([sim
ple_past],[rush
]),'VAR408'))))  .

pc_sent(exists([he],pronoun([he]),f('VAR415',det([his]),(f([revenge],'
VAR415')&si
```

```
ng('VAR415'))&f([have],[he],'VAR415')&tense([future],[have])))) .

pc_sent(exists([dylan],proper_noun([dylan]),f('VAR422',det([his]),(f([
step_father
],'VAR422')&sing('VAR422'))&f([see],[dylan],'VAR422')&tense([simp
le_past],[see]))
)) .

pc_sent(f(but,exists([he],pronoun([he]),all('VAR425',det([the]),(f([ma
n],'VAR425'
)&sing('VAR425'))&f([old],'VAR425')&sing('VAR425')=>f([pity],[he],'
VAR425')&tense
([simple_past],[pity])))))) .

pc_sent(f('VAR432',det([his]),(f([step_father],'VAR432')&sing('VAR
432'))&(f([very
],'VAR440')&sing('VAR440')','(f([ill],'VAR440')&sing('VAR440'))&f([
be],'VAR432','
VAR440')&tense([simple_past],[be])))) .

pc_sent(exists([dylan],proper_noun([dylan]),f('VAR447',det([his]),(f([
step_father
],'VAR447')&sing('VAR447'))&f([forgive],[dylan],'VAR447')&tense([si
mple_past],[fo
rgive])))) .

pc_sent(f(before,exists([he],pronoun([he]),(f([die],[he])&tense([simple
_past],[di
e]))&all('VAR450',det([the]),f([hospital],'VAR450')&sing('VAR450')=>
f([in],[he],'
VAR450'))))) .
```

APPENDIX 2.

MULTISTORY User Interface Listings

2.1. MULTISTORY System Overview: Rule Based Support System:



GO.BAT - DOS batch file which loads VML Prolog and consults file REDIRECT
REDIRECT - forces Prolog to consult and run RUNSUGS.PRO
RUNSUGS.PRO - main Prolog program. Loops around user interface STORYM.PAS. Produces suggestions for user.
CSTORYF.PRO - contains story type, situation, and character settings for current story. Consulted by RUNSUGS.PRO.
REVONE.PRO - suggestions file for revenge type stories using situation one. Consulted by RUNSUGS.PRO.
HELPFILE.TMP - temporary file holding current suggestion. Read by STORYM.PAS
DIR1.ASM - assembler program returning directory of stories on users floppy disk. Produces file STODIR.SYS.
TESTCHR & GETCHR - assembler routines to poll and intercept the keyboard buffer
STORY.TMP - temporary file containing current story text.
WPINFO.TMP - temporary file containing current word-processor settings.

2.2. Main Pascal source code STORYM.PAS

Pro Pascal Compiler - Version iid 2.1

Compilation of: STORYM.PAS

```
PROGRAM STORYM (INPUT,OUTPUT);
CONST
maxspritesize = 800;
{$I SBGOCNST.PAS}
{$I SBGICNST.PAS}
TYPE
{$I SBGOTYPS.PAS}
STRING136 = STRING[136];
STRING15  = STRING[15];
STRING8   = STRING[8];
TFILE=RECORD
FNAMEOUT:STRING8;
PBEGINOUT,
XCOUT,
YCOUT,
WPINDEXOUT,
OLDXOUT,
            OLDYOUT:INTEGER;
END;
COMMON
{$I SBGOCOMM.PAS}
{$I SBGICOMM.PAS}
VAR OLDX,OLDY,DUMMY1,DUMMY2:INTEGER;
   ATTRIBUTE: ARRAY [1..13] OF INT;
  TEMPFILE:TFILE;
   icode,CH:char;
    temp,STYPE,RN,RC,DINDX,SITUATION,PBEGIN,
    XC,YC,NSPACEX,NSPACEY,TEMPINDEX,TINDEX:int;
    CHARIN,TESTCHAR:BYTE;
   WPINDEX,WPENDMARKER:INT;
   DIRAREA: ARRAY [1..200] OF CHAR;
```

```
   WORKAREA: ARRAY [1..400] OF CHAR;
   WORKAREA2: ARRAY [1..100] OF CHAR;
   INCHAR: ARRAY[1..10] OF CHAR;
   WPAREA: ARRAY[1..25000] OF CHAR;
   TEMPWP: ARRAY[1..1120] OF CHAR;
   SPRITESAVE: ARRAY[1..80] OF INTEGER;
   INPNO:REAL;

LEFTB,RIGHTB,ENDCREATE,FINISHED,HELPSET:BOOLEAN;
   F:FILE OF CHAR;
    CHARACTER,NEWCHARSTRING:STRING;
   FILENAME:STRING8;
   DRIVEFILENAME:STRING15;
{$I SBDPRCS.PAS}
{$I SBGOPRCS.PAS}
{$I SBGIPRCS.PAS}
{-------------------------------------------------}
PROCEDURE EXITPROG(RETCODE:INTEGER); EXTERNAL;
{-------------------------------------------------}
FUNCTION GETCHAR:BYTE; EXTERNAL;
{-------------------------------------------------}
FUNCTION TESTCHR:BYTE; EXTERNAL;
{-------------------------------------------------}
FUNCTION RAND:REAL; EXTERNAL;
{-------------------------------------------------}
PROCEDURE EXECPROG(COMMAND:STRING136;VAR;
RETURNCODE:INTEGER); EXTERNAL;
{-------------------------------------------------}
{new sprite handling routines}
PROCEDURE SAVEAREA1(VAR X,Y:INTEGER);
VAR XT,I,YT:INTEGER;

BEGIN;
          XT:=X-4; YT:=Y-8;
          FOR I:=1 TO 80 DO BEGIN;
                         IF XT=X+4 THEN BEGIN;
                           XT:=X-4;
                           YT:=YT+1;
```

```
                    END;
                    SPRITESAVE[I]:=ENQPIXEL(XT,YT);
                    XT:=XT+1;
        END;
END; {savearea1}
PROCEDURE PLOTSPRITECHAR(VAR X,Y:INTEGER);
BEGIN;
        SETFONT(1);
        PLOTTEXT(X-4,Y-8,CHR(94));
        SETFONT(0);
END; {plotspritechar}
PROCEDURE REDISPLAYSAVEAREA(VAR X,Y:INTEGER);
VAR XT,YT,I:INTEGER;
BEGIN;
        XT:=X-4; YT:=Y-8;
        FOR I:=1 TO 80 DO BEGIN;
                    IF XT=X+4 THEN BEGIN;
                      XT:=X-4;
                      YT:=YT+1;
                    END;
                    SETPIXEL(XT,YT,SPRITESAVE[I]);
                    XT:=XT+1;
        END;
END; {redisplaysavearea}
PROCEDURE DRAWSPRITE1(VAR X,Y:INTEGER);
BEGIN;
        SAVEAREA1(X,Y);
        PLOTSPRITECHAR(X,Y);
END; {drawsprite1}
PROCEDURE MOVESPRITE1(VAR X,Y:INTEGER; NX,NY:INT);
BEGIN;
        IF ((X<>NX)OR(Y<>NY)) THEN BEGIN;
        REDISPLAYSAVEAREA(X,Y);
        X:=NX;
        Y:=NY;
        SAVEAREA1(X,Y);
        PLOTSPRITECHAR(X,Y);
        END;
```

```
END; {movesprite1}
PROCEDURE ERASESPRITE1(VAR X,Y:INTEGER);
BEGIN;
            REDISPLAYSAVEAREA(X,Y);
END; {erasesprite1}
{-----------------------------------------------}
PROCEDURE READSCREEN;
VAR  newx,newy:INT;
  NOTSELECTED:BOOLEAN;
  R1:RECT;
BEGIN;
   LEFTB:=FALSE; RIGHTB:=FALSE; NOTSELECTED:=TRUE;
    ICODE:='X';
            DRAWSPRITE1(OLDX,OLDY);
   WHILE NOTSELECTED DO BEGIN;
         TESTCHAR:=TESTCHR;
         IF TESTCHAR=255 THEN BEGIN;
                  CHARIN:=GETCHAR;
                 NOTSELECTED:=FALSE;
                END
        ELSE BEGIN;
         ENQLOCATION(DEVMOUSE,NEWX,NEWY);
         IF LEFTB OR RIGHTB THEN BEGIN
             LEFTB:=FALSE; RIGHTB:=FALSE;
            NOTSELECTED:=FALSE;
            IF (NEWY >190) AND (NEWY <235)

           THEN BEGIN;
                 IF NEWX<106 THEN
                                                    ICODE:='A'
                ELSE
                 IF NEWX<212 THEN
                                                ICODE:='B'
                ELSE
                 IF NEWX<318 THEN
                                                ICODE:='C'
                 ELSE
```

```
                    IF NEWX<424 THEN
                                                    ICODE:='D'
                   ELSE
                    IF NEWX<530 THEN
                                                    ICODE:='E'
                     ELSE ICODE:='F';
                END;
             END;
             MOVESPRITE1(OLDX,OLDY,NEWX,NEWY);
             OLDX:=NEWX; OLDY:=NEWY;
             MOUSESWITCHES(LEFTB,RIGHTB);
            END;
      END;
            erasesprite1(oldx,oldy);
END;{READSCREEN}
{-------------------------------------------------}
PROCEDURE CLEAR_SCR;
VAR R1:RECT;
BEGIN;
     SETBRUSHCOLOUR(1);
     SETRECT(R1,6,5,633,189);
     FILLRECT(R1);
END;  {clear_scr}
{-------------------------------------------------}
PROCEDURE GETNAME;
VAR I,J,XCOORD:INTEGER;
  NOTFINISHED:BOOLEAN;
  R1:RECT;
  F:FILE OF CHAR;
BEGIN;
            CLEAR_SCR;
            PLOTTEXT(130,150,'Make sure your STORY DISK is in
the drive !!');
            PLOTTEXT(170,130,'Press any key when ready to
continue.');
            TESTCHAR:=0;
            SETRECT(R1,470,236,630,245);
            FILLRECT(R1);
```

```
            REPEAT TESTCHAR:=TESTCHR UNTIL
TESTCHAR=255;
            CHARIN:=GETCHAR;
    CLEAR_SCR;
            IF NOT(FSTAT('A:USERNAME.SYS')) THEN BEGIN;
                                PLOTTEXT(130,170,'Please enter your
name (maximum 20 characters).');
                                PLOTTEXT(230,152,'[                ]');
              XC:=238; YC:=152; I:=1;
              FOR J:=1 TO 20 DO INCHAR[J]:=' ';
              PLOTTEXT(XC,150,CHR(95));
      NOTFINISHED:=TRUE;
      WHILE NOTFINISHED=TRUE DO BEGIN;
               TESTCHAR:=0;
       REPEAT TESTCHAR:=TESTCHR UNTIL TESTCHAR=255;
               CHARIN:=GETCHAR;
        INCHAR[I]:=CHR(CHARIN);
               IF (((CHARIN>31)AND(CHARIN<126)) OR
                                (CHARIN=8) OR (CHARIN=13))
THEN BEGIN;
       IF INCHAR[I]=CHR(13) THEN
                                        NOTFINISHED:=FALSE
      ELSE
      BEGIN
          IF INCHAR[I]=CHR(8) THEN BEGIN
              IF I > 1 THEN BEGIN
                  XC:=XC-8;
                                        SETTEXTCOLOUR(1);
                   PLOTTEXT(XC,YC,CHR(219);
                  PLOTTEXT(XC+8,22 ,CHR(219));
                  SETTEXTCOLOUR(3);
            PLOTTEXT(XC,150,CHR(95));

                  I:=I-1;
                     END;
               END
           ELSE BEGIN;
               IF I<21 THEN BEGIN
```

```
            PLOTTEXT(XC,YC,INCHAR[I]);
            XC:=XC+8; I:=I+1;
                                IF I<>21 THEN BEGIN;
                                SETTEXTCOLOUR(1);
                                PLOTTEXT(XC-
                                8,142,CHR(219));
                                SETTEXTCOLOUR(3);
                                PLOTTEXT(XC,150,CHR(95));
                                        END;
                END;
        END;
          END;
 END;
END;
                        ASSIGN(F,'A:USERNAME.SYS');
                        REWRITE(F);
                        XC:=470; J:=1;
                        REPEAT
                          WRITE(F,INCHAR[J]);
                          IF ((ORD(INCHAR[J])>31)AND
                                (ORD(INCHAR[J])<126)) THEN
        BEGIN;

        PLOTTEXT(XC,236,INCHAR[J]);
                                XC:=XC+8;
                          END;
                          J:=J+1;
                        UNTIL J=21;
                        CLOSE(F);
                        CLEAR_SCR;
        END ELSE BEGIN;
                        ASSIGN(F,'A:USERNAME.SYS');
                        RESET(F);
                        XC:=470;
                        WHILE NOT(EOF(F)) DO BEGIN;
                          READ(F,CH);
                          IF ((ORD(CH)>31)AND
                                (ORD(CH)<126)) THEN BEGIN;
```

```
                                        PLOTTEXT(XC,236,CH);
                                        XC:=XC+8;
                                    END;
                                END;
                                CLOSE(F);
            END;
END; {getname}
{-------------------------------------------------}
PROCEDURE GETDIRANDNAME;
VAR RC:INTEGER;
  F:FILE OF CHAR;
BEGIN;
            GETNAME;
            EXECPROG('\MICK\DIR1',RC);
            ASSIGN(F,'F:STODIR3.SYS');
            RESET(F);
            DINDX:=0;
            WHILE NOT(EOF(F)) DO BEGIN;
                                DINDX:=DINDX+1;
                                READ(F,DIRAREA[DINDX]);
                                IF DINDX=400 THEN EXITPROG(0);
            END;
            CLOSE(F);
END; {getdirandname}
{-------------------------------------------------}
PROCEDURE SETUP;
var r:rect;
BEGIN;
     GRAPHICSON;
     GINPUTON(DevMouse);
     IF (GOErr MOD 65536) <> 0 THEN BEGIN
          GOErr := GOErr MOD 65536;
           WRITELN('Sub-bios error starting up graphics');
          WRITELN('Error code = ',GOErr);
          CASE GOErr OF
            8062h : WRITELN('Graphics in use');
            8064h : WRITELN('Graphics already on');
            8063h : WRITELN('Graphics rejected');
```

```
        OTHERWISE
          WRITELN('Unexpected error while trying to start up
          graphics');
        END;
         EXITPROG(1);
    END;
                              IF GIERR<>0 THEN
          WRITELN('Error starting up input.');
  r.xl := 1; r.yl := 1;
  r.xr := 638; r.yr := 248;    {set up tracking rectangle}
   GINPUTTRACK(DEVMOUSE,R);
  IF GIERR<>0 THEN WRITELN('Error setting up tracking
rectangle');
  SETCOLOURENT(0,lightcyan);
  SETCOLOURENT(1,white);
  SETCOLOURENT(2,darkgrey);
  SETCOLOURENT(3,black);
  SETBORDERCOLOUR(lightblue);
 END; {SETUP}
{-------------------------------------------------}
PROCEDURE SCREEN1;
VAR R1,R2:RECT;
   I,XCOORD:INT;
    VCHAR:CHAR;
BEGIN;
  SETBRUSHCOLOUR(1);
  SETBRUSHSTYLE(solid);
  SETBRUSHMODE(greplace);
  SETPENCOLOUR(2);
  SETPENSTYLE(solid);
  SETPENMODE(greplace);
  MOVETO(0,0);              {draw screen outline}
  LINETO(0,249);
  LINETO(639,249);
  LINETO(639,0);
  LINETO(0,0);
  MOVETO(1,1);
  LINETO(1,248);
```

