COMPUTER ASSESSMENT IN MATHEMATICS

By

David G. Wild B.Sc. (Hons) *Wales*

SUBMITTED FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

AT HERIOT-WATT UNIVERSITY

ON COMPLETION OF RESEARCH IN THE

DEPARTMENT OF MATHEMATICS

APRIL 1996.

I hereby declare that the work presented in this thesis was carried out by myself at Heriot-Watt University, Edinburgh, except where due acknowledgement is made, and has not been submitted for any other degree.

David Wild

David G. Wild B.Sc. (Hons) *Wales* (Candidate)

[illegible]

Prof. C.E. Beevers (Supervisor)

19 July 1996

Date

Contents

List of Figures

Acknowledgements

I dedicate this thesis to my father, Peter and my brother, Andrew for without them my love for learning and hard work would not have materialised. I owe them a great deal.

I thank my supervisor and friend, Professor Cliff Beevers, for his great wisdom and support during my work at Heriot-Watt. Without his knowledge and understanding this thesis could not have been written.

Also to my colleague Dr George McGuire for all his help and guidance.

I thank Dr Jan Abas who, as my undergraduate supervisor and great friend, geared me towards this field which I now enjoy so much.

Dr Hakim Moi helped me to gather my thoughts and ideas and aided me greatly towards the end of the thesis.

Alan Pithie was a great help with the DRIFT project and I thank him enormously.

To my girlfriend, who I love dearly, for her companionship in hard times.

Last but not least, I would like to thank my family and friends for their love and support.

Abstract

This thesis investigates methods of assessing students' mathematical ability by using the computer.

It starts by reviewing the general types of assessment within mathematics educational software and then describes some different ways of presenting the assessment on the computer by the use of varying types of questions.

In Chapter 2 there is a review of the literature and research conducted in the area of computer assessment of mathematics. In particular, the most prevalent dilemmas of computer aided learning and computer aided assessment are highlighted whilst looking forward at how the contents of further chapters in the thesis can help in addressing some of these difficulties.

The following chapter gives an historical account of how the CALM [1] software has addressed some of the inherent difficulties of assessment and highlights the ways in which some of these hurdles have been overcome. The shortfalls of CALM are described and, where relevant, pointers to the parts of thesis which tackle these shortfalls are given. In particular, the work in Chapter 4 undertakes an improvement in the way simple mathematical expressions [2] can be handled as it shows how binary tree constructions can be utilised within an educational environment.

Chapter 5 tests out two applications of the binary tree structures with the creation of a tool to aid student-computer communication of mathematics and by providing a method of comparing student-set questions against a true answer.

The following chapter describes an educational experiment which set out to show how a computer can be used to assess students' mathematical ability during a formal

[1]CALM is the acronym for the Computer Aided Learning in Mathematics project at the Department of Mathematics, Heriot- Watt University

[2]in this thesis, the word expression is taken to be a mathematical entity which does not contain any comparison operators

university examination. It deals with very important educational issues which arise when performing such examinations and gives conclusions as to their educational validity. In particular, issues of student input, partial credit, objectivity, consistency, flexibility and efficiency are considered along with the impact that this research could have for future testing of mathematics.

The final chapter describes how the thesis has been instrumental in further research and development within the field of computer assessment of mathematics.

Chapter 1

Introduction

This chapter gives an introduction to the field of computer assessment in mathematics. It starts by giving a description and a critique of the different types of assessment styles that are in common practice within computer aided learning software.

1.1 Assessment

Assessment is a process in which those being assessed are appraised or criticised on their actions. One definition of the assessment of a student [15] is

> **Assessing a student involves taking a sample of what he or she can do, and drawing inferences about a student's capability from this sample.**

Computer assessment can therefore be thought of as being a method of drawing conclusions on a student's capability where part or all of the assessment is being carried out by the computer.

From this generalisation of assessment stems the different types of assessment which exist within different computer coursewares. These types of assessment will now be defined.

The power of the computer can be utilised to provide many different styles of computer assessment. These different styles are described below:

1.1.1 Monitoring Assessment

Monitoring assessment, as the name suggests is a form of assessment used to monitor student progress throughout a period of study. Monitoring is often used in continuous assessment where taking marks and building progress profiles is in common practice. CALM 1 software (see Chapter 3) employs the monitoring process. This enables the facility to identify weaker students as a course progresses while encouraging the better students by acknowledging higher performance.

A great deal of work on monitoring assessment has been done at Glasgow Caledonian University under the CALMAT programme and at Brunel and the Open Universities.

1.1.2 Diagnostic Assessment

The word diagnostic within assessment refers to a process which undertakes to test, or diagnose, what students' strengths or weakness are within a particular subject area. Typically, computer packages are geared towards providing diagnostic information at the start of a student's studies. Some teachers and educationalists believe that computer diagnostic testing is the key to acquiring the initial knowledge of students' ability so that weaker students can be targeted more quickly. This and other reasons why diagnostic testing is useful is described in [14]. This article points out the aid to syllabus design and a possible method of streaming students that diagnostic testing gives. It also highlights that computer diagnostic testing is often easier to implement for large classes - particularly service mathematics classes.

Computer diagnostic testing methods cannot just help identify individual students. If properly implemented, these tests can provide important information on the general mathematical ability of a particular year's intake. This information can then be used in an assessment of a University advertising policy or can point out a weak admissions procedure.

Diagnostic testing is not only important to teachers. The more enthusiastic and conscientious student can use a diagnostic test result to identify his or her own weaknesses and hence iron out problems early on.

It may be argued, naturally, that it is not only a computer that can provide a means of diagnosing students. There are, however, good reasons why a computer can provide a better means of delivering diagnostic assessment to students. Some of these reasons are perhaps obvious but it is nevertheless important to point them out.