```
  LINETO(638,248);
  LINETO(638,1);
  LINETO(1,1);
  moveto(5,4);
  lineto(5,190);
  lineto(634,190);                {draw main window}
  lineto(634,4);
  lineto(5,4);
  setrect(r1,6,5,633,189);
  fillrect(r1);
  moveto(0,235);
  lineto(649,235);
  SETRECT(R2,2,236,637,247);
  FILLRECT(R2);
  MOVETO(106,190);
  LINETO(106,235);
  MOVETO(212,190);
  LINETO(212,235);
  MOVETO(318,190);
  LINETO(318,235);
  MOVETO(424,190);
  LINETO(424,235);
  MOVETO(530,190);
  LINETO(530,235);
  SETTEXTCOLOUR(3);
  SETCHARUP(0);
  SETCHARHEIGHT(1);          {write top bar message}
  SETCHARWIDTH(1);
  SETFONT(PRIMARYFONT);
  PLOTTEXT(1,236,' MULTISTORY');
  OLDX:=200; OLDY:=100;
END;  {setupscreen}
{-------------------------------------------------}
PROCEDURE PRINTERICON;
VAR R2,R3:RECT;
BEGIN;
    SETBRUSHCOLOUR(2);
    SETRECT(R2,337,203,407,215);
```

```
      FILLRECT(R2);
      SETRECT(R3,352,215,392,223);
      SETBRUSHCOLOUR(1);
      FILLRECT(R3);
      DEFPENSTYLE(63967);          {draw printer icon}
      SETPENSTYLE(6);
      SETPENCOLOUR(3);
      SETPENCOLOUR2(1);
      MOVETO(353,217);
      LINETO(391,217);
      DEFPENSTYLE(40686);
      MOVETO(353,221);
      LINETO(391,221);
END; {printericon}
{-----------------------------------------------------}
PROCEDURE EXITANDHELPICONS;
BEGIN;
      SETCHARHEIGHT(3);
      SETCHARWIDTH(3);
      PLOTTEXT(570,196,'X');{draw exit and help icons}
      PLOTTEXT(469,196,'?');
END; {exitandhelpicons}
{-----------------------------------------------------}
PROCEDURE TOPLEVEL;    {draw top level screen}
VAR R2,R3:RECT;
BEGIN;
   SCREEN1;
                setbrushcolour(0);
                setrect(r2,545,191,625,201);
                fillrect(r2);
      PLOTTEXT(184,236,'        TOP LEVEL      ');
      SETTEXTCOLOUR(2);
      PLOTTEXT(1,224,'  CREATE      WORD      STORY
PRINTER      HELP      EXIT');
   PLOTTEXT(1,190,'  STORY      PROCESS   INFORMATION ');
      PLOTTEXT(545,190,'MULTISTORY');
                EXITANDHELPICONS;
                PRINTERICON;
```

```
{---------------------------------}
    SETPENSTYLE(1);
    SETPENCOLOUR(2);
    ARCELLIPSE(267,212,35,10,0,0,0);
    ARCELLIPSE(267,212,35,7,0,0,0);
    FLOODFILL(267,212);
    SETBRUSHCOLOUR(3);          {draw eye icon}
    FILLCIRCLE(267,212,6);
    SETBRUSHCOLOUR(1);
    FILLCIRCLE(267,212,1);
{-------------------------------}
    MOVETO(145,201);
    LINETO(145,223);
    LINETO(175,223);
    LINETO(175,201);
    LINETO(145,201);
    FLOODFILL(152,206);
    SETPENSTYLE(6);
    SETPENCOLOUR(3);
    MOVETO(147,220);
    LINETO(173,220);      {draw word process icon}
    MOVETO(173,218);
    LINETO(147,218);
    MOVETO(173,216);
    LINETO(147,216);
    MOVETO(147,214);
    LINETO(173,214);
    MOVETO(147,212);
    LINETO(173,212);
{-----------------------------------}
    SETPENSTYLE(1);
    SETPENCOLOUR(2);      {draw create story icon}
    MOVETO(33,201);
    LINETO(33,223);
    LINETO(73,223);
    LINETO(73,201);
    LINETO(33,201);
    SETBRUSHCOLOUR(2);
```

```
      FLOODFILL(60,210);
      SETBRUSHCOLOUR(1);
      FILLCIRCLE(53,212,2);
      SETTEXTCOLOUR(1);
      SETCHARHEIGHT(1);
      SETCHARWIDTH(1);
      PLOTTEXT(52,213,CHR(223));
      SETTEXTCOLOUR(3);
END; {toplevel}
{-----------------------------------------------------}
PROCEDURE FINISHOFF;
BEGIN;
  SETBORDERCOLOUR(black);
   GRAPHICSOFF;
   GINPUTOFF(devmouse);
   WRITE(CHR(27),'[~G');    {cursor visible}
   WRITE(CHR(27),'c');  {reset to initial state}
END; {FINISHOFF}
{-----------------------------------------------------}
PROCEDURE GETSTORYDIRECTORY;
VAR
    YC,XC,I,J,XCOORD,LINEC,COUNT:INT;
   NOTFINISHED,REALLYFINISHED:BOOLEAN;
    INCHARSTRING:STRING;
   R1:RECT;
BEGIN;
      SETTEXTCOLOUR(3); setcharup(0); setcharheight(1);
      setcharwidth(1); setfont(primaryfont);
      PLOTTEXT(35,160,'You already have the following stories :-');
              XC:=35; YC:=130; I:=1; LINEC:=0;
              SETTEXTCOLOUR(2);
              WHILE I<=DINDX DO BEGIN;
                                  COUNT:=0;
                                  WHILE
((DIRAREA[I]<>'.')AND(I<=DINDX)AND
                                          (COUNT<8))       DO BEGIN;
                                  COUNT:=COUNT+1;
                                  PLOTTEXT(XC,YC,DIRAREA[I]);
```

```
                    I:=I+1;
                    XC:=XC+8;
                    END;
                    IF ((COUNT<>8)AND(I<=DINDX))
THEN
                        FOR J:=COUNT TO 7 DO
BEGIN
                            PLOTTEXT(XC,YC,' ');
                            XC:=XC+8;
                            I:=I+1;
                        END;
            XC:=XC+40;
            LINEC:=LINEC+1;
            IF LINEC=5 THEN BEGIN;
                    XC:=35;
                    YC:=YC-10;
                    LINEC:=0;
             END;
            END;
             REALLYFINISHED:=FALSE;
            SETTEXTCOLOUR(3);
    PLOTTEXT(35,50,'Enter the name of the story you wish to
create.');
    PLOTTEXT(80,32,'[      ]');
            XC:=88; YC:=32; I:=1;
            FOR J:=1 TO 10 DO INCHAR[J]:=' ';
            PLOTTEXT(XC,30,CHR(95));
            WHILE REALLYFINISHED=FALSE DO BEGIN;
    NOTFINISHED:=TRUE;
    WHILE NOTFINISHED=TRUE DO BEGIN;
     READSCREEN;
     IF TESTCHAR=0 THEN BEGIN
        IF ICODE='F' THEN BEGIN;
                        NOTFINISHED:=FALSE;
                        REALLYFINISHED:=TRUE;
                        END
                    END
    ELSE BEGIN;
```

```
    END;
            INCHARSTRING:=''; NEWCHARSTRING:='';
     IF ICODE='X' THEN BEGIN;
                          SETRECT(R1,250,10,630,30);
                          SETBRUSHCOLOUR(1);
                          FILLRECT(R1);
        FOR J:=I TO 10 DO INCHAR[J]:=' ';
                          FOR J:=1 TO (I-1) DO
INCHARSTRING:=

          CONCAT(INCHARSTRING,INCHAR[J]);
                          FOR J:=1 TO 8 DO
NEWCHARSTRING:=

          CONCAT(NEWCHARSTRING,INCHAR[J]);
                          NEWCHARSTRING:='';

NEWCHARSTRING:=CONCAT('A:',INCHARSTRING,'.STY');
                          IF
(NOT(CHECKFN(NEWCHARSTRING)) OR
                                (FSTAT(NEWCHARSTRING)))
THEN BEGIN;
                                SETTEXTCOLOUR(2);
                                PLOTTEXT(250,10,'You must
enter a valid new file name.');
                                SETTEXTCOLOUR(3); END
                          ELSE BEGIN;
REALLYFINISHED:=TRUE; END;
                         END;
            END;
             IF ICODE='X' THEN BEGIN;
         XCOORD:=360;
         FOR I:=1 TO 8 DO BEGIN;
             PLOTTEXT(XCOORD,236,INCHAR[I]);
             XCOORD:=XCOORD+8;
            END;
         END;
```

```
END; {getstorydirectory}

{-------------------------------------------------}
PROCEDURE CREATE_TOP_ICONS;
VAR R1:RECT;
  I:INTEGER;
BEGIN;
    SETBRUSHCOLOUR(1);
    SETRECT(R1,184,236,469,246);
    FILLRECT(R1);
    SETTEXTCOLOUR(3);
    PLOTTEXT(100,236,'      CREATE LEVEL     ');
    SETBRUSHCOLOUR(0);
    SETRECT(R1,2,191,317,234);
    FILLRECT(R1);
    SETRECT(R1,545,191,630,200);
    FILLRECT(R1);
          SETTEXTCOLOUR(2);
    PLOTTEXT(564,190,'BACK');
          SETTEXTCOLOUR(3);
END; {create_top_icons}
{-------------------------------------------------}
PROCEDURE GETTYPE;
BEGIN;
    READSCREEN;
    IF (OLDY<143) AND (OLDY>129) THEN BEGIN;
      IF (OLDX<173) AND (OLDX>59) THEN

        STYPE:=1;
      IF (OLDX<373) AND (OLDX>259) THEN

        STYPE:=2;
      IF (OLDX<573) AND (OLDX>459) THEN

        STYPE:=3;
    END
    ELSE BEGIN;
    IF (OLDY<93) AND (OLDY>79) THEN BEGIN;
```

```
          IF (OLDX<173) AND (OLDX>59) THEN

            STYPE:=4;
          IF (OLDX<373) AND (OLDX>259) THEN

            STYPE:=5;
          IF (OLDX<573) AND (OLDX>459) THEN

            STYPE:=6;
          END;
     END;
END;   {gettype}
{-------------------------------------------------}
PROCEDURE GETYCOORD(VAR ATTRIB,YCOORD:INT);
BEGIN;
            CASE ATTRIB OF
             1:YCOORD:=160;
             2:YCOORD:=150;
             3:YCOORD:=140;
             4:YCOORD:=130;
             5:YCOORD:=120;
             6:YCOORD:=110;
             7:YCOORD:=100;
             8:YCOORD:=90;
             9:YCOORD:=80;
             10:YCOORD:=70;
             11:YCOORD:=60;
             12:YCOORD:=50;
             13:YCOORD:=40;
            END;
END; {getycoord}
{-------------------------------------------------}
PROCEDURE GETATTRIBUTES;
VAR
XCOORD,YCOORD,COUNT,SECTOR,TRACK,ATTRIB,SELECT:I
NT;
  R1:RECT;
   INCHAR:CHAR;
```

```
BEGIN;
        RESET(F);
        READSCREEN;
        IF ((OLDX>182) AND (OLDX<326)
                        AND (OLDY>40) AND (OLDY<170))
THEN BEGIN;
                        IF OLDX<206 THEN SELECT:=1
                         ELSE IF OLDX<230 THEN SELECT:=2
                          ELSE IF OLDX<254 THEN SELECT:=3
                           ELSE IF OLDX<278 THEN
SELECT:=4
                            ELSE IF OLDX<302 THEN
SELECT:=5
                            ELSE SELECT:=6;
                        IF OLDY<50 THEN ATTRIB:=13
                        ELSE IF OLDY<60 THEN ATTRIB:=12
                         ELSE IF OLDY<70 THEN
ATTRIB:=11
                          ELSE IF OLDY<80 THEN
ATTRIB:=10
                           ELSE IF OLDY<90 THEN
ATTRIB:=9
                          ELSE IF OLDY<100 THEN
ATTRIB:=8
                           ELSE IF OLDY<110 THEN
ATTRIB:=7
                            ELSE IF OLDY<120 THEN
ATTRIB:=6
                             ELSE IF OLDY<130 THEN
ATTRIB:=5
                              ELSE IF OLDY<140 THEN
ATTRIB:=4
                                  ELSE IF OLDY<150 THEN
ATTRIB:=3
                                   ELSE IF OLDY<160 THEN
ATTRIB:=2
                                    ELSE ATTRIB:=1;
                        XCOORD:=340;
```

```
                    GETYCOORD(ATTRIB,YCOORD);

          SETRECT(R1,340,YCOORD,630,YCOORD+10);
                    SETBRUSHCOLOUR(1);
                    FILLRECT(R1);
                    IF SELECT=6 THEN BEGIN;
                     ATTRIBUTE[ATTRIB]:=0;
                     END
                    ELSE BEGIN;
                     COUNT:=0;
                     SECTOR:=(ATTRIB-1)*5;
                     WHILE COUNT<SECTOR DO
BEGIN;
                          REPEAT READ(F,INCHAR)
UNTIL
                    INCHAR=CHR(13);
                          READ(F,INCHAR);
                          COUNT:=COUNT+1;
                     END;
                     TRACK:=SELECT-1; COUNT:=0;
                     WHILE COUNT<TRACK DO BEGIN;
                          REPEAT READ(F,INCHAR)
UNTIL

          INCHAR=CHR(13);
                          READ(F,INCHAR);
                          COUNT:=COUNT+1;
                     END;
                     READ(F,INCHAR);
                     WHILE INCHAR<>CHR(13) DO
BEGIN;

          PLOTTEXT(XCOORD,YCOORD,INCHAR);
                          XCOORD:=XCOORD+8;
                          READ(F,INCHAR);
                     END;
                     ATTRIBUTE[ATTRIB]:=SELECT;
                    END;
```

```
            END;
END; {getattributes}
{-------------------------------------------------}
PROCEDURE CREATE2_4;
VAR  Y:INT;
  R1:RECT;
BEGIN;
            SETTEXTCOLOUR(1);
            PLOTTEXT(270,236,CHR(219));
            SETTEXTCOLOUR(3);
            PLOTTEXT(270,236,'5');
            PLOTTEXT(10,179,'Choose the attributes of your
character (on a 1 to 5 scale).');
            PLOTTEXT(10,160,'1)HEALTH');
            PLOTTEXT(10,150,'2)ATTRACTIVENESS');
            PLOTTEXT(10,140,'3)CALMNESS');
            PLOTTEXT(10,130,'4)DETERMINATION');
            PLOTTEXT(10,120,'5)FRIENDLINESS');
            PLOTTEXT(10,110,'6)HUMOUR');
            PLOTTEXT(10,100,'7)INTELLIGENCE');
            PLOTTEXT(10,90,'8)IMAGINATION');
            PLOTTEXT(10,80,'9)KINDNESS');
            PLOTTEXT(10,70,'10)SELF-CONFIDENCE');
            PLOTTEXT(10,60,'11)SKILLFULNESS');
            PLOTTEXT(10,50,'12)TRUTHFULNESS');
            PLOTTEXT(10,40,'13)STRENGTH');
            Y:=160;
            WHILE Y>39 DO BEGIN;
                        PLOTTEXT(190,Y,'1  2  3  4  5 *');
                        Y:=Y-10;
            END;
            PLOTTEXT(10,6,'When you have finished point to the
CONTINUE box....');
            SETRECT(R1,452,6,532,36);
            SETBRUSHCOLOUR(2);
            FILLRECT(R1);
            SETTEXTCOLOUR(1);
            PLOTTEXT(460,16,'CONTINUE');
```

```
            SETTEXTCOLOUR(3);
            FOR Y:=1 TO 13 DO ATTRIBUTE[Y]:=0;
            ASSIGN(F,'C:ATTFILE.STO');
            SETTEXTCOLOUR(2);
            REPEAT GETATTRIBUTES UNTIL ((ICODE='F') OR
                              ((OLDX>452) AND (OLDX<532)
                             AND    (OLDY>6) AND (OLDY<36)));
            SETTEXTCOLOUR(3);
            IF ICODE<>'F' THEN BEGIN;
                              ENDCREATE:=TRUE;
                              LEFTB:=FALSE;
                              ICODE:='F';
            END;
END; {create2_4}
{-------------------------------------------------}
PROCEDURE GETCHARACTER;
VAR  CHARSTRING,TEMPSTRING:STRING;
   XCOORD,YCOORD:INT;
    INPUTCHAR:CHAR;
  R1:RECT;
BEGIN;
            TEMPSTRING:='';
            WHILE ((WORKAREA2[TEMP]<>',') AND
                              (WORKAREA2[TEMP]<>'>')) DO
BEGIN;

            TEMPSTRING:=CONCAT(TEMPSTRING,WORKAREA
2[TEMP]);
                              TEMP:=TEMP+1;
            END;
            CHARACTER:=TEMPSTRING;
            CHARSTRING:=CONCAT('C:CHAR',TEMPSTRING,'.
STO');
            ASSIGN(F,CHARSTRING);
            RESET(F);
            XCOORD:=75; YCOORD:=120;
            WHILE NOT EOF(F) DO BEGIN;
                              READ(F,INPUTCHAR);
```

```
                                IF INPUTCHAR<>CHR(13) THEN
BEGIN;

PLOTTEXT(XCOORD,YCOORD,INPUTCHAR);
                                 XCOORD:=XCOORD+8;
                                  END
                                ELSE BEGIN;
                                  XCOORD:=75;
                                  YCOORD:=YCOORD-10;
                                 READ(F,INPUTCHAR);
                                END;
            END;
            IF WORKAREA2[TEMP] = '>' THEN TEMP:=2
                                ELSE TEMP:=TEMP+1;
            READSCREEN;
            SETRECT(R1,61,61,589,139);
            SETBRUSHCOLOUR(1);
            FILLRECT(R1);
END; {getcharacter}
{-------------------------------------------------}
PROCEDURE CREATE2_3;
VAR R1:RECT;
BEGIN;
            SETTEXTCOLOUR(1);
            PLOTTEXT(270,236,CHR(219));
            SETTEXTCOLOUR(3);
            PLOTTEXT(270,236,'4');
   PLOTTEXT(10,175,'Story type chosen = ');
   CASE STYPE OF
      1:PLOTTEXT(170,175,'REVENGE');
      2:PLOTTEXT(170,175,'GROWING UP');
      3:PLOTTEXT(170,175,'FANTASY/HORROR');
      4:PLOTTEXT(170,175,'ANIMALS');
      5:PLOTTEXT(170,175,'LOVE');
      6:PLOTTEXT(170,175,'CONFLICT');
   END;
   MOVETO(5,174);
            LINETO(634,174);
```

```
            PLOTTEXT(10,150,'Here is a main character for your
story :-');
            SETRECT(R1,542,10,622,40);
            SETBRUSHCOLOUR(2);
            FILLRECT(R1);
            SETRECT(R1,452,10,532,40);
            FILLRECT(R1);
            SETTEXTCOLOUR(1);
            PLOTTEXT(460,20,'CONTINUE');
            PLOTTEXT(546,12,'CHARACTER');
            PLOTTEXT(558,27,'ANOTHER');
            SETTEXTCOLOUR(3);
            MOVETO(60,140);
            LINETO(590,140);
            LINETO(590,60);
            LINETO(60,60);
            LINETO(60,140);
   TEMP:=2;
            REPEAT GETCHARACTER UNTIL ((ICODE='F') OR
                  ((OLDX>452) AND (OLDX<532)
         AND (OLDY>10) AND (OLDY<40)));
            IF ICODE<>'F' THEN BEGIN;
                              CLEAR_SCR;
                              REPEAT CREATE2_4 UNTIL
ICODE='F';
                              CLEAR_SCR;
                              IF LEFTB THEN ICODE:='X';
            END;
END; {create2_3}
{-----------------------------------------------------}
PROCEDURE DISPLAY_SIT;
VAR SIT,SITSTRING:STRING;
   XCOORD,YCOORD,I:INT;
   INPUTCHAR:CHAR;
  R1:RECT;
  ENDFOUND:BOOLEAN;
BEGIN;
            ENDFOUND:=FALSE;
```

```
SETRECT(R1,61,61,589,139);
SETBRUSHCOLOUR(1);
FILLRECT(R1);
IF INPNO>0.5 THEN BEGIN;
                IF RN=8 THEN BEGIN;
                                RN:=1;
                            END
                  ELSE BEGIN;
                                RN:=RN+1;
                            END
                END
ELSE BEGIN;
                IF RN=1 THEN BEGIN;
                                RN:=8;
                            END
                  ELSE BEGIN;
                                RN:=RN-1;
                            END;
                END;
STR(RN,SIT);
SITUATION:=RN;
CASE STYPE OF
  1:SITSTRING:=CONCAT('C:SIT',SIT,'.','REV');
  2:SITSTRING:=CONCAT('C:SIT',SIT,'.','GRO');
  3:SITSTRING:=CONCAT('C:SIT',SIT,'.','FHR');
  4:SITSTRING:=CONCAT('C:SIT',SIT,'.','ANI');
  5:SITSTRING:=CONCAT('C:SIT',SIT,'.','LOV');
  6:SITSTRING:=CONCAT('C:SIT',SIT,'.','CON');
END;
ASSIGN(F,SITSTRING);
RESET(F);
XCOORD:=75; YCOORD:=120; I:=1;
WHILE NOT EOF(F) DO BEGIN;
                READ(F,INPUTCHAR);
                IF INPUTCHAR='<' THEN
ENDFOUND:=TRUE;
                IF NOT ENDFOUND THEN BEGIN;
```

```
                                        IF INPUTCHAR<>CHR(13)
THEN BEGIN;

PLOTTEXT(XCOORD,YCOORD,INPUTCHAR);
                                         XCOORD:=XCOORD+8;
                                        END
                                        ELSE BEGIN;
                                         XCOORD:=75;
                                         YCOORD:=YCOORD-10;
                                         READ(F,INPUTCHAR);
                                        END
                                    END ELSE BEGIN;

            WORKAREA2[I]:=INPUTCHAR;
                                        I:=I+1;
                                    END;
            END;
            READSCREEN;
END; {display_sit}
{-------------------------------------------------}
PROCEDURE GETSTOSIT;
VAR R1:RECT;
BEGIN;
            MOVETO(60,140);
            LINETO(590,140);
            LINETO(590,60);
            LINETO(60,60);
            LINETO(60,140);
            SETRECT(R1,542,10,622,40);
            SETBRUSHCOLOUR(2);
            FILLRECT(R1);
            SETRECT(R1,452,10,532,40);
            FILLRECT(R1);
            SETTEXTCOLOUR(1);
            PLOTTEXT(460,20,'CONTINUE');
            PLOTTEXT(546,12,'SITUATION');
            PLOTTEXT(558,27,'ANOTHER');
            SETTEXTCOLOUR(3);
```

```
            INPNO:=RAND;
            IF INPNO<0.125 THEN RN:=1
                              ELSE IF INPNO<0.25 THEN RN:=2
                                ELSE IF INPNO<0.375 THEN RN:=3
                                     ELSE IF INPNO<0.5 THEN
RN:=4
                                                ELSE IF INPNO<0.625
THEN RN:=5
                                                     ELSE IF
INPNO<0.75 THEN

            RN:=6
                                                           ELSE IF
INPNO<0.875
                                                                THEN
RN:=7
                                                                ELSE
RN:=8;
            INPNO:=RAND;
            REPEAT DISPLAY_SIT UNTIL ((ICODE = 'F') OR
                 ((OLDX>452) AND (OLDX<532)
         AND (OLDY>10) AND (OLDY<40)));
    IF ICODE<>'F' THEN BEGIN;
                              TEMP:=2;
                              CLEAR_SCR;
                              REPEAT CREATE2_3 UNTIL
ICODE='F';
                              IF LEFTB THEN ICODE:='X';
                              CLEAR_SCR;
            END;
END; {GETSTOSIT}
{-------------------------------------------------}
PROCEDURE CREATE2_2;
BEGIN;
              SETTEXTCOLOUR(1);
              PLOTTEXT(270,236,CHR(219));
              SETTEXTCOLOUR(3);
              PLOTTEXT(270,236,'3');
```

```
    PLOTTEXT(10,175,'Story type chosen = ');
    CASE STYPE OF
      1:PLOTTEXT(170,175,'REVENGE');
      2:PLOTTEXT(170,175,'GROWING UP');
      3:PLOTTEXT(170,175,'FANTASY/HORROR');
      4:PLOTTEXT(170,175,'ANIMALS');
      5:PLOTTEXT(170,175,'LOVE');
      6:PLOTTEXT(170,175,'CONFLICT');
    END;
    MOVETO(5,174);
          LINETO(634,174);
          PLOTTEXT(10,150,'Here is a situation for your story :-
');
          GETSTOSIT;
END; {CREATE2_2}
{-------------------------------------------------}
PROCEDURE CREATE2_1;
VAR I,J,XCOORD,YCOORD:INTEGER;
BEGIN;
          SETTEXTCOLOUR(1);
          PLOTTEXT(270,236,CHR(219));
          SETTEXTCOLOUR(3);
          PLOTTEXT(270,236,'2');
    PLOTTEXT(10,170,'Select one of the following story types :-');
    PLOTTEXT(89,131,'REVENGE');
    PLOTTEXT(277,131,'GROWING UP');
    PLOTTEXT(461,131,'FANTASY/HORROR');
    PLOTTEXT(89,81,'ANIMALS');
    PLOTTEXT(301,81,'LOVE');
    PLOTTEXT(485,81,'CONFLICT');
    XCOORD:=60; YCOORD:=130;
    FOR I:=1 TO 2 DO BEGIN;
        FOR J:=1 TO 3 DO BEGIN
          MOVETO(XCOORD,YCOORD);
          LINETO(XCOORD+112,YCOORD);
          LINETO(XCOORD+112,YCOORD+12);
          LINETO(XCOORD,YCOORD+12);
          LINETO(XCOORD,YCOORD);
```

```
            XCOORD:=XCOORD+200;
          END;
          YCOORD:=80; XCOORD:=60;
      END;
      STYPE:=9;
      REPEAT GETTYPE UNTIL (ICODE='F') OR (STYPE<>9);
      IF ICODE<>'F' THEN BEGIN;
           CLEAR_SCR;
           REPEAT CREATE2_2 UNTIL ICODE='F';
           IF LEFTB THEN ICODE:='X';
           CLEAR_SCR;
          END;
END; {create2_1}
{-----------------------------------------------------}
PROCEDURE CREATESTORY;
BEGIN;  CLEAR_SCR;
      CREATE_TOP_ICONS;
              SETTEXTCOLOUR(1);
              PLOTTEXT(270,236,CHR(219));
              SETTEXTCOLOUR(3);
              PLOTTEXT(270,236,'1');
      GETSTORYDIRECTORY;
      CLEAR_SCR;
      IF ICODE<> 'F' THEN BEGIN;
               REPEAT CREATE2_1 UNTIL ICODE='F';
               IF LEFTB THEN ICODE:='X';
               CLEAR_SCR;
              END;
END;  {create story}
{-----------------------------------------------------}
PROCEDURE CREATE2_5;
VAR  I,xc:int;
   F1:TEXT;
   OUT,ST,ST1:STRING;
BEGIN;
            ENDCREATE:=FALSE; XC:=196;
            MOVETO(20,160);
            LINETO(280,160);
```

```
            LINETO(280,120);
            LINETO(20,120);
            LINETO(20,160);
            PLOTTEXT(30,140,'The new story file.. ');
            SETTEXTCOLOUR(2);
            FOR I:=1 TO 8 DO BEGIN;
                        DINDX:=DINDX+1;
                        DIRAREA[DINDX]:=INCHAR[I];

            PLOTTEXT(XC,140,DIRAREA[DINDX]);
                        XC:=XC+8;
            END;
            SETTEXTCOLOUR(3);
            PLOTTEXT(30,130,'has been created.');
            ASSIGN(F1,NEWCHARSTRING);
            REWRITE(F1);
            STR(STYPE,ST);
            OUT:=CONCAT('storytype(',ST,').');
            WRITELN(F1,OUT);
            STR(SITUATION,ST);
            OUT:=CONCAT('storysit(',ST,').');
            WRITELN(F1,OUT);
            OUT:=CONCAT('storychar(',CHARACTER,').');
            WRITELN(F1,OUT);
            FOR I:=1 TO 13 DO BEGIN;
                        STR(ATTRIBUTE[I],ST);
                        STR(I,ST1);
                        OUT:=CONCAT('att',ST1,'(',ST,').');
                        WRITELN(F1,OUT);
            END;
            CLOSE(F1);
END; {create2_5}
{-------------------------------------------------}
PROCEDURE WP_TOP_ICONS;
VAR R1:RECT;
BEGIN;
            CLEAR_SCR;
            SETBRUSHCOLOUR(1);
```

```
            SETRECT(R1,184,236,469,246);
            FILLRECT(R1);
            SETTEXTCOLOUR(3);
            PLOTTEXT(100,236,'     WORD PROCESSOR   ');
            SETBRUSHCOLOUR(0);
            SETRECT(R1,2,191,211,234);
            FILLRECT(R1);
            SETRECT(R1,545,191,630,200);
            FILLRECT(R1);
            SETTEXTCOLOUR(2);
            PLOTTEXT(564,190,'BACK');
            SETTEXTCOLOUR(3);
END; {wp_top_icons}
{-------------------------------------------------}
PROCEDURE PLOTCUR;
BEGIN;
            SETTEXTCOLOUR(2);
            PLOTTEXT(XC-4,YC,CHR(179));
            SETTEXTCOLOUR(3);
END; {plotcur}
{-------------------------------------------------}
PROCEDURE DELETECUR;
BEGIN;
            SETTEXTCOLOUR(1);
            PLOTTEXT(XC-4,YC,CHR(179));
            SETTEXTCOLOUR(3);
END; {deletecur}
{-------------------------------------------------}
PROCEDURE DELETELINE(VAR DY:INTEGER);
VAR R1:RECT;
BEGIN;
            SETRECT(R1,20,DY,590,DY+10);
            FILLRECT(R1);
END; {deleteline}
{-------------------------------------------------}
PROCEDURE SCROLLSCREEN1;
VAR FINISHED,LINEFINISHED:BOOLEAN;
   DY,COUNT,XT:INTEGER;
```