Tests at Nottingham University (see [14]) showed that computer diagnostic testing :

- encourages students to be self-critical;
- allows students to gain immediate feedback on their current ability;
- gives students their own learning profile;
- allows students to direct their own additional studies

However, many of these can be achieved by a traditional pen and paper diagnostic test and it is therefore only the second item in this list which gives the computer a distinct advantage. Another advantages, as with many other forms of computer assessment, are that the computer is objective. That is, the computer does not look at who is sitting the test. Student's handwriting can often cause a problem in traditional testing methods but with computer tests this is not the case (although with Multiple Choice Questions (MCQs) this is not a concern). Computer diagnostic testing mechanisms can contain questions with built in randomness. Possible answers to MCQs can appear in a random placement on the screen and any parameters in a question can vary. This gives computer testing another advantage in that, because of this randomness, the test can be taken again - perhaps to gauge any improvement in any student..

Not all diagnostic systems are of a multiple choice format. DIAGNOSIS from the University of Newcastle Upon Tyne asks for structured response answers (in the form of mathematical expressions). Here, Diagnosis places a skill tag with each question in the diagnostic test. From these tags, the system is able to gain knowledge of whether a student is able to perform a particular task. This enables the system to home in on students' weaknesses more quickly since time is not wasted testing skills which the student has already been deemed to understand. Quickly, a list of skills which a student does and does not have can be drawn and then the student can be directed to particular resources on the system.

Heriot-Watt, Nottingham and Napier Universities are among many others who adopt some sort of diagnostic computer assessment.

1.1.3 Self Assessment

Self assessment techniques involve students judging for themselves where their weaknesses lie. It has been shown that self assessment is an important educational skill for students to acquire [18].

As will be seen in Chapter 3, the CALM II project at Heriot-Watt University has employed self-assessment sections in their CAL materials. Here, students are given the maximum amount of feedback on the status of their answers and some possible incorrect answers have been pre-programmed into the software, thus enabling higher quality feedback to incorrect answers.

The United Kingdom Mathematics Courseware Consortium (UKMCC) have, with Mathwise (the product name for the UKMCC), included self assessment throughout the varying modules of the courseware. As students work they are invited to answer self-assessment questions based on the arguments seen in the theory sections.

1.1.4 Grading Assessment

Grading assessment is where a student is given a mark which is used in part (or completely) to grade a student's performance.

Grading assessment on the computer has not yet been fully exploited (see Literature Review - Chapter 2). The computer grading assessment exercises in mathematics have concentrated on either multiple choice questioning or other simple techniques such as text matching. However, as Chapter 6 will show, there are many other inherent problems of using grading assessment on the computer in mathematics. Although formal grading examinations have yet to be employed universally, many continuous grading assessment projects have been put into place. Brunel University and the University of Wales, Bangor, for example, used the CALM 1 software as a continuous grading assessment package. Here the marks carried 5 percent towards the first year undergraduate mathematics mark.

1.2 Styles of Question

Although there are many types of assessment, as described above, there are also differing styles of questions. MCQs have already been considered and other types of questions are now described.

- 'Multiple Choice' Questions

  Multiple Choice Questions (MCQs) have been used as a method of assessing students for many years. Here, students are asked a question and are presented with a number (usually between 3 and 5) of possible answers. Students must decide which answer is correct. A simple marking scheme adopted by many assessors consists of awarding one mark per correct answer. The problem with this type of multiple choice testing is that students could quite easily guess answers and therefore gain marks which they perhaps did not deserve. One strategy for combating this problem was to introduce "negative marking". Here, the expected mark on an examination for which all the answers had been guessed would be fixed to 0%. Naturally, this depends on the number of possible answers given to the student. Perhaps a better method of marking is to include, as one possible answer, an abstention. That is, a student could choose to abstain from answering a question - thus gaining or losing a mark. This, however, introduces a "gambling" element into the assessment - "am I totally sure that this is the answer or is it safer just to abstain?".

  Joanna Bull [15] suggests that

  > **guessing is a useful skill and should be encouraged and measured.**

  although in mathematics this is less true than in other disciplines.

  Although there are some serious educational apprehensions concerning the use of multiple choice testing, MCQs do have advantages over other, perhaps more technologically complex, types of assessment. One advantage is simply that

MCQs are easy (or easier) to implement than other forms of assessment and are particularly easy to computerise. Once computerised, the possible answers to questions can be placed randomly on the screen - thus making the test appear different to, perhaps, a neighbouring student. In mathematics, random parameters can be easily included in the questions. This has the advantage of enabling students to sit a test more than once or enables students to sit closer together for a test (usually, computer terminals in laboratories are closer together than desks in an examination hall).

Another advantage that MCQs has over other methods of computer assessment of mathematics is that they require no complicated answer checking. As will be demonstrated in forthcoming chapters, marking answers in mathematics on a computer can prove to be no easy task.

The disadvantage of MCQs is that they do not give any flexibility for the mathematical answers. For example, the expressions $\frac{1}{\sqrt{2}}$ and $\frac{\sqrt{2}}{2}$ are equivalent so which one should be written as being the correct answer?

There are now many different forms of MCQ testing. Hidden multiple choice questions (HMCQs) provide students with possible answers. Here, the possible answers appear one at a time whilst students have to dismiss or accept them, without the opportunity to reconsider an answer later. Although this method of testing has shown to reduce some of the afore mentioned problems with multiple choice testing, it has not been popular with students.

- 'Click' Questions
  Here, students are invited to click (using a mouse or other pointing device) on an object on the screen. For example, given a plot of a curve, a student could be asked to 'click on a local maximum' or 'click on a vertical asymptote'.

- 'Click and Drag' Questions
  A student could be required to click on an object and drag it to a correct point. For example 'drag the cross to the y-intercept'.

- 'List Choosing' Questions
  These answers require students to choose one or more items from a list. These

type of questions are particularly useful when there is more than one correct answer to a question.

- 'Ordering' Questions

  Here, a student could be asked to put into order a list of objects.

- 'Structured Response' Questions

  These type of responses require students to type mathematics into the computer. Although there are other types of questions, structured response questioning usually brings a greater degree of freedom to what can be asked. Amongst some of the advantages are that the "guessing" element in answering is diluted and students can give answers more as they would in a conventional examination.

Chapter 2

Literature Review

This chapter is intended to review and illuminate the historical progress of computer assessment. The work reviewed and the problems raised will be paralleled with those in the field of computer assessment of mathematics.

2.1 The Lack of Dedicated Literature

Whilst writing this review, it became more evident that assessment is not generally treated as being separate from the learning process. It could be argued that more traditional methods of learning mathematics (such as school examinations) do separate the assessment from the learning but for Computer-Assisted techniques, the assessment and learning seems to be more integrated. Because of this, there is a void of literature which is dedicated to the discussion of computer assessment; rather the literature plays down the importance of "end of topic" assessment and concentrates on "within topic" self-assessment.

This view is shared by Pritchett and Zakrzewski [33] saying that

> **Where computer assisted assessment (CAA) has been used it has usually been small scale using in house developed software.**

They add that

> **they (computers) have not hitherto been much used in this way for formal assessment purposes in higher education.**

2.2 Some Early Work and the Importance of Feedback

One of the earliest practices of computer based assessment was that of Skinner [37] in the 1960's. Skinner's program took student input and ...

> **immediately informed whether he (the student) is right or wrong**

.

Skinner emphasised the importance of including feedback within computer assisted learning packages, reinforcing Crowder's [17] view that

> **the essential problem is that of controlling a communication process by the use of feedback.**

Kulhavy [21] considered that

> **supplying feedback after an error is probably far more important than providing confirmation.**

2.3 How can Answers be Marked?

This work of Kulhavy unveiled one of the great problems of computer based assessment in that

> **there is a restriction as to the type of answer that can be required or demanded.**

In this sense, the literature shows that, in some way, assessment is driving the learning process in that these restrictions reduce the ways in which questions can be put to students.

O'Shea [31] states that

The evaluation of student responses by computer presents no problem (as long as they are expected to match the given answer exactly).

This quote encapsulates, but fails to appreciate, some of the inherent problems of assessing mathematics students on the computer. Chapter 3 presents the work of the CALM project in 1985 to address this and the work in Chapter 5 provides an additional solution to in the form of a more sophisticated numerical comparison routine.

Crowder [14] advocated that all questions and answers had to be of a multiple-choice format but authoring languages [36] (1976), although still primitive at the time, allowed basic string comparisons (for example, spelling mistakes, such as a missed out letter, were allowed).

Palmer and Oldehoeft [32] developed some interesting work on computer assessment in mathematics. Their program enabled students to type in equations as answers. The program marked the answers right or wrong and gave appropriate feedback. The idea of "random parameters" emerged where parts of the questions would contain randomly generated numbers, therefore creating, in effect, an infinite number of questions.

2.4 Artificial Intelligence

Kimball's integration tutor [20] hightlighted work on student assessment. Here, Kimball set up a dialog between his computer tutor and the student where the computer would be prompted to answer questions on simple integration. O'Shea [31] illustrates some interesting examples of how Kimball's system works - see below:

`Our integral is` $\int x \times \log(x)dx$

`What shall we do to solve it? :` *sub*

`Can you think of a substitution? (yes, help or exit)`

yes

```
Let $u = \log(x)$
Ok, Ruth, we will let $u = \log(x)$
Thus $x = e^u$ and $dx = e^u du$
which we substituted into the integrand.
Our new integral is now $\int u \times e^{(2\times u)} du$.
Please type good, bad or giveup.
```
bad

```
Can you think of a substitution? (yes, help or exit)
```
help

```
I can't find a suitable substitution! Will exit ···
```

This example has given the impression that his program is *intelligent* but in fact O'Shea [31] emphasises that

> **The program does have access to a solution, but this is not used directly; neither does the program make use of a general symbolic integration procedure, since this would be computationally expensive.**

During the artificial intelligence boom in the 60's and 70's, researchers and programmers began to acknowledge that building automated intelligent tutors was near impossible. Instead, *expert systems* [16] (systems which were deemed expert at a specific, usually small, task) were introduced. In this sense, the work done by Kimball could be thought of as an expert system, rather than an intelligent tutor.

2.5 Assessment Drives the Learning Process

Historically, more effort has been made in creating programs to teach rather than to assess what has been taught. However, it has often been the case that students prefer the assessment strand of computer assisted learning programs. A good example of this was within the TICCIT[1] project which set out to show that computer aided teaching in mathematics[2] could be more cost effective than more traditional methods. The studies showed that

> **the practice (test/assessment section) appeared to be the cornerstone of the TICCIT system.**

Another project, called PLATO (Programmed Logic for Automatic Teaching Operation) started early in the 1970's. Kane and Sherwood describe a use of the PLATO system in [19] teaching a course on classical mechanics in the Department of Physics, University of Illinois. Here, PLATO was used for homework exercises to

> **provide on-line checking of student answers to problems which are similar to the more difficult problems found in typical physics textbooks.**

As part of the students' activity on the PLATO system, during a proof of a theorem,

> **students type algebraic expressions whose correctness is checked by simple techniques available in the TUTOR language [36].**

These lessons on classical mechanics also contained "check-up" questions interspersed throughout the lesson. These problems also contained random parameters. Sherwood and Kane allowed simple checking of syntax of expressions - such as unbalanced parentheses. The system also enabled student to input appropriate units. If there units were wrong then a feedback message was given advising the student that "these are not the units of acceleration (for example).