```
  R1:RECT;
BEGIN;
          FINISHED:=FALSE;
          DY:=170; TINDEX:=PBEGIN;
          SETBRUSHCOLOUR(1);
          WHILE NOT FINISHED DO BEGIN;
                         DELETELINE(DY);
                         XT:=20;
                         LINEFINISHED:=FALSE; COUNT:=0;
                         WHILE NOT LINEFINISHED DO
BEGIN;
                          IF (TINDEX>=WPENDMARKER)
THEN BEGIN;
                           LINEFINISHED:=TRUE;
                           FINISHED:=TRUE;
                         END ELSE BEGIN;
                          IF WPAREA[TINDEX]=CHR(13)
THEN BEGIN;
                                LINEFINISHED:=TRUE;
                                TINDEX:=TINDEX+2;
                                COUNT:=0;
                         END ELSE BEGIN;
                                IF COUNT>=69 THEN BEGIN;
                                      COUNT:=0;

          LINEFINISHED:=TRUE;
                                END
                                ELSE BEGIN;

PLOTTEXT(XT,DY,WPAREA[TINDEX]);
                                 TINDEX:=TINDEX+1;
                                 COUNT:=COUNT+1;
                                 XT:=XT+8;
                                END;
                          END;
                         END;
                        END;
                        DY:=DY-10;
```

```
            IF ((DY=10)OR(TINDEX>=WPENDMARKER)) THEN
FINISHED:=TRUE;
        END;
        IF DY>10 THEN BEGIN;
                        SETRECT(R1,20,20,580,DY+10);
                        FILLRECT(R1);
        END;
END; {scrollscreen1}
{-----------------------------------------------}
PROCEDURE MOVESCREEN;
VAR NOOFCHARS:INTEGER;
        R1:RECT;
BEGIN;
        IF XC=580 THEN NOOFCHARS:=70
                        ELSE NOOFCHARS:=(XC-20)DIV 8;
        IF WPAREA[WPINDEX-(NOOFCHARS+1)]=CHR(10)
THEN BEGIN;
                        PBEGIN:=WPINDEX-NOOFCHARS;
                        YC:=170;
        END ELSE BEGIN;
                        PBEGIN:=WPINDEX-NOOFCHARS-
70;
                        YC:=160;
        END;
        SETRECT(R1,20,20,580,180);
        FILLRECT(R1);
        SCROLLSCREEN1;
END; {movescreen}
{-----------------------------------------------}
PROCEDURE NORMALINPUT;
BEGIN;
        DELETECUR;
        CH:=CHR(CHARIN);
        WPAREA[WPINDEX]:=CH;
        WPINDEX:=WPINDEX+1;
        WPENDMARKER:=WPENDMARKER+1;
        IF (((CH=CHR(13))AND(YC=20))OR
          ((XC>=580)AND(YC=20))) THEN MOVESCREEN;
```

```
            IF CHARIN=13 THEN BEGIN;
                              WPAREA[WPINDEX]:=CHR(10);
                              WPINDEX:=WPINDEX+1;

            WPENDMARKER:=WPENDMARKER+1;
                              XC:=20;
                              YC:=YC-10;
            END ELSE BEGIN;
                              PLOTTEXT(XC,YC,CH);
                              IF XC<580 THEN XC:=XC+8
                              ELSE BEGIN;
                                XC:=20;
                                YC:=YC-10;
                              END;
            END;
            PLOTCUR;
END; {normalinput}
{-----------------------------------------------}
PROCEDURE INSERTCHAR;
BEGIN;
            DELETECUR;
            CH:=CHR(CHARIN);
            IF NOT(((CH=CHR(13))AND (YC=20))OR
                              ((XC>=580)AND(YC=20))) THEN
BEGIN;
            TINDEX:=TINDEX+1;
            TEMPWP[TINDEX]:=CH;
            IF CHARIN=13 THEN BEGIN;
                              TINDEX:=TINDEX+1;
                              TEMPWP[TINDEX]:=CHR(10);
                              XC:=20; YC:=YC-10;
            END ELSE          PLOTTEXT(XC,YC,CH);
            IF XC<580 THEN XC:=XC+8
            ELSE BEGIN;
                              XC:=20;
                              YC:=YC-10;
            END;
            END;
```

```
            PLOTCUR;
            TESTCHAR:=0;
            READSCREEN;
END; {INSERTCHAR}
{-----------------------------------------------}
PROCEDURE INSERTMODE;
VAR R1:RECT;
   STARTWP,ENDWP,I,J,XT,YT:INT;
BEGIN;
            SETBRUSHCOLOUR(1);
            SETRECT(R1,XC,YC,610,YC+10);
            FILLRECT(R1);
            SETRECT(R1,20,10,610,YC);
            FILLRECT(R1);
            I:=WPINDEX; XT:=20;
            SETTEXTCOLOUR(2);
            WHILE ((I<=WPENDMARKER)AND
                              (WPAREA[I]<>CHR(13))AND
                              ((I-WPINDEX)<70)) DO BEGIN;
                             PLOTTEXT(XT,10,WPAREA[I]);
                             XT:=XT+8;
                             I:=I+1;
            END;
            SETTEXTCOLOUR(3);
            TINDEX:=0; LEFTB:=FALSE; RIGHTB:=FALSE;
            REPEAT INSERTCHAR UNTIL LEFTB OR RIGHTB;
            SETRECT(R1,20,10,610,20);
            SETBRUSHCOLOUR(1);
            FILLRECT(R1);
            I:=WPINDEX; XT:=XC; YT:=YC;
            WHILE ((YT>10)AND(I<=WPENDMARKER))DO
BEGIN;
                             IF WPAREA[I]=CHR(13) THEN
BEGIN;
                              XT:=12; YT:=YT-10; I:=I+1;
                              END ELSE
PLOTTEXT(XT,YT,WPAREA[I]);
                             IF XT<600 THEN XT:=XT+8
```

```
                        ELSE BEGIN;
                                XT:=20; YT:=YT-10;
                          END;

                        I:=I+1;
            END;
            FOR I:=WPENDMARKER DOWNTO WPINDEX DO
               WPAREA[I+TINDEX]:=WPAREA[I];
            J:=1;
            FOR I:=WPINDEX TO (WPINDEX+TINDEX-1) DO
BEGIN;
               WPAREA[I]:=TEMPWP[J];
              J:=J+1;
            END;
            WPINDEX:=WPINDEX+TINDEX;
            WPENDMARKER:=WPENDMARKER+TINDEX;
END; {INSERTMODE}
{-----------------------------------------------}
PROCEDURE SCROLLSCREEN;
BEGIN;
            SCROLLSCREEN1;
            IF ((WPINDEX>PBEGIN)AND(WPINDEX<TINDEX))
THEN BEGIN;
                        PLOTCUR; END
            ELSE BEGIN;
                        XC:=20; YC:=170;
                        WPINDEX:=PBEGIN;
                        PLOTCUR;
            END;
END; {scrollscreen}
{-----------------------------------------------}
FUNCTION CHECKEND:BOOLEAN;
VAR  COUNT,TINDEX:INTEGER;
  FINISHED:BOOLEAN;
BEGIN;
            FINISHED:=FALSE;
            COUNT:=0;
            TINDEX:=PBEGIN;
```

```
            CHECKEND:=FALSE;
            WHILE NOT FINISHED DO BEGIN;
                                IF TINDEX>=WPENDMARKER THEN
BEGIN;
                                  FINISHED:=TRUE;
                                  CHECKEND:=TRUE;
                                END;
                                IF
((WPAREA[TINDEX]=CHR(13))OR(COUNT>=69)) THEN
                                  FINISHED:=TRUE;
                                COUNT:=COUNT+1;
                                TINDEX:=TINDEX+1;
            END;
END; {checkend}
{-----------------------------------------------}
PROCEDURE FINDPREV;
VAR TINDEX,LASTLINE,COUNT:INTEGER;
BEGIN;
            TINDEX:=PBEGIN-3;
            WHILE
((WPAREA[TINDEX]<>CHR(10))AND(TINDEX>1)) DO
                                TINDEX:=TINDEX-1;
            IF TINDEX<>1 THEN TINDEX:=TINDEX+1;
            COUNT:=0; LASTLINE:=TINDEX;
            WHILE TINDEX<(PBEGIN-2) DO BEGIN;
                                IF WPAREA[TINDEX]=CHR(13)
THEN BEGIN;
                                  COUNT:=0;
                                  TINDEX:=TINDEX+1;
                                  LASTLINE:=TINDEX+1;
                                END;
                                IF COUNT=69 THEN BEGIN;
                                  LASTLINE:=TINDEX+1;
                                  COUNT:=0;
                                END;
                                TINDEX:=TINDEX+1;
                                COUNT:=COUNT+1;
            END;
```

```
            PBEGIN:=LASTLINE;
END; {findprev}
{-----------------------------------------------}
PROCEDURE SCROLLDOWNSETUP;
VAR TINDEX,COUNT,LASTLINE:INTEGER;
  FINISHED:BOOLEAN;
BEGIN;
            TINDEX:=PBEGIN;
            WHILE ((WPAREA[TINDEX]<>CHR(10)) AND
               (TINDEX>1)) DO TINDEX:=TINDEX-1;
            IF TINDEX<>1 THEN TINDEX:=TINDEX+1;
            COUNT:=0;
            FINISHED:=FALSE;
            LASTLINE:=TINDEX;
            WHILE NOT FINISHED DO BEGIN;
                              IF WPAREA[TINDEX]=CHR(13)
THEN BEGIN;
                                COUNT:=0;
                                TINDEX:=TINDEX+1;
                                LASTLINE:=TINDEX+1;
                                IF LASTLINE>PBEGIN THEN
FINISHED:=TRUE;
                              END;
                              IF COUNT>69 THEN BEGIN;
                                COUNT:=0;
                                LASTLINE:=TINDEX+1;
                                IF LASTLINE>PBEGIN THEN
FINISHED:=TRUE;
                              END;
                              TINDEX:=TINDEX+1;
COUNT:=COUNT+1;
            END;
            PBEGIN:=LASTLINE;
END; {scrolldownsetup}
{-----------------------------------------------}
PROCEDURE SCROLL;
BEGIN;
IF ((LEFTB)AND(PBEGIN<>1)) THEN BEGIN;
```

```
            IF WPAREA[PBEGIN-1]<>CHR(10) THEN
PBEGIN:=PBEGIN-70
                                ELSE FINDPREV;
            YC:=YC-10;
            SCROLLSCREEN;
            END;
IF ((RIGHTB)AND(NOT CHECKEND)) THEN BEGIN;
            SCROLLDOWNSETUP;
            YC:=YC+10;
            SCROLLSCREEN;
            END;
END; {scroll}
{-----------------------------------------------}
PROCEDURE CURSORMOVE;
VAR
LINENO,COLNO,TINDEX,LINE,COUNT,ENDCOL,I:INTEGER;
  FINISHED:BOOLEAN;
BEGIN;
                                OLDX:=OLDX-20;
                                IF OLDX<8 THEN OLDX:=0
                                      ELSE OLDX:=OLDX DIV 8;
                                COLNO:=OLDX+1;
                                OLDX:=20+(OLDX*8);
                                OLDY:=OLDY-20;
                                IF OLDY<10 THEN OLDY:=0
                                      ELSE OLDY:=OLDY DIV 10;
                                LINENO:=16-OLDY;
                                OLDY:=20+(OLDY*10);
                                DELETECUR;
                                TINDEX:=PBEGIN;
                                LINE:=1;
                                COUNT:=0;
                                WHILE LINE<LINENO DO BEGIN;
                                      IF
WPAREA[TINDEX]=CHR(13) THEN BEGIN;
                                                LINE:=LINE+1;
                                                COUNT:=0;
                                                TINDEX:=TINDEX+1;
```

```
                    END;
                    IF COUNT=69 THEN BEGIN;
                        LINE:=LINE+1;
                        COUNT:=0;
                    END;
                    TINDEX:=TINDEX+1;
                    COUNT:=COUNT+1;
                END;
                ENDCOL:=0;
                FINISHED:=FALSE;
                I:=TINDEX;
                WHILE NOT FINISHED DO BEGIN;
                    IF WPAREA[I]=CHR(13)
THEN BEGIN;

        ENDCOL:=I+1;

        FINISHED:=TRUE;
                                                END;
                IF I=(TINDEX+COLNO)-1 THEN
FINISHED:=TRUE;
                    I:=I+1;
                END;
                IF ENDCOL=0 THEN BEGIN;
                    WPINDEX:=I-1;
                    XC:=OLDX;
                    END ELSE BEGIN;
                    WPINDEX:=ENDCOL-2;
                    XC:=12+((ENDCOL-
TINDEX)*8);
                END;
                YC:=OLDY;
                PLOTCUR;
END; {cursormove}
{-----------------------------------------------}
PROCEDURE INPUTSCREEN;
BEGIN;
        READSCREEN;
```

```
            IF TESTCHAR=0 THEN BEGIN;
                        IF ((OLDX<=6) AND (OLDX>=2) AND
                          (OLDY>=20) AND (OLDY<180))
THEN SCROLL;
                        IF ((OLDX>=20) AND (OLDX<580) AND
                          (OLDY>=20) AND (OLDY < 180))
THEN CURSORMOVE;
            END;
END; {inputscreen}
{-------------------------------------------------}
PROCEDURE GETHELP;
var F10:TEXT;
  F1:FILE OF CHAR;
  I:INTEGER;
BEGIN;
            WITH TEMPFILE DO BEGIN
                        FNAMEOUT:=FILENAME;
                        PBEGINOUT:=PBEGIN;
                        XCOUT:=XC;
                        YCOUT:=YC;
                        WPINDEXOUT:=WPINDEX;
                        OLDXOUT:=OLDX;
                        OLDYOUT:=OLDY;
            END;
            ASSIGN(F10,'F:WPINFO.TMP');
            REWRITE(F10);
            WITH TEMPFILE DO

            WRITE(F10,FNAMEOUT,PBEGINOUT,XCOUT,YCOU
T,
                        WPINDEXOUT,OLDXOUT,
                        OLDYOUT);
            CLOSE(F10);
            I:=1;
            ASSIGN(F1,'F:STORY.TMP');
            REWRITE(F1);
            WHILE I<= WPENDMARKER DO BEGIN;
                        WRITE(F1,WPAREA[I]);
```

```
                        I:=I+1;
            END;
            CLOSE(F1);
            GRAPHICSOFF;
            GINPUTOFF(DEVMOUSE);
            EXITPROG(0);
END;        {gethelp}
{-----------------------------------------------}
PROCEDURE WP_WRITE2;
BEGIN;
            INPUTSCREEN;
            IF ICODE='E' THEN GETHELP;
            IF TESTCHAR<>0 THEN BEGIN;
                        IF WPINDEX<WPENDMARKER
THEN INSERTMODE
                          ELSE NORMALINPUT;
            END;
END; {WP_WRITE2}
{-----------------------------------------------}
PROCEDURE SAVETEXT(VAR OPTION:INTEGER);
VAR R1:RECT;
  TINDEX:INTEGER;
  F:FILE OF CHAR;
BEGIN;
            SETRECT(R1,50,70,200,150);
            SETBRUSHCOLOUR(0);
            FILLRECT(R1);
            SETBRUSHCOLOUR(1);
            SETPENCOLOUR(2);
            MOVETO(50,70);
            LINETO(50,150);
            LINETO(200,150);
            LINETO(200,70);
            LINETO(50,70);
            SETBRUSHCOLOUR(2);
            SETRECT(R1,60,120,124,140);
            FILLRECT(R1);
            SETRECT(R1,60,100,124,110);
```