2.6 Difficulties of Computer Assessment

Kenneth Mann points to some of the problems of computer testing in mathematics in [23]. Mann says that

[1]TICCIT = Time-shared Interactive Computer Controlled Information Television - see [31]
[2]TICCIT also targeted the teaching of English grammar

there is value pedagogically and economically in the development of a ... computerised system of assessment.

He continues by highlighting some benefits of using the computer in assessment saying that computer assessment

would enable a user to draw a random test, of reasonable distinctness

and

utilise the time of the mathematics educator more effectively.

Mann advocates that computer assessment systems

would enable the educator to test as often as deemed appropriate.

However, Mann highlights some of the problems using computer assessment saying that

there is a limitation on the type of question that may be asked

and

multiple-choice is the type of question that has been used in mathematics tests for many years.

Mann highlights the limited work done on structured response questioning saying

More emphasis is now being given to `calculation` questions, in which the student enters his answer as a number.

2.7 Formal Assessment on the Computer

Neill, in [30] points out some important issues which arise in formally assessing students on the computer in mathematics. Firstly, the importance of having good reliable hardware with suitable access is highlighted which is still a problem in many educational establishments. The paper shows that, if a course has been taught using Computer Aided Learning techniques then it is a natural progression to examine on the computer too. The examination questions, described in this paper, are not marked by the computer. Students simply fill in intermediate steps (on a piece of paper) in answering a question which are then marked by a human marker. In fact, Neill reports that

> **laboratory based examinations need more preparation, organisation and** *marking time* **than traditional examinations.**

This view will be argued in chapter 6 where it is suggested that if computers are to be used instead of traditional paper examinations then the effort to create and mark by computer should, in fact, be more efficient than that for written papers.

Neil points out some interesting issues concerning the learning process which arose during the computer examination. Firstly, students reported that they were

> **less stressed**

by

> **working in an environment with which they are familiar**

rather than

> **sitting papers in a large hall with several hundred others (students).**

This view is backed up by the work described in chapter 6 (see [11]).

Lomax describes how the computer can help in generating questions which differ slightly for each student. His article [22] shows how MC Questioning with random parameters can enhance the testing (particularly in self-testing) process.

2.8 Communication with the Computer

O'Shea [31] states that a student

> **should not be distracted from the subject at hand by having to search ways to express himself.**

Here, O'Shea hits upon one of the most prevalent problems in the assessment of mathematics students by computer. That is, how can a computer program assess students effectively if the communication of answers between student and computer is inadequate?

It is clear from the literature available that educators have generally avoided the problems of communication by using Multiple-Choice Questions [33]. Communication, in itself, need not cause a problem but for the more complex styles of questions, such as structured response, good communication is vital.

McCabe and Greenhow [25], identify 5 different question types which they employ in their software. These are Multiple Choice, Multiple Response, Text Match, Numeric and Hot-spot. That is, at that time, they avoid structured response questions and the problems which they bring.

Chapter 3

CALM Project Review

This chapter is intended to give a general background to the CALM Project within which the work for this thesis was based. In particular it will concentrate on the assessment strand of the project.

The CALM Project (Computer Aided Learning in Mathematics) was established at Heriot-Watt University in 1985. It was funded as part of the Computers in Teaching Initiative where the money was spent on creating a computerised tutorial system. The first CALM materials were written on Nimbus Research Machines for first year science and engineering students. Later, the software was ported to the IBM compatible personal computers by J. H Renshaw at Southampton University.

The CALM software divided the learning process into three main sections; theory, worked examples and a test. After extensive evaluation, the students reported that the test section was the most popular - showing again that students of mathematics generally like "learning by doing". In fact, the test section of the CALM software turned out to be it is cornerstone and the work done on the test has proved to be the driving force for the future development of CALM.

3.1 The CALM Test Section

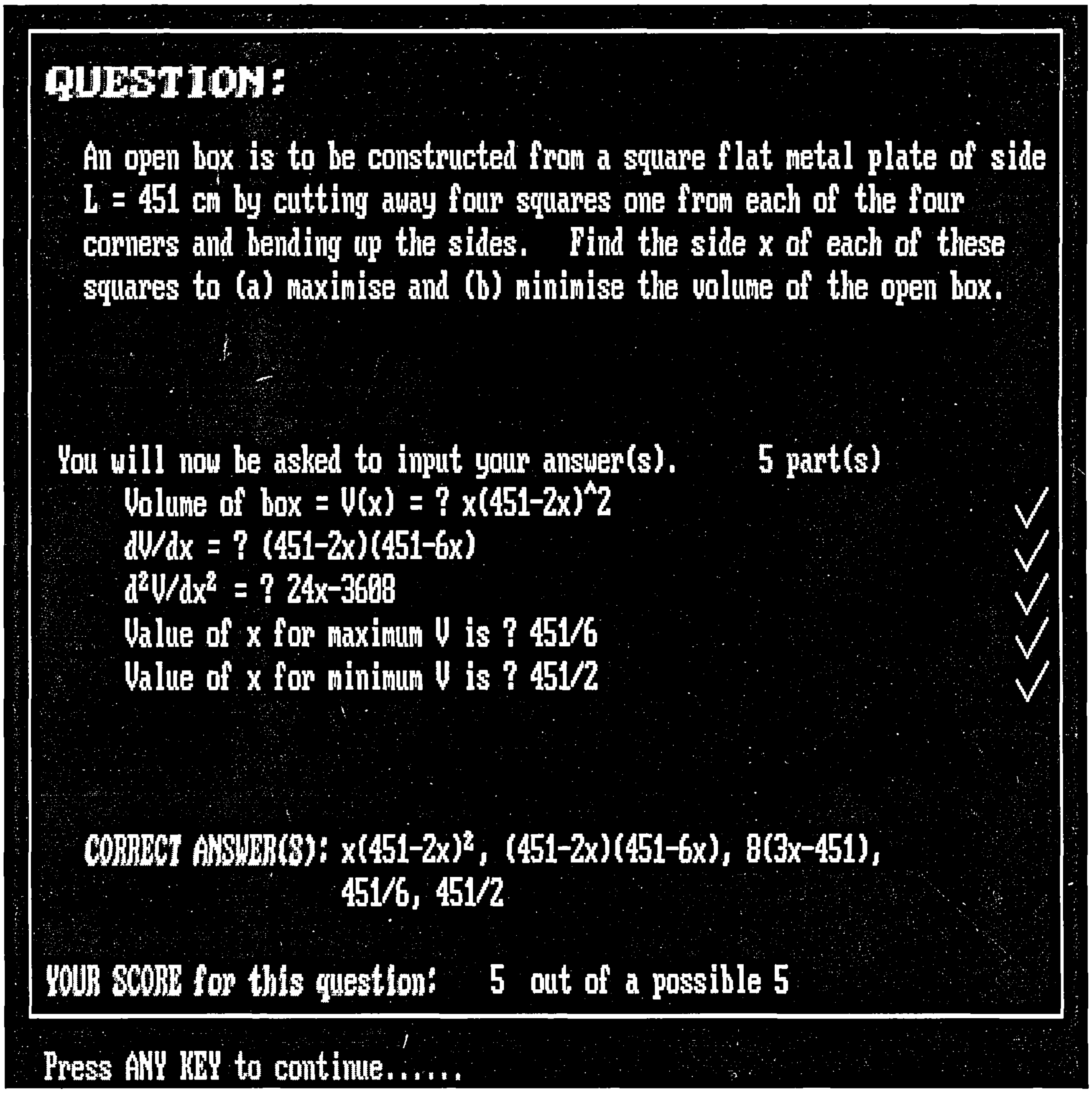
The significance of the CALM Test Section is shown in [7] by saying

> **In all CAL software one of the most important elements is some form of test which allows the students the chance to assess their**

understanding of the subject and to reinforce the theory being presented.

The test section set questions which appeared in a more "traditional style." Here, the main text of the questions were presented to students much as traditional tutorial questions. There is a difference in the overall question presentation due to the lack of computer intelligence. The original statement of the question is the same on the computer as it is on paper but the computer is unable to judge what is important when answering a question and therefore a human must break the question up into smaller parts and get the computer to ask it in particular stages.

Moreover, mathematics questions can usually be solved in varying ways and therefore the questions are broken down into steps which are deemed important in answering the complete question. The following diagram (figure 3.1) shows a typical breakdown of such a question.


QUESTION:

An open box is to be constructed from a square flat metal plate of side L = 451 cm by cutting away four squares one from each of the four corners and bending up the sides. Find the side x of each of these squares to (a) maximise and (b) minimise the volume of the open box.

```
You will now be asked to input your answer(s).      5 part(s)
    Volume of box = V(x) = ? x(451-2x)^2                        ✓
    dV/dx = ? (451-2x)(451-6x)                                  ✓
    d²V/dx² = ? 24x-3608                                        ✓
    Value of x for maximum V is ? 451/6                         ✓
    Value of x for minimum V is ? 451/2                         ✓
```

CORRECT ANSWER(S): $x(451-2x)^2$, $(451-2x)(451-6x)$, $8(3x-451)$, 451/6, 451/2

YOUR SCORE for this question: 5 out of a possible 5

Press ANY KEY to continue......

Figure 3.1: The CALM Test Section - Question Breakdown

3.1.1 Answering Questions in a CALM Test

Students answer questions and are marked according to the rules set out in 3.2. The feedback from the software was dependent on whether students decided to sit a "very easy", "easy" or "hard" test. These first two categories give ticks and crosses to answers and the "very easy" test gives answers to incorrectly answered parts after three incorrect attempts. The hard test is marked at the end of the question. Diagram 3.1 shows a question within an "easy" test.

The student performance can be reviewed via a program which reads output files from the CALM tests. From these, teachers can easily identify students with low marks or those students who are absent.

3.1.2 Mathematical Input

Since student answers are generally in the form of mathematical expressions, a method of expression input must exist within the software. CALM employed a simple one-line input where mathematical expressions must be typed in a FORTRAN-like manner.

Within this one-line, a few mathematical symbols could be input such as $\surd$, π, the squared symbol $()^2$ and θ. A power mode was also available which allowed the input of integer powers. The squares of trigonometric functions could be input in the usual manner ($\cos^2(x)$, for example). Fractions were input with the *slash* symbol - such as `1/(x-1)` for $\frac{1}{x-1}$.

[9] points out the difficulties students sometimes had using this notation and [7] writes

> **we need to create a better interface between students and computer to enable them to enter mathematical expressions more easily.**

It continues by saying that

> **... the student still does not have the freedom available with pen and paper to express mathematical ideas. The solution is**

not trivial but it will lead to a much more successful use of CAL in Mathematics

Further details can be seen in [6].

Chapter 4 deals with this very issue and puts forward a solution to this problem.

3.2 Marking Student Answers

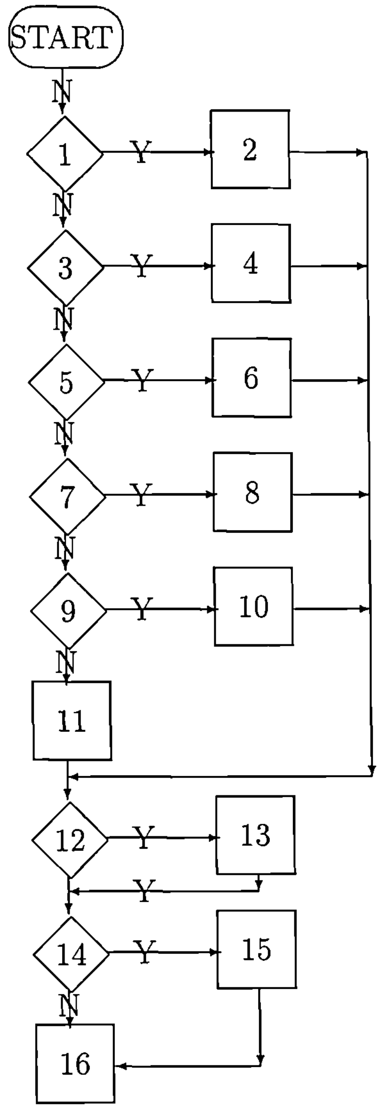
In order to mark student answers, the CALM software employs "expression evaluation." Here, student answers are compared against true answers by numerically comparing the student and true expressions at a number of suitable points over a particular range.