```
            FILLRECT(R1);
            SETRECT(R1,60,80,124,90);
            FILLRECT(R1);
            SETBRUSHCOLOUR(1);
            SETTEXTCOLOUR(1);
            PLOTTEXT(60,130,' EXIT &');
            PLOTTEXT(60,120,'  SAVE');
            PLOTTEXT(60,100,'  EXIT');
            PLOTTEXT(60,80,' CANCEL');
            SETTEXTCOLOUR(3);
            PLOTTEXT(140,110,'SELECT');
            PLOTTEXT(140,100,'OPTION');
            OPTION:=0;
            REPEAT
                              READSCREEN;
                              IF ((OLDX>60)AND(OLDX<124)) THEN
                               IF ((OLDY>120)AND(OLDY<140))
THEN OPTION:=1
                               ELSE IF
((OLDY>100)AND(OLDY<110)) THEN OPTION:=2
                                ELSE IF ((OLDY>80)AND(OLDY<90))
THEN OPTION:=3;
            UNTIL OPTION<>0;
            IF OPTION=1 THEN BEGIN;

            DRIVEFILENAME:=CONCAT('A:',FILENAME);
                              ASSIGN(F,DRIVEFILENAME);
                              REWRITE(F);
                              TINDEX:=1;
                              REPEAT
                               WRITE(F,WPAREA[TINDEX]);
                               TINDEX:=TINDEX+1;
                              UNTIL TINDEX>WPENDMARKER;
                              CLOSE(F);
            END ELSE IF OPTION=3 THEN SCROLLSCREEN1;
            IF FSTAT('F:WPINFO.TMP') THEN  BEGIN;
                              ASSIGN(F,'F:WPINFO.TMP');
      ERASE(F);
```

```
            END;
END; {savetext}
{-----------------------------------------------}
PROCEDURE RESETVARIABLES;
VAR F10:TEXT;
  F1:FILE OF CHAR;
  I:INTEGER;
  CH:CHAR;
BEGIN;
            SETTEXTCOLOUR(3);
            SETBRUSHCOLOUR(1);
            SETBRUSHSTYLE(SOLID);
            SETBRUSHMODE(GREPLACE);
            SETPENCOLOUR(2);
            SETPENSTYLE(SOLID);
            SETPENMODE(GREPLACE);
            SETCHARUP(0);
            SETFONT(PRIMARYFONT);
            SETCHARHEIGHT(1);
            SETCHARWIDTH(1);
            SETCHARHEIGHT(1);
            SETCHARWIDTH(1);
            SETPENSTYLE(SOLID);
            ASSIGN(F10,'F:WPINFO.TMP');
            RESET(F10);
            WITH TEMPFILE DO

            READ(F10,FNAMEOUT,PBEGINOUT,XCOUT,YCOUT,

            WPINDEXOUT,OLDXOUT,OLDYOUT);
            CLOSE(F10);
            WITH TEMPFILE DO BEGIN
                        FILENAME:=FNAMEOUT;
                        PBEGIN:=PBEGINOUT;
                        XC:=XCOUT;
                        YC:=YCOUT;
                        WPINDEX:=WPINDEXOUT;
                        OLDX:=OLDXOUT;
```

```
                        OLDY:=OLDYOUT;
            END;
            ASSIGN(F1,'F:STORY.TMP');
            RESET(F1);
            I:=1;
            WHILE NOT(EOF(F1)) DO BEGIN;
                        READ(F1,WPAREA[I]);
                        I:=I+1;
            END;
            WPENDMARKER:=I-1;
            CLOSE(F1);
            HELPSET:=FALSE;
END;  {resetvariables}
{-------------------------------------------------}
PROCEDURE DISPLAY_HELP;
VAR F:FILE OF CHAR;
  R1:RECT;
   INCHAR:CHAR;
  XT:INTEGER;
BEGIN;
            ASSIGN(F,'F:HELPFILE.TMP');
            RESET(F);
            SETBRUSHCOLOUR(0);
            SETRECT(R1,50,50,400,150);
            FILLRECT(R1);
            MOVETO(50,50);
            LINETO(50,150);
            LINETO(400,150);
            LINETO(400,50);
            LINETO(50,50);
             XT:=60;
            WHILE NOT(EOF(F)) DO BEGIN;
                        READ(F,INCHAR);
                        PLOTTEXT(XT,100,INCHAR);
                        XT:=XT+8;
            END;
            CLOSE(F);
            TESTCHAR:=0;
```

```
            REPEAT TESTCHAR:=TESTCHR UNTIL
TESTCHAR=255;
            CHARIN:=GETCHAR;
            SETBRUSHCOLOUR(1);
            SCROLLSCREEN;
END;   {display_help}
{-----------------------------------------------}
PROCEDURE WP_SCREEN;
VAR F:FILE OF CHAR;
   OPTION,COUNT:INTEGER;
BEGIN;
IF NOT HELPSET THEN BEGIN;
            DRIVEFILENAME:=CONCAT('A:',FILENAME);
            IF FSTAT(DRIVEFILENAME) THEN BEGIN;
                              ASSIGN(F,DRIVEFILENAME);
                              RESET(F);
                              WPINDEX:=0;
                              WHILE NOT EOF(F) DO BEGIN;
                                WPINDEX:=WPINDEX+1;
                                READ(F,WPAREA[WPINDEX]);
                                END;
                              WPENDMARKER:=WPINDEX;
WPINDEX:=1;
                              COUNT:=0; XC:=20; YC:=170;
                              WHILE ((COUNT<17) AND
(WPINDEX<WPENDMARKER))
                                                DO BEGIN;
                                IF WPAREA[WPINDEX]<> CHR(13)
THEN BEGIN;

            PLOTTEXT(XC,YC,WPAREA[WPINDEX]);
                                    XC:=XC+8;
                                END ELSE BEGIN;
                                    XC:=20;
                                    COUNT:=COUNT+1;
                                    YC:=YC-10;
                                    WPINDEX:=WPINDEX+1;
                                END;
```

```
                                IF XC>580 THEN BEGIN;
                                        COUNT:=COUNT+1;
                                        XC:=20;
                                        YC:=YC-10;
                                 END;
                                 WPINDEX:=WPINDEX+1;
                                END
            END
            ELSE BEGIN;
                                WPENDMARKER:=1;
                                END;
            WPINDEX:=1;
            XC:=20; YC:=170;
            PBEGIN:=1;
END ELSE BEGIN;
            RESETVARIABLES;
            DISPLAY_HELP;
END;
            PLOTCUR;
            REPEAT
                                OPTION:=0;
                                REPEAT WP_WRITE2 UNTIL
ICODE='F';
                                IF WPENDMARKER>1 THEN
SAVETEXT(OPTION)
                                 ELSE OPTION:=2;
                                ICODE:='F';
            UNTIL ((OPTION=1)OR(OPTION=2));
END; {wp_screen}
{-----------------------------------------------}
PROCEDURE WORD_PROCESS;
VAR SELECTED,NOPRINT:BOOLEAN;
   XC,YC,LINEC,I,J,COL,ROW,ST1,COUNT:INTEGER;
BEGIN;
IF NOT HELPSET THEN BEGIN;
            WP_TOP_ICONS;
   PLOTTEXT(35,160,'You have the following stories :-');
            XC:=35; YC:=130; I:=1; LINEC:=0;
```

```
            SETTEXTCOLOUR(2);
            WHILE I<=DINDX DO BEGIN;
                                    COUNT:=0;
                                    WHILE
((DIRAREA[I]<>'.')AND(I<=DINDX)AND
                                            (COUNT<8))      DO BEGIN;
                                    COUNT:=COUNT+1;
                                    PLOTTEXT(XC,YC,DIRAREA[I]);
                                    I:=I+1;
                                    XC:=XC+8;
                                    END;
                                    IF ((COUNT<>8)AND(I<=DINDX))
THEN
                                            FOR J:=COUNT TO 7 DO
BEGIN
                                                    PLOTTEXT(XC,YC,' ');
                                                    XC:=XC+8;
                                                    I:=I+1;
                                            END;
              XC:=XC+40;
              LINEC:=LINEC+1;
              IF LINEC=5 THEN BEGIN;
                                    XC:=35;
                                    YC:=YC-10;
                                    LINEC:=0;
               END;
            END;
            SETTEXTCOLOUR(3);
            PLOTTEXT(35,50,'Select the story that you wish to
write.');
            SELECTED:=FALSE;
            WHILE (( ICODE<>'F') AND (SELECTED=FALSE)) DO
BEGIN;
                                    READSCREEN; COL:=0;
                                    IF
((ICODE='X')AND(OLDY<140)AND(OLDY>80)) THEN
                                      BEGIN;
                                    IF OLDY>130 THEN ROW:=1
```

```
                                ELSE IF OLDY>120 THEN ROW:=2
                                 ELSE IF OLDY>110 THEN ROW:=3
                                  ELSE IF OLDY>100 THEN ROW:=4
                                   ELSE IF OLDY>90 THEN ROW:=5
                                       ELSE ROW:=6;
                               IF (( OLDX>35) AND (OLDX<619))
THEN BEGIN;
                                       IF OLDX<99 THEN COL:=1
                                        ELSE IF OLDX>455 THEN
COL:=5
                                        ELSE IF
((OLDX>140)AND(OLDX<204)) THEN

COL:=2
                                          ELSE IF
((OLDX>245)AND(OLDX<309))
                                                    THEN   COL:=3
                                         ELSE IF
((OLDX>350)AND(OLDX<414))
                                                          THEN
COL:=4;
                              END;
                              IF COL<>0 THEN BEGIN;
                               CASE ROW OF
                                1:ST1:=0;
                                2:ST1:=5;
                                3:ST1:=10;
                                4:ST1:=15;
                                5:ST1:=20;
                                6:ST1:=25;
                               END;
                               ST1:=ST1+COL;
                               IF ST1<=(DINDX/8) THEN BEGIN;
                                     SELECTED:=TRUE;
                                     ST1:=((ST1-1)*8)+1;
                                     END;
                              END;
                             END;
```

```
                              END;
                              CLEAR_SCR;
                              IF ICODE='X' THEN BEGIN;
                               XC:=360; FILENAME:='';
NOPRINT:=FALSE;
                              FOR I:=ST1 TO (ST1+7) DO BEGIN;
                               IF DIRAREA[I]='.' THEN
NOPRINT:=TRUE;
                               IF NOT NOPRINT THEN BEGIN;

          PLOTTEXT(XC,236,DIRAREA[I]);
                                    IF DIRAREA[I]<>' ' THEN

          FILENAME:=CONCAT(FILENAME,DIRAREA[I]);
                                    XC:=XC+8;
                               END;
                              END;
                              WP_SCREEN;
                             END;
END ELSE WP_SCREEN;
END;  {word_process}
{-------------------------------------------------}
PROCEDURE RUNSTORYM;
VAR RETCODE:INTEGER;
BEGIN;
            IF NOT HELPSET THEN
      REPEAT READSCREEN UNTIL ICODE<>'X';
    IF ICODE = 'A' THEN BEGIN;
                               GETDIRANDNAME;
        REPEAT CREATESTORY UNTIL ICODE='F';
                              ICODE:='X';
                              TOPLEVEL;
                              IF ENDCREATE THEN CREATE2_5;
    END;
    IF ICODE = 'B' THEN BEGIN;
                IF NOT HELPSET THEN GETDIRANDNAME;
        REPEAT WORD_PROCESS UNTIL
                                                ICODE='F';
```

```
                                ICODE:='X';
          TOPLEVEL;
             END;
END; {RUNSTORYM}
{-------------------------------------------------}
PROCEDURE WELCOME;
BEGIN;
            SETFONT(SECONDARYFONT);
            SETCHARWIDTH(2);
            SETCHARHEIGHT(2);
            PLOTTEXT(240,160,'Welcome to');
            SETCHARWIDTH(7);
            SETCHARHEIGHT(6);
            PLOTTEXT(40,90,'MULTISTORY');
            SETTEXTCOLOUR(2);
            PLOTTEXT(42,88,'MULTISTORY');
            SETTEXTCOLOUR(3);
            SETCHARWIDTH(1);
            SETCHARHEIGHT(1);
            PLOTTEXT(75,50,'(c) 1986. M.R.Gardner.');
            SETFONT(PRIMARYFONT);
END; {welcome}
{-------------------------------------------------}
BEGIN   {main program}
  ENDCREATE:=FALSE;
   ICODE:='X';
  OLDX:=425; OLDY:=190;
   IF FSTAT('F:WPINFO.TMP') THEN BEGIN;
                                ICODE:='B';
                                HELPSET:=TRUE;
                                 SETUP;
  END ELSE BEGIN;
    WRITE(CHR(27),'[~F');
    WRITE(CHR(27),'[=2h');
            HELPSET:=FALSE;
   SETUP;
   TOPLEVEL;
   WELCOME;
```

```
  END;
  WHILE ICODE<>'F' DO RUNSTORYM;
  FINISHOFF;
END.
```

2.3. Batch file to startup MultiStory: GO.BAT

```
echo off
cls
echo Loading PROLOG .....
DEL F:*.TMP
PROLOG REDIRECT
echo on
```

2.4. Redirection file consulted on startup: REDIRECT

```
consult('\mick\runsugs.pro').
run.
```

2.5. Main Prolog control program: RUNSUGS.PRO

```
retractall(X):-retract(X),fail.
retractall(X):-retract((X:-Y)),fail.
retractall(_).
check_consult:-clause(storytype(_),_).
check_consult:-consult('f:cstoryf.pro'),
        storytype(St),storysit(Sit),cons_sug(St,Sit,File),
        conscl(File),setup.
cons_sug(1,Sit,File):-concat(File,['c:',rev,Sit,'.pro']).
cons_sug(2,Sit,File):-concat(File,['c:',lov,Sit,'.pro']).
run:-exec(storym),check_consult,
   do_sugs.
do_sugs:-
        trysug,do_sugs.
trysug :-   sug(X,Y),retractall((sug(X,_))),
```

```
            send_to_pas(Y).
trysug :-   send_to_pas(['No more suggestions.']).
send_to_pas(Mes):-open('f:helpfile.tmp',w),
                  name56(N),writef('f:helpfile.tmp',N),
                  nlf('f:helpfile.tmp'),
                  records_out(Mes),
                  close('f:helpfile.tmp'),
                  exec('storym').
records_out([]).
records_out([H | T]):-writef('f:helpfile.tmp',H),
                  nlf('f:helpfile.tmp'),
                  records_out(T).
```

2.6. Temporary story parameter file: CSTORYF.PRO

```
storytype(1).
storysit(one).
storychar(5).
att1(0).
att2(2).
att3(0).
att4(4).
att5(5).
att6(0).
att7(0).
att8(1).
att9(0).
att10(0).
att11(0).
att12(0).
att13(2).
```

2.7. Suggestions file for 'Revenge'/situation 1 type stories: REVONE.PRO