The expression, which is in one-line format can contain variables from a set a to z (but missing out e which is used as the mathematical constant $exp(1)$). Each variable in the expression can be either locked or unlocked. A locked variable indicates that it has already appeared in the expression and therefore it is value remains fixed (locked) during a particular evaluation. For example, during the evaluation of the expression $a\cos(a\theta)$, the value of a must not change from being one value outside the cos function to another inside.

The mathematical expressions contain various operators and identifiers, all of which have an associated precedence (priority). These precedences help to give the order in which parts of the expressions are evaluated. CALM eval makes use of recursive programming - whose process will now be described:

A mathematical expression is deemed to contain the following constructs: An expression, a simple expression, a term, a signed factor or a factor. Here, an expression is said to be the most complex whereas a factor is a smaller part of an expression. For example, the **expression** $x(y+z)$ has two **factors**, x and $(y+z)$. The second **factor** $(y+z)$ is too an **expression** which can be broken down into two **terms** y and z separated by the operator $+$. This evaluation procedure checks for implied multiplication. That is, expressions need not necessarily contain the $*$ operator to signify multiplication. The juxtaposition of two variables, a variable and an expression or two factors means that multiplication is implied. The ability to determine implied multiplication was built into the evaluation procedure.

Once the routine finds factors within the expression, it is necessary to evaluate the factor at the valid points of the variables which appear. This process is best illustrated by the diagram (figure 3.2) below. Note that the labels for the nodes on this diagram are described in table 3.1.



3.3 CALM Compare

Since it was required that students' answers be marked against true answers, CALM Compare was set up to check if students' answers were mathematically equivalent to a predefined true answer. The question of how two answers can be mathematically compared must therefore be asked.

◇, □ or ⊂⊃ Number	Associated Text
⊂⊃ 1	START the algorithm
◇ 1	Is the expression a constant?
□ 2	Convert the input string into its numeric equiv.
◇ 3	Is the expression surrounded by brackets?
□ 4	Recurse with bracketed expression and return value
◇ 5	Is the expression π?
□ 6	Return the value π
◇ 7	Is the expression surrounded by a $\surd$?
□ 8	Recurse, taking the $\surd$ of argument
◇ 9	Is the expression a function?
□ 10	Recurse and return the value operated on by the function
□ 11	It must be a variable - get value for it
◇ 12	Is the variable being squared?
□ 13	Square the value of the variable
◇ 14	Is multiplication implied?
□ 15	Insert a multiplication $*$ in expression
□ 16	Return the result

Table 3.1: Text for Nodes on figure 3.2

3.3.1 String Comparison

The naive reader may ask the question "why can't the symbols in each expression be checked against each other to see if they are the same - thus proving that the true and student answers were either equivalent or different". The following very simple examples (see table 3.2) will illustrate why this method of comparison is, in general, insufficient and inadequate.

TRUE Answer	Equivalent STUDENT Answer	Comment
$\frac{3}{2}$	$1 + \frac{1}{2}$	Any number of fractions are equivalent
$\frac{3}{2}$	$+ \frac{3}{2}$	Include a unary plus or spaces for readability
$\frac{3}{2}$	1.5	The decimal representation of $\frac{3}{2}$
$\cos(2x)$	$\cos(2 * x)$	Not using implied multiplication
$\cos(2x)$	$\cos(x)\cos(x) - \sin(x)\sin(x)$	Double angle formulae
$\cos(2x)$	$\cos^2(x) - \sin^2(x)$	Another version using double angle formulae
$\cos(2x)$	$1 - 2\sin^2(x)$	A trigonometric identity

Table 3.2: Examples of the Inadequacy of String Comparison

These simple examples show that there can sometimes be an uncountably infinite number of ways of writing some very simple expressions.

3.3.2 CALM Compare's String Evaluation

The CALM software uses the recursive evaluation routine described above along with a comparison routine which is now explained.

Another name of String Evaluation is Numerical Comparison. This, as well as the student and true answers, an interval $[a, b]$ (over which to compare) and the number of comparisons inside $[a, b]$ are needed. The algorithm also uses a specified tolerance t and a failure rate f.

The different types of expressions which need to be compared are now described, along with stated suitable values for the range, number of points, tolerance and failure rate.

3.3.2.1 Expressions Representing Constants

True and Student answers which do not contain either variables or identifiers (letters which are constants in the context of a question) can be compared relatively easily by computing the value of each expression.

Two constant expressions are said to be equal if they are either identical in value or agree within a certain tolerance t. For example, if the true answer is the mathematical constant π then, using a certain value of t, 3.14 is equivalent. This is to say that within CALM Compare, two answers are *equal* if they *almost* agree. In fact, for constant expressions, they are equal if the relative error is less than or equal to a given value of t.