```
setup:-storychar(5),asserta(name56('Jake')).
setup:-asserta(name56('Dylan')).
sug(1,['The story should be retrospective.',
    'i.e. The character is looking back',
    'on a past childhood.']).
sug(2,['Decide whether Jake has a mother',
     'and a father, or just a father.']):-name56('Jake').
sug(3,['Describe life at home.',
     'Was the character badly treated ?']):-att1(X),X>2.
sug(3,['Describe life at home.',
    'Was the character badly treated ?',
    'There is a low health rating.',
    'This could be due to a poor upbringing.']):-att1(X),X<3.
sug(4,['Describe in what ways Dylans',
     'step parents are mean to him.']):-name56('Dylan').
sug(5,['Jake is very disruptive at home.',
   'His father beats him to try and control him.',
    'Jake has a high self-confidence rating.']):-name56('Jake'),
                                att10(X),X>3.
sug(5,['Jake tends to be disruptive at home.']):-name56('Jake').
sug(6,['Decide whether the father is always working',
     'earning a living and running the house; or',
     'has your character had to do all the house',
     'chores from an early age (more likely if',
     'determination is high.']).
sug(7,['Decide whether Jake s father is a rocker.',
     'This will affect the sort of life that',
     'he led. e.g. "parties", motorbike rallies.']):-name56('Jake').
sug(8,['Describe the events leading up to leaving',
     'home. Was it well planned or on the spur ',
     'of the moment.']).
sug(9,['Dylan could run away to find out who his ',
     'real parents are. His step parents will not',
     'tell him.']):-name56('Dylan').
sug(10,['Describe the parents reaction to ',
```

```
        'your character leaving.']).
sug(11,['Describe your characters life in the',
        'big wide world.']).
sug(12,['Your character meets and makes many friends.',
        'High friendliness and attractiveness rating.']):-att5(X),X>2,
                                                    att2(Y),Y>2.
sug(13,['Your character finds it easy to get a job.',
        'High intelligence and skillfulness rating.']):-att7(X),X>2,
                                                    att11(Y),Y>2.
sug(14,['Your character struggles against set-backs.',
        'High determination rating.']):-att4(X),X>3.
sug(15,['Describe events leading up to your characters',
        'return home. Has he/she made good !']).
sug(16,['Dylan could either find out the truth about',
        'his parents or return home in desperation.']):-
                                                    name56('Dylan').
sug(17,['Jake rours into town on his motorbike,',
        'intent on paying his father back']):-name56('Jake').
sug(18,['Why is your characters father to be pitied ?',
        'Possibilities : father himself was beaten when a lad,',
        '                the father has repented his ways,',
        '                the father has many problems.',
         'Your characters  high kindness increases the',
        'possibility of forgiveness.']):-att9(X),X>2.
sug(18,['Why is your characters father to be pitied ?',
        'Possibilities : father himself was beaten when a lad,',
        '                the father has repented his ways,',
        '                the father has many problems.']).
sug(19,['Think of an ending. What does your character',
        'resolve to do ? Does Dylan find out who his ',
         'real parents are ?']):-name56('Dylan').
sug(19,['Think of an ending. What does your charactar',
        'resolve to do ?']).
```

2.8. Assembler program DIR1.ASM

Assembler program to examine the users disk directory and create a directory file 'STODIR3.SYS' in the ram disk (drive F) of the names of all story text files (files ending with '.sty'). This routine is called from the main Pascal MULTISTORY program.

```
stack_seg       segment stack
          db 256 dup(?)
stack_seg       ends
data_seg        segment
           FILESPEC DB 'A:*.STY',0
BUFFER  STRUC
SUBS         DB 15H DUP(?)
ATF          DB 1 DUP(?)
TSF          DB 2 DUP(?)
DSF          DB 2 DUP(?)
LWF          DB 2 DUP(?)
HWF          DB 2 DUP(?)
OUTFILE DB 8H DUP(?)
BUFFER  ENDS
           createf db 'f:stodir3.sys',0
data_seg        ends
code_seg        segment
             assume cs:code_seg,ds:data_seg
program proc far
          CLC
           mov ax,data_seg
           MOV DS,AX
           MOV DX,OFFSET CREATEF
           MOV CX,00H
           MOV AH,3CH
          INT 21H
           MOV BX,AX
           mov dx,offset buffer
           mov ah,1ah
          int 21h
           mov dx,offset filespec
```

```
            MOV CX,01H
            MOV AH,4EH
            INT 21H
            JC L1
L2:         PUSH DX
            PUSH CX
            MOV CX,8H
            MOV DX,OFFSET OUTFILE
            MOV AH,40H
            INT 21H
            MOV AH,4FH
            POP CX
            POP DX
            INT 21H
            JNC L2
L1:    MOV AH,3EH
       INT 21H
       MOV AH,4CH
       mov al,0
       int 21h
program  endp
code_seg       ends
end     program
```

2.9. Assembler external functions GETCHAR and TESTCHR: GETCHAR.ASM.

These external assembler functions are used to poll the keyboard for determining when a key has been pressed (TESTCHR) and reading a key from the keyboard buffer(GETCHAR).