The relative error E_{rel} between a true answer A_t and a student answer A_s is calculated as follows:

$$E_{rel} = \frac{|A_t - A_s|}{|A_t|}$$

The only difficulty with comparing constant functions in this way is when the true answer A_t is close to 0. Here, E_{rel} would get very large and give an inaccurate determination as to the correctness of the student answer. To remedy this, the absolute error, E_{abs}, is used when A_t is close to 0. E_{abs} is defined as follows:

$$E_{abs} = |A_t - A_s|$$

3.3.2.2 Expressions Representing Functions of One Variable

Mathematically, two expressions can be considered as being equal if their domains are equal and their values agree at all points of the domain. Obviously, it is impossible to compare true and student answers at all (possibly an infinite number) points on their domains. Therefore, CALM uses a subset of their domains, the interval $[a, b]$, in which to compare answers. Comparison is made c times from randomly chosen points in $[a, b]$. The way in which $[a, b]$ and t are chosen is crucial in the comparison of expressions and the choice depends on the expression for the true answer. For example, suppose a true answer is $\frac{1}{1+x^2}$. Suppose that $c = 11$ random points are chosen in $[a, b] = [-0.1, 0.1]$ with a tolerance of $t = 0.01$ (1% relative error), consider the following student answers:

- 0

  Here, from its definition, the relative error between student and true answers is $E_{rel} = 1$. Since this value is greater than t, the student answer is marked wrong.

- $\frac{1}{1+y^2}$

  This answer does not represent a function of x (the variable in the true answer). The student answer will therefore be marked as incorrect.

- 1

  Here $\forall x$, $E_{rel} = x^2$. Now, since $|x| \leq 0.1$, this gives a value for $E_{rel} \leq t \; \forall \; x \in [-0.1, 0.1]$ and will therefore be marked as being correct.

As can be seen from this example, it is not always trivial to choose values of a and b for the compare interval $[a, b]$. In fact, it is always possible to choose values for a and b so that an incorrect student answer is marked as being right! However, it should be pointed out that the student must know the correct answer for these values of a and b to be calculated.

One must also be wary that an interval is not chosen in which, for some values, the true answer is not defined. For example, if a true answer is $\frac{1}{x}$ then the range should not include the point $x = 0$. One problem, however, is that teachers can never be sure what a student will type as an answer. Even if the range is carefully chosen so that the true answer does not become undefined, a student answer may be written in such a form as to contravene this range.

For example, say a true answer is $\frac{1}{x+1}$ and the range was set to $[a, b] = [1, 2]$. A student could, quite conceivably, forget to factorise their final answer and input $\frac{x-1}{x^2-1}$. As a teacher, it could be argued that this answer is not strictly incorrect. However, if $x = 1$ is chosen as one of the comparison points then it would be marked as being wrong.

To try and avoid this, the concept of a failure rate f was introduced. This enabled answer equivalence to fail f times during the comparison. As long as c was chosen significantly high and the student's expression agreed with the true expression $c - f$ times then the answers can still be marked as being equal. In practice, f is chosen to be a percentage of c.

3.3.2.3 Expressions Representing Functions of Two or More Variables

The choice of the range becomes yet more complicated when comparing answers of two or more variables. Here, say for a function of two variables, CALM Compare checks to see if the true and student answers are equal on a set of four regions of the plane. The regions of the plane are defined by $[a, b]$ or $[-b, -a]$. Therefore, answers are compared over the four regions $[a, b] \times [a, b]$, $[a, b] \times [-b, -a]$, $[-b, -a] \times [a, b]$ and $[-b, -a] \times [-b, -a]$. The book [7] gives an interesting example as to why these regions are chosen. This example follows:

Take a true answer as $\ln(|(1-x)(1-y)|)$ which is defined $\forall x$ and $y \neq 1$. If the range $[a, b]$ is set to $[2, 10]$, consider the following student answers:

- $\ln((1-x)(1-y))$ This answer will be marked as correct, since the true and student expressions will agree in the region $[2, 10] \times [2, 10]$ over which $(1 - x)(1-y)$ is positive.
- $\ln(1-x) + \ln(1-y)$ This answer will be marked as incorrect in the first three regions but as correct in the region $[-10, 2] \times [-10, 2]$.

The second answer here illustrates the need for using the negative as well as positive ranges for if the negative range was not included, the second answer would have been marked as being incorrect.

3.3.2.4 Restrictions of CALM Compare

There are types of answers which CALM Compare has difficulty in answering. For example, if the true answer is either very small or very large (in the order of $10^{\pm 5}$) then the tolerance must be set accordingly.

Another problem of the CALM Compare algorithm is that the meaning of expressions is lost as the recursive procedures progress. All that is left of the evaluation is the numerical values of the expression at particular values of the identifiers. The work in chapters 4 and 5 introduce expression trees and show how valuable they can be in understanding more about true and student answers. Chapter 5 shows a method of comparing answers without actually knowing the true answer in advance - therefore enabling students, in certain situations, to ask themselves questions!

This thesis, in the proceeding chapters, will expand on the work done by CALM by providing a method of marking students' answers when the range $[a, b]$ is unknown. It will also show, in Chapter 6, how additional but simple features can be built into a testing routine in order to make the assessment more reliable and robust. In particular there will be discussion on how restrictions on student answers can be imposed in order to alleviate some of the problems of marking answers.

3.4 Second Generation CALM - CALM II

From the experiences of writing the original CALM materials came the CALM II software. In order to bring CALM more up to date, the new software was written for the two major hardware platforms used in the computing community - both IBM PC (Microsoft Windows) and the Apple Macintosh. The materials were authored using Authorware Professional which supports easier and more advanced methods of interaction, such as clicking and dragging, textual input, hot-spots and hot-words. The mathematical content of the new CALM was geared towards the Scottish Highers' syllabus.

After noticing the importance of the assessment element in CALM, CALM II included a new element in the software - Self Assessment Questions (SAQs). Here, students were tested in a way which is less confrontational than in the test section. The Self Assessment tests tended to be more informative in that a greater degree of feedback was delivered throughout the assessment exercise. Predicted incorrect answers were programmed into the marking process (which still used CALM Compare) so that the teachers could prevent mistakes which were common amongst students.

The most interesting research about the way in which CALM II can be used to aid students was that of Moi [28] (see Chapter 2). Moi's thesis also provides an excellent educational review of Computer Aided Learning (CAL).