```
NAME    CHROUT
DGROUP  GROUP @C
     PUBLIC GETCHAR
     PUBLIC TESTCHR
```

```
@C      SEGMENT WORD PUBLIC 'DATA'
GETCHAR DD _GETCHAR
TESTCHR DD _TESTCHR
@C      ENDS
?CHROUT SEGMENT PUBLIC 'CODE'
        ASSUME CS:?CHROUT,DS:DGROUP
GC_STK  STRUC
        DW      ?       ;bp save
        DW      ?       ;ds save
        DD      ?       ;cs,ip save
GC_STK  ENDS
_GETCHAR PROC FAR
        PUSH DS
        PUSH BP
        PUSH ES
        MOV AH,7H
        INT 21H
        POP ES
        POP BP
        POP DS
        RET
_GETCHAR ENDP
_TESTCHR PROC FAR
        PUSH DS
        PUSH BP
        PUSH ES
        MOV AH,0BH
        INT 21H
        POP ES
        POP BP
        POP DS
        RET
_TESTCHR ENDP
?CHROUT ENDS
        END
```

2.10. Story Situations used by MULTISTORY

The following is a list of the situations used by MULTISTORY. Only story types 'Love' and 'Revenge' were properly implemented. Although data existed for story types 'Animals', 'Fantasy/Horror', 'Growing Up', and 'Conflict'.

The numbers following each situation are used by MULTISTORY to index and select the characters which are suitable for each particular situation (see the listing of characters in this appendix).

Situations for stories of type LOVE

SITUATION1: LOVE
A girl falls for a big macho (hard) man, a big rocker with sparkling studs. Her parents not only disapprove of him, but of her as she changes her clothes and her life style. She sticks by him through thick and thin, court appearances and a short spell in gaol for assault. (It was the other man's fault.) He does not look as if he is going to change.
<9,6,5>

SITUATION2: LOVE
Girl and boy in love. Girl still at school where she has a good chance of getting 'A' levels and a place in University. Boy is unemployed and unqualified. They are moneyless but happy. Then the boy is offered a good job by Canadian Uncle.
<8,7>

SITUATION3: LOVE
He is fantastic! He has everything. He prefers somebody else. A chance comes for your girl to leave the area. She would be much happier if she left, but she wants one last try.
<2,6,8,9>

SITUATION4: LOVE
Sweethearts since childhood, they are engaged to be married. But a story is going about one Beloved that makes the other wonder if marriage would be such a good idea.
<2,5,9>

SITUATION5: LOVE
(Historical) Daughter of Duke loves young stable lad (lots of opportunity to meet when she goes out riding). But naturally keeps it secret. Duke intends to marry her to rich Nobleman in the new year. A servant discovers the love affair and threatens to tell the Duke.
<29>

SITUATION6: LOVE
An elf princess is captured by Irma 'Skon Bekaa - cruel Gnorlanlord. She is imprisoned in the Tower of Rath Kimolyne. A young elf lord is later imprisoned in the cell next to hers. They plan escape and fall in love at the same time.
<21,10>

SITUATION7: LOVE
Life in the big city is much more difficult than he/she could have believed when at school a year ago. So she/he seems like the answer to those lonely prayers. Unfortunately Beloved has an alcohol problem.
<7,9>

SITUATION8: LOVE
She/he visits in winter the seaside town where she/he found a holiday romance two years ago. The past is not quite dead.
<2,5,6,7,9,23,24,28>

Situations for stories of type REVENGE

SITUATION1: REVENGE
A child, often beaten by a rather unloving dad, leaves home as soon as he can. Sometime later he returns to his home town intending to pay his father back for his years of suffering. But he finds the father more to be pitied than blamed.
<28,5>

SITUATION2: REVENGE
A student bullied during his first years at school, turns into a large and strong adolescent. What now ?
<2,5,7,23>

SITUATION3: REVENGE
Someone at work meets the person who bullied him at school. The ex-bully is asking for badly needed help. Someone is in a position of power.
<3,4,28>

SITUATION4: REVENGE
Some students plan revenge on a disliked teacher. They visit the teachers' home and discover another side of the teachers' character which changes their view of things.
<1>

SITUATION5: REVENGE
A fight ends in victory for one: humiliation for the other. The loser runs away to plan a revenge which will seem to come from someone else. But complicated plans lead to complications.
<2,3,5,7,10,23>

SITUATION6: REVENGE
One group wreak (do) a terrible revenge on another. One of the first group feels sorrow and guilt for what they have done and tries to make it right. His mates do not understand.
<7,5>

SITUATION7: REVENGE
A bossy supervisor at work. The workers plan and carry out their revenge. But somehow it is not as satisfying as it should have been.
<3,4>

SITUATION8: REVENGE
Someone who takes revenge on a hated policeman seems to have the last laugh - until the policeman is needed in an urgent crisis.
<5,7,25>

2.11. Characters used by MULTISTORY

CHAR1
Mr Johnson, male aged mid 30's. Teacher at the local school. Dissatisfied with his work, but cannot find a new job. He looks after his elderly disabled mother.

CHAR2
Peter, male, teenager. Midlands Judo champion. Spends most of his time between training for Judo and down the local disco. Hopes to eventually represent Britain in the Olympic games.

CHAR3
Jane, female, aged 28. Works as an accountant in a large company. She is very career minded and will do almost anything to progress further up the ladder.

CHAR4
Mr Arkwright, male, late 50's. Supervisor in a large steel mill in south Wales. Has spent all his life in the same factory and is very proud of his position.

CHAR5
Jake, male, aged 22. Rocker, who is never seen without his black leather motorcycle jacket and Brylcream in his hair. He is very proud of his large oily motorcycle and every year goes to the Isle of Man TT races. Elvis Presley is his idol.

CHAR6
Amanda, female, aged 16. Very shy and unassuming and wears dull looking clothes. Her parents are very old and never let her go out or bring friends to the house.

CHAR7
Bill, male, aged 18. Left school at 16 and hasn't worked since. Every week he visits the job centre but as he has no qualifications is always unlucky. He is not optimistic about his prospects.

CHAR8
Sue, female, aged 15. She is very bright and is regularly top of the class in maths and physics which are her favourite subjects. This causes her to be dis-liked by several of her class-mates who are jealous of her success.

CHAR9
Maureen, female, aged 19. Works as a shop assistant in the local chemist. She is happy with her lot and is looking forward to eventually getting married.

CHAR10
Glorfindel, Elf, aged 54,762 earth years. Elf lord, last surviving son of Elwe' King of the Elves. He has yet to claim his kingdom and will have to prove his lineage. He is a master of both sword and magic.

CHAR21
Marianna le feu, Elf, aged 29,131 earth years. She can change shape, disappear and cause fire. Her weakness is lead, against which her image will not work and which reduces her to a mere mortal.

CHAR23
Ken, male, aged 19. A young farmer without much hope of owning a farm. He cannot decide whether to stay working for his uncle, to go for an agricultural degree at college or to give up farming and take up boxing seriously. He was school boy champion of the South West 3 years ago.

CHAR24
Josh (stage name Melvyn Zapp), male 22. Rock singer. His group and his life have run into problems. So he has decided to spend some time away from the rock scene to sort them and himself out. The group promise to work with him again at Christmas.

CHAR25
Jeremiah, male aged ? A tramp, a wanderer who has met many strange people in his travels and has a treasure chest of interesting stories. He is thin and wears old suits, has a great taste for cider and cheese and loves all animals.

CHAR28
Dylan, male, aged 17. An orphan brought up by mean foster parents. Like many in his position he wants to know who his real parents are and is prepared to spend effort locating them.

CHAR29
Lady Margaret, female, aged 18, related to the powerful and determined Duke of Flint. She is a born romantic, likes horses and riding. She loves her parents but is beginning to see their faults. She lives in Flint Keep, a gaunt and giant castle situated in the wild and remote borders.

2.12. Character Attributes file used by MULTISTORY

There are 5 descriptions per character attribute, which are: health, attractiveness, calmness, determination, friendliness, Humour, intelligence, imagination, kindness, self-confidence, skillfulness, truthfulness, and strength.

Needs special care and decrepit
Always has coughs and colds
Mainly a clean bill of health
Healthy and fit
Never ill,robust,lots of vitality
Miserable looking and disfigured
Average features and dull
Pleasant and smiling
Handsome face
Unusually appealling
Nervous, restless and frets a lot
Anxious, emotional and excitable
Even-tempered and steady
Cool, composed and placid
Happy-go-lucky, unruffled
Weak-willed and easily persuaded
Half-hearted and gives up easily
Firm decision maker
Resolute and persevering
Stubborn, ruthless and single-minded
Unsociable and hostile
Keeps to one or two friends
Likes company and is loyal
Party-minded and seeks people out
Very sociable and gregarious
Humourless, miserable and gloomy
Sober and staid, rarely smiles
Cheerful, content and genial
Merry and laughs a lot

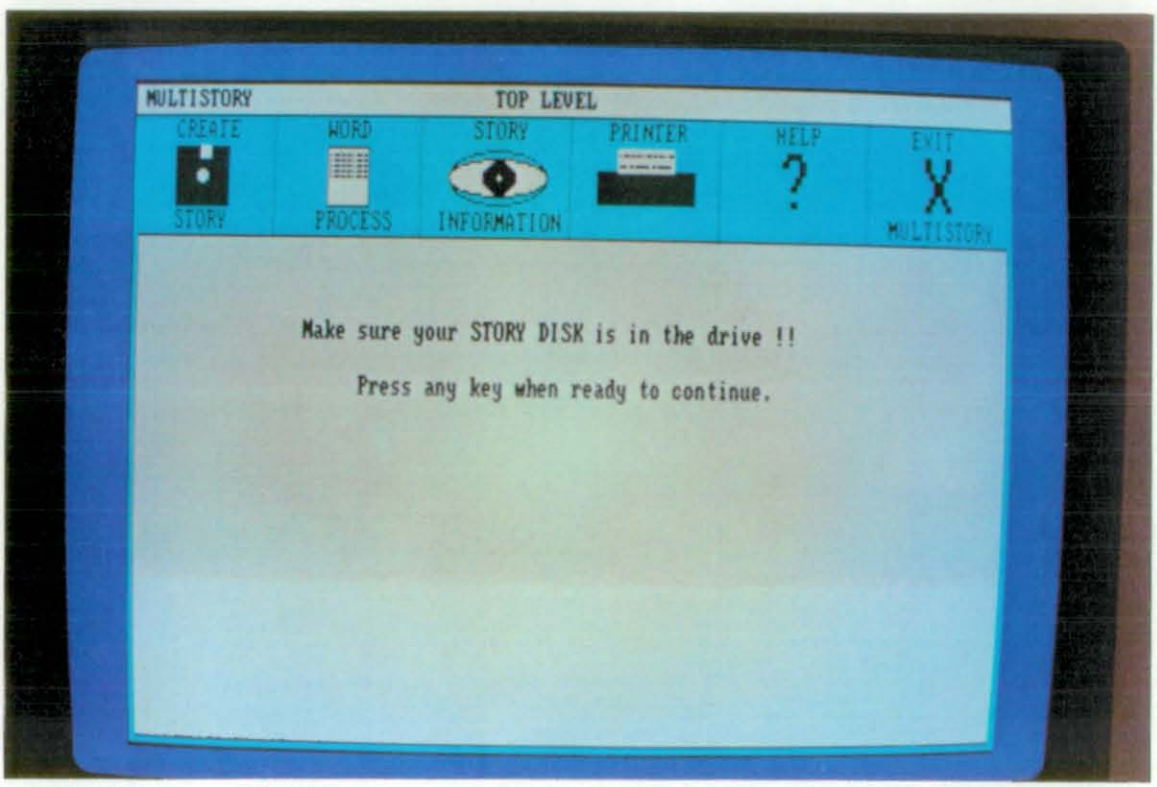
A joker who makes witty remarks
Slow, dull and irrational
A plodder who is unclear in thinking
Average ability with common sense
Sharp and quick to understand
Talented, perceptive and brilliant
Takes other peoples ideas
Some ideas but mainly guesswork
A day-dreamer who is quite creative
Empathetic and has good insight
Inspired, original and creative
Tough, inhumane and cruel
Ill natured and unaffectionate
Well meaning and helpful at times
Warm hearted and considerate
Benevolent and generous
Insecure,lets others do the talking
Hesitant, wavering and indecisive
Reliable, open-minded
Sure of self, authoritative
Opinionated, bigoted and cocksure
Clumsy and bungling
Untrained and slapdash
Out of practice but capable
Knows most answers to problems
An expert, versatile
Untrustworthy and corrupt
Unreliable with no principles
Fair minded and dependable
Law abiding and trustworthy
Honourable and incorruptible
Delicate, helpless and inactive
Puny, small and weak
Sturdy and able bodied
Muscular and well-built
Powerful, good stamina

APPENDIX 3.

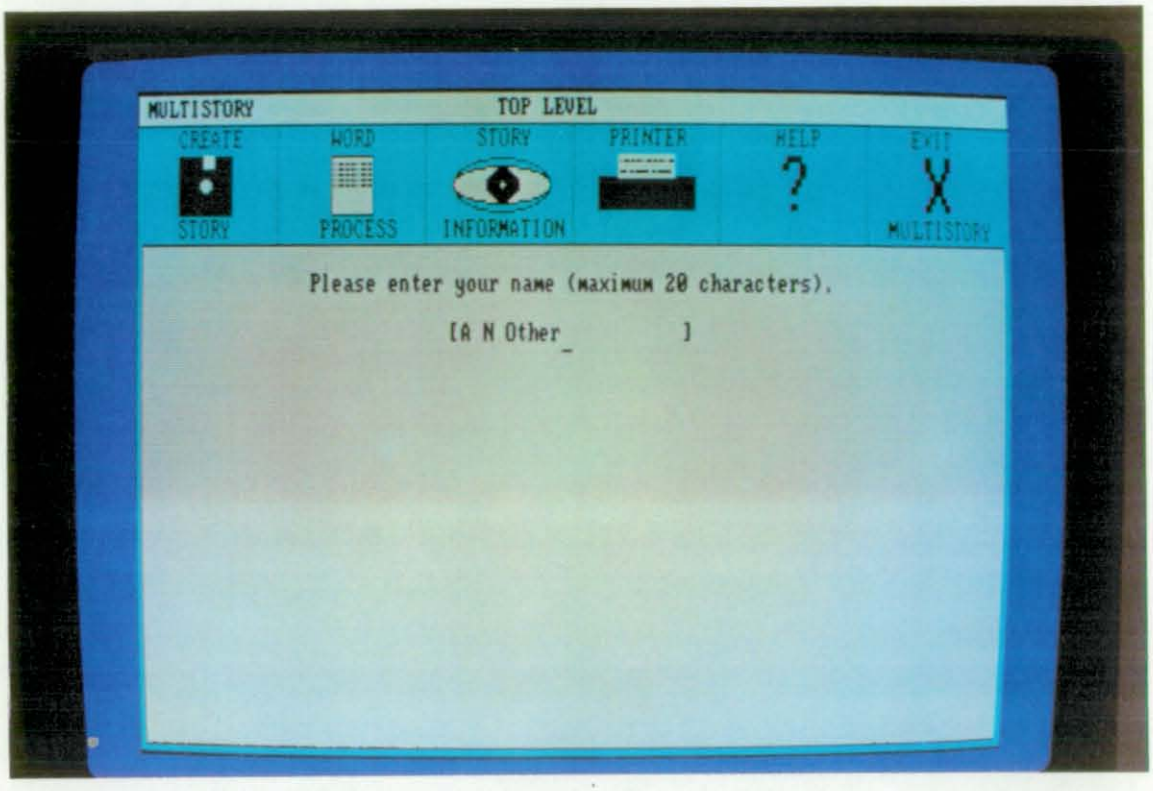
Sample MULTISTORY screens

MULTISTORY - Welcome screen.

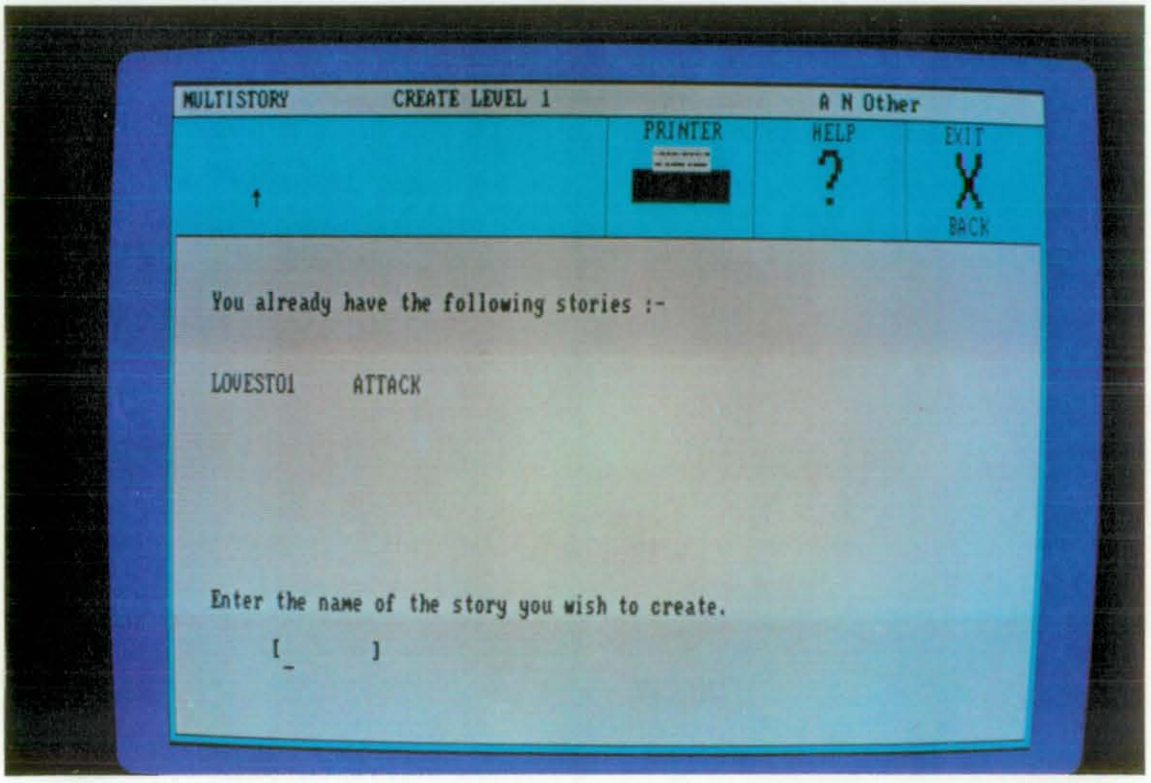
MULTISTORY - STORY DISK screen.



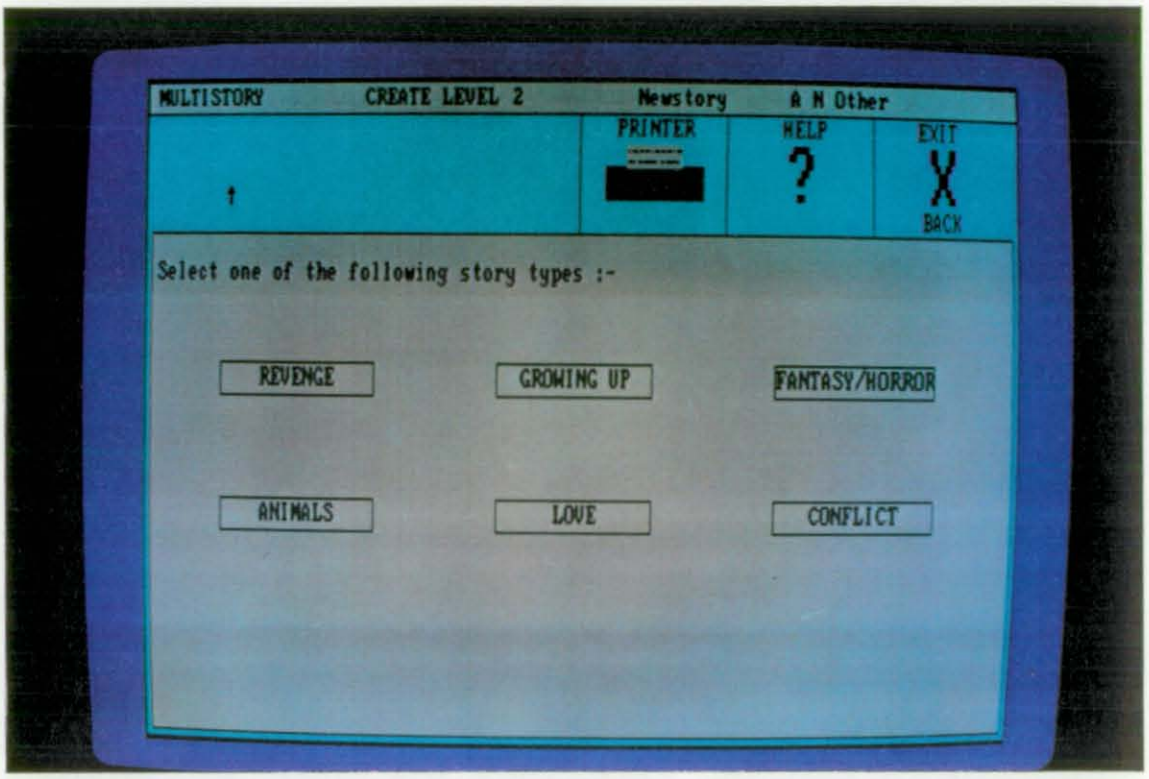
MULTISTORY - Enter user name (label is put on floppy disk).



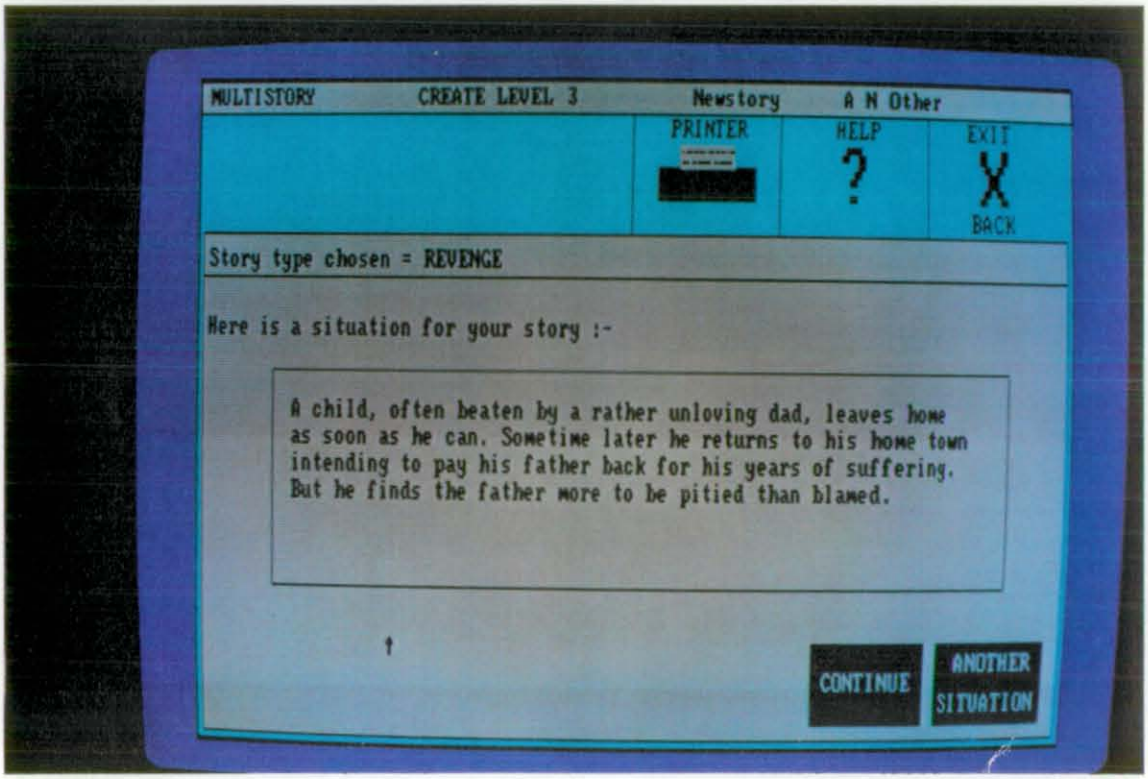
MULTISTORY - Create Story (level 1) - Enter new story name.



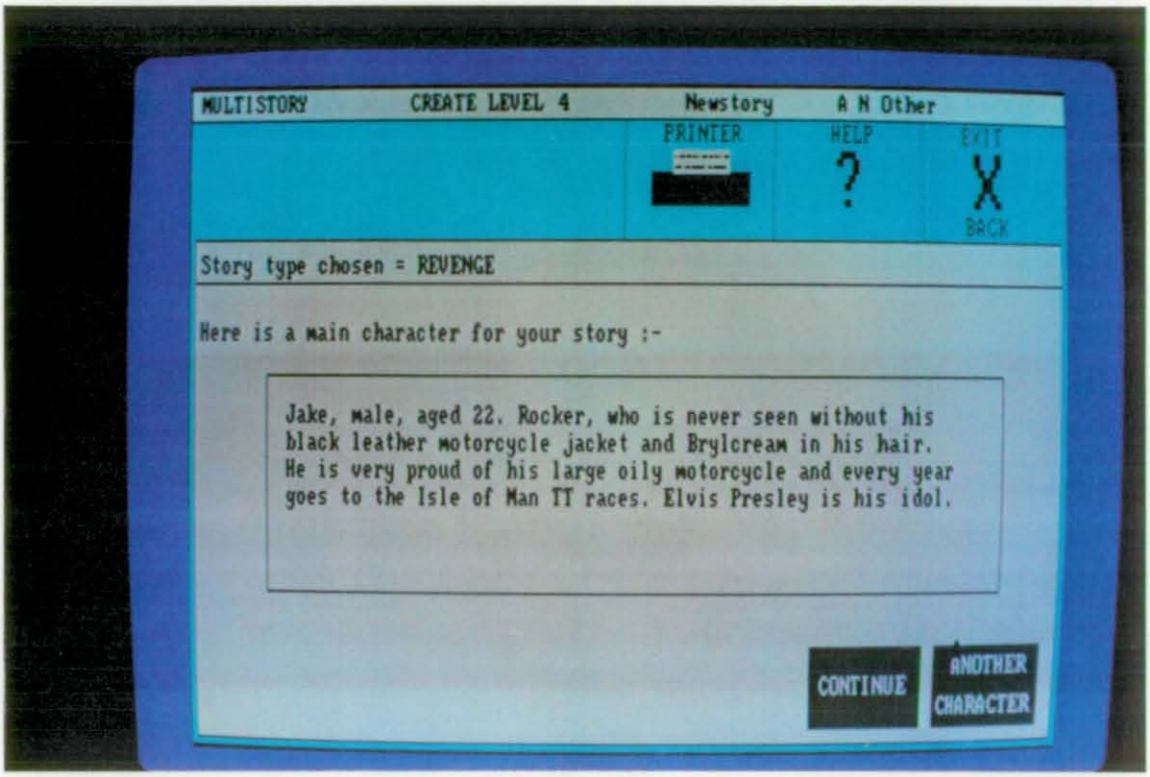
MULTISTORY - Create Story (level 2) - Select story type.



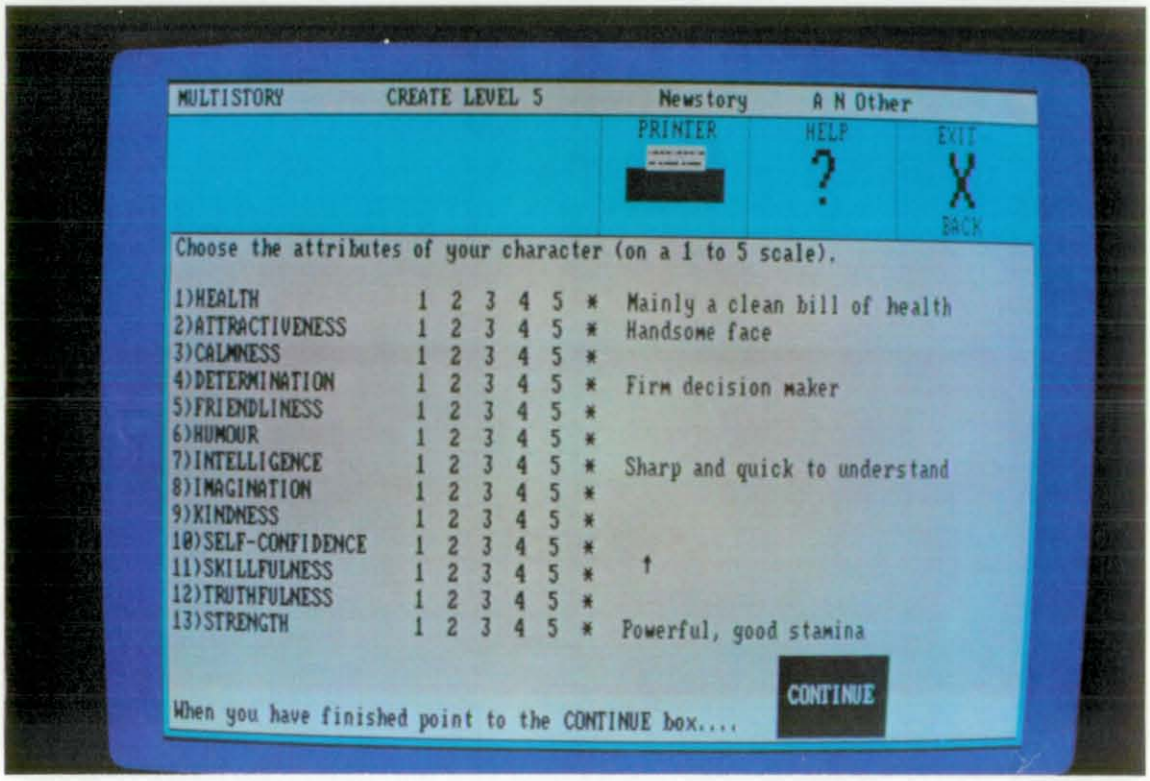
MULTISTORY - Create Story (level 3) - Select story situation.



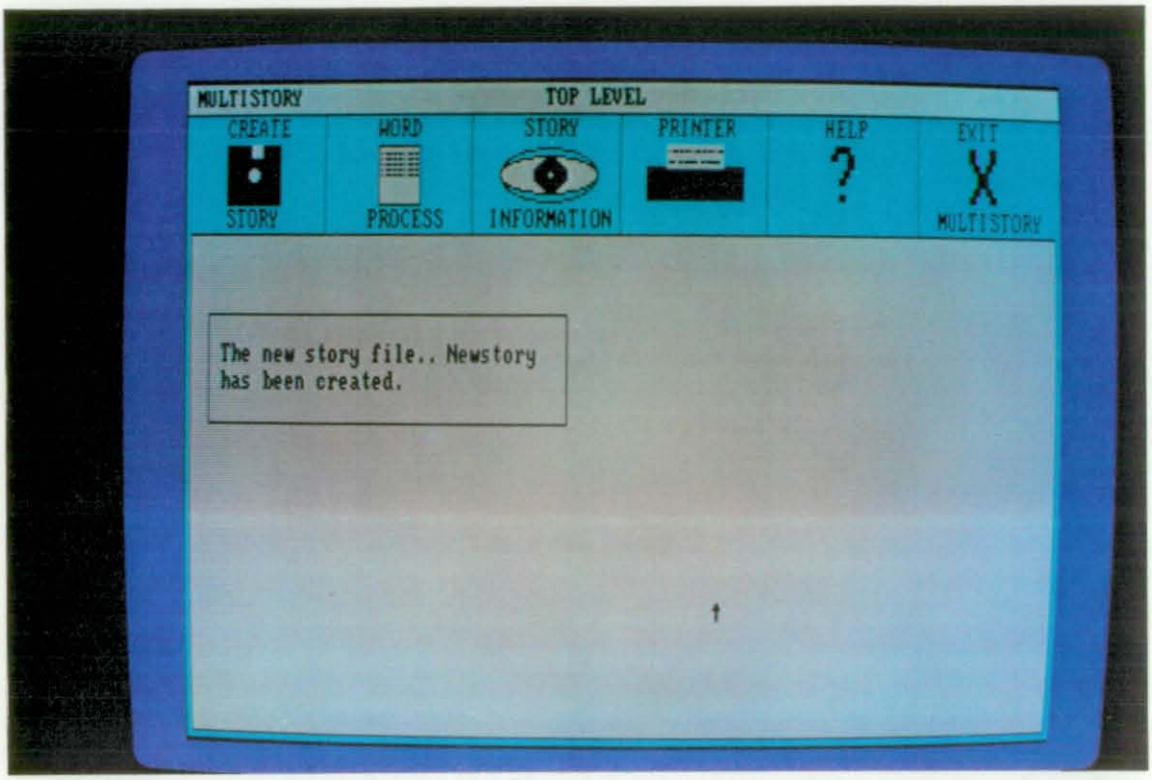
MULTISTORY - Create Story (level 4) - Select main character.



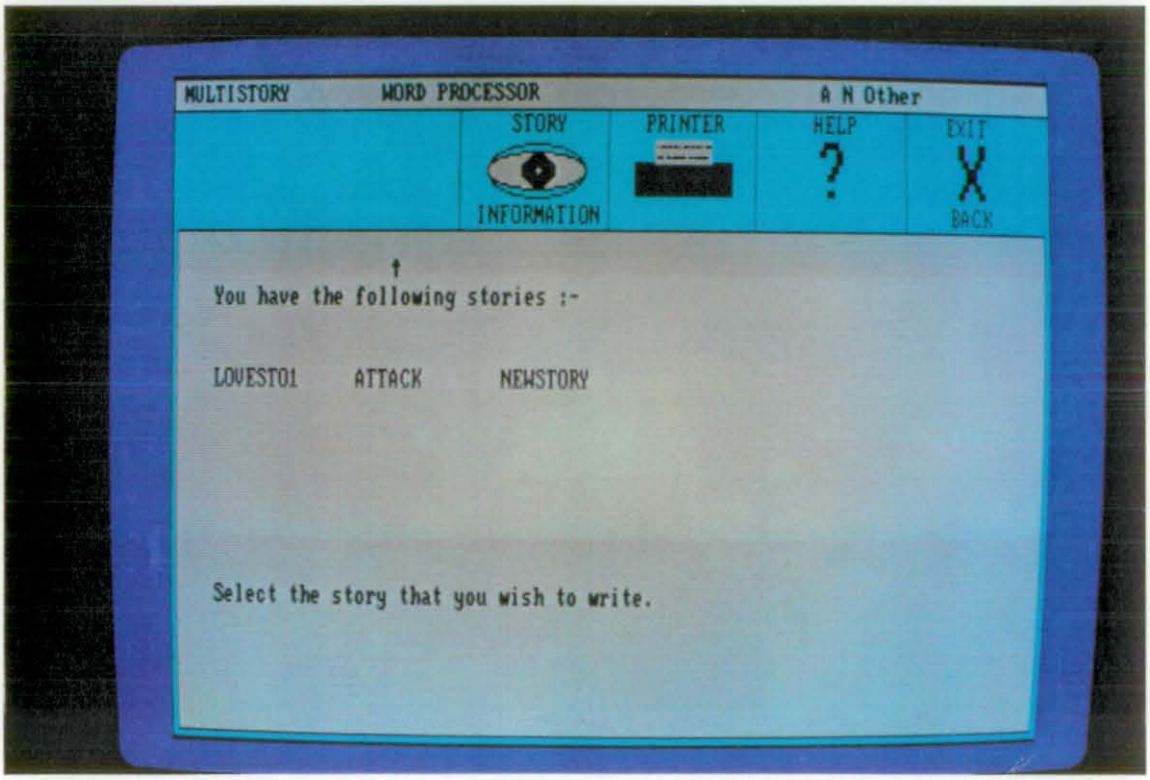
MULTISTORY - Create Story (level 5) - Select character attributes.



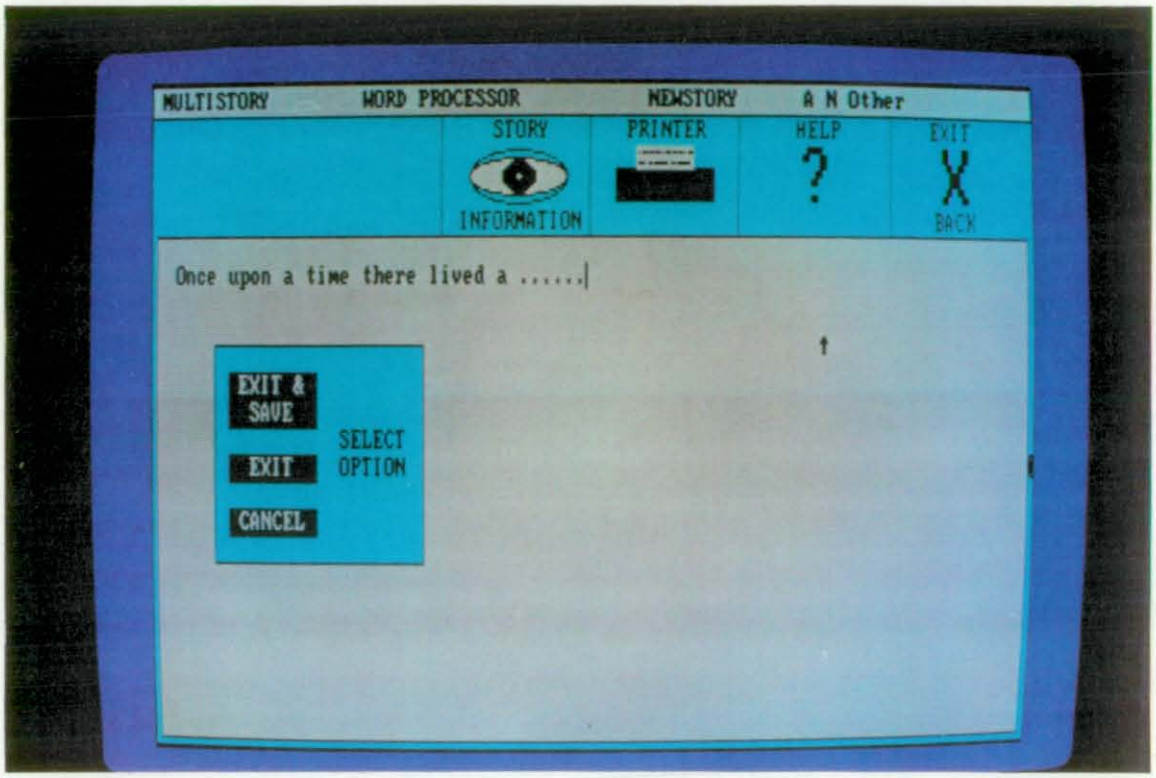
MULTISTORY - New story file created.



MULTISTORY - Word Processor - Select story.



MULTISTORY - Word Processor - enter story text & close editor window.



APPENDIX 4.

Research Machines Nimbus technical specification

TECHNICAL SPECIFICATION

ries	PC-186
ocessor	80186
ock Rate	8 MHz
eed	1 MIPS
ternal Bus	16-bit
usic Chip	8910
-Processor Option	8087
perating System	v3.1
icrosoft Windows*	✓
M Basic*	✓
C Basic*	✓
M Logo*	✓
M Mode Software/Utility	Options
andard Memory	512K or 1Mb
etwork Station Memory	1Mb
aximum Memory†	1.5Mb
raphics Memory	64K
MA Channel	1
pansion Slots	3
rial Port	✓
conet/Aux Serial Port	✓
und/Music Out (3 channels)	✓
ouse Port	✓
onochrome Video (multiple shades)	✓
inter Out	✓
ilt-in Loudspeaker	✓
C Parallel Printer/User Port	Option
ternal Serial Piconet Module	Option
ternal Parallel Piconet Module	Option
ta Communications ntroller (DCC)	Option
C Transfer Rate:	
Synchronous	Up to 60 Kbaud
Asynchronous	Up to 38 Kbaud
2-key Industry Standard Keyboard	✓
use (Microsoft Compatible)	Option
Concept Keyboard	Option
SketchPad	Option

Monitor (12" Hi-res Mono or 14" Med-res Colour)	Standard
High Speed LSI Graphics Processor	✓
640 x 250 x 4 Colour	✓
320 x 250 x 16 Colour	✓
CGA Graphics	With IBM Mode
1 x 256 IBM Compatible Char Set	✓
1 x 256 480Z Compatible Char Set	✓

Networking Option	RM Net 3
RM Net/Z-Net Server Cables	Up to 3
Stations Per Server	Up to 48
Diskless Station Options	✓
Cable Bandwidth	0.8 Mbits/Sec
SDLC CSMA/CD	✓
NRZI Encoded	✓
Processor Indopendent	✓
Multi-drop Bus	✓
Opto Isolation	✓
Max Cable Length	1200 metres
Coaxial Cable	50 ohm
BNC Connectors	✓

Diskless Network Station	Option
3.5" 720K Floppy Drives	Single or twin
Internal Hard Disk	20Mb
Average Hard Disk Access Time	23.7 ms
External Hard Disk	20 or 60 Mb
Cartridge Tape Streamer	20Mb
Transfer Rate	Up to 5Mb/minute
External Floppy Disk Drive Option	5.25" 600K

Width	356 mm
Depth	365 mm
Height	96 mm (case) + feet 4mm
Weight	7 kg max approx
Power Requirements	220/240V 50Hz
Heat Output	< 100 W

RM reserves the right to alter specifications without prior notice.

*A licence to run this software is provided as standard with all systems. In many cases the software itself is supplied with the system.
†The upper 512K of this 1.5Mb is for use as silicon disk or disk cache.