An end-of-module test section completes the CALM II software. A new version of this test is in production as this thesis is being completed and will be used in future educational research. This test is an improvement on the Mathwise UKMCC test mechanism which is described in Chapter 6.

Chapter 4

An Application of Data Structures and Algorithms Within a Computer-Based Teaching, Learning and Testing Environment

4.1 Introduction

With the ever increasing use of technology in the teaching and testing of mathematics, it is vital that better methods of student / keyboard communication of mathematics are found. This chapter concentrates on the description of a mechanism (now referred to as the Input Tool or IT) which has been built to reduce student concern about mathematical one-line (Fortran-like) input. The first part of the chapter will describe the features and aesthetics of the tool and the second part will describe the use of data structures and algorithms for its creation. The evaluation of its educational benefits will be described in this chapter and Chapter 6.

4.2 Features of the Input Tool

Since 1985, CALM software has relied heavily on students typing mathematical expressions into the computer. [7] highlighted the problems in computer-student interaction within a CAL in mathematics framework and specified the need for a way of representing mathematical input in a less computer science oriented way.

This and the previous chapter highlighted the need for improving the way in which students type mathematics into the computer. With this in mind, an Input Tool (IT) has been created which alleviates some of the concerns that students have with mathematical input. In questionnaires given to users of CALM, students reported that the key features of such an input tool are to:

- Reduce student concern about communicating mathematical expressions to the computer;
- Display student inputs in a more conventional two-dimensional math-like manner; and
- Give useful feedback on wrongly formed expressions (syntax etc.)

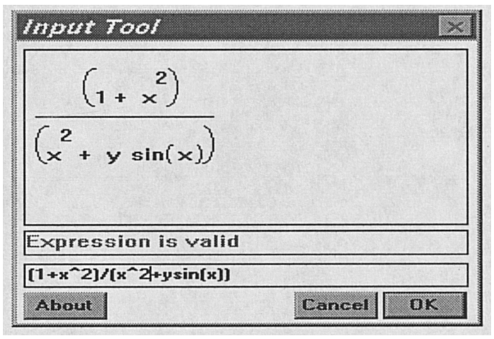
From a teacher's point of view, the IT should also:

- Be easily integrated (both functionally and aesthetically) into a teaching and learning and/or testing environment

The implementation of these criteria can be seen pictorially in diagram 4.2 below (and in chapter 6). The input tool employs a continual type-analyse-translate-redisplay sequence. That is, whilst students are typing an expression character-by-character, the IT continuously analyses the expression, translates it into something more meaningful and redisplays it, along with any relevant feedback, in mathematics notation.

Keyboard input appears in the box at the bottom of the Input Tool and the math-display and feedback appear at the top and centre respectively.

Although there are many commercial packages which contain an expression display tool, they are generally inaccessible to other software. Also, they are generally



$$\frac{\left(1 + x^2\right)}{\left(x^2 + y\ \sin(x)\right)}$$

Expression is valid

(1+x^2)/(x^2+ysin(x))

Figure 4.2: Diagram Showing the Input Tool

written without the educational need of students in mind and therefore are, as yet, not suitable for educational purposes.

4.3 Data Structures and Algorithms

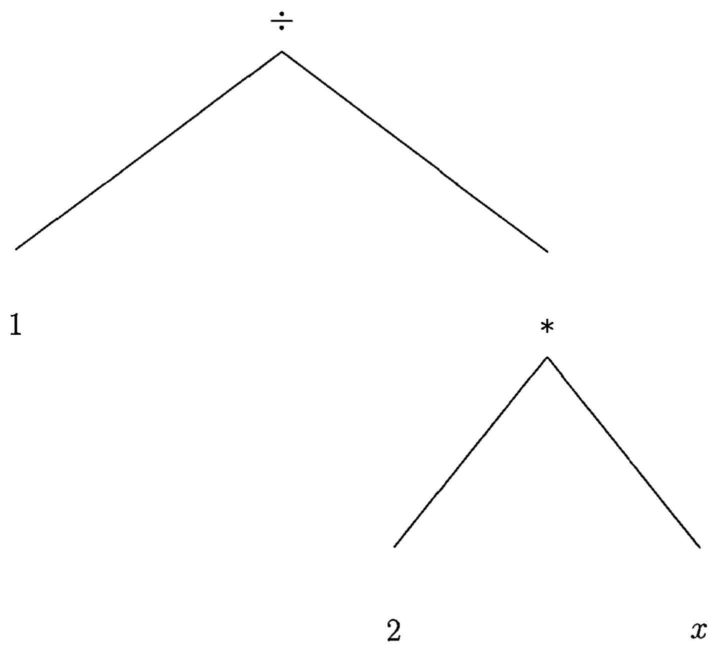
This section will describe in detail the data structures and algorithms required to create the Input Tool. The ideas presented are illustrated using a language-independent pseudo-code and the data structures appear in Turbo Pascal syntax.

The goal of the coding is to produce a binary tree representation of the input expression. The advantages of using binary trees within the context of mathematical CAL are numerous. Firstly, an expression in a binary tree requires no brackets and therefore removes superfluous parentheses from students' expressions. Traversal of an expression is made easier using recursive programming techniques and therefore trees can be used to compare true answers to student answers and evaluate expressions, identify numerators, denominators, powers and bases. Trees allow for easier algebraic differentiation (see next chapter), simplification of expressions and, perhaps most importantly, lead towards higher quality feedback for students. The use of trees, for which the Input Tool appeals to, is that of typesetting a one-line expression - that is, redisplaying an expression in mathematical, many-line format.

A binary tree is defined as follows[13]:

> **A binary tree is a tree which every node has at most two successors (children) where a tree is a connected graph which has no loops or paths leading from any vertex back to itself.**

The concept of the **root** of a tree is also used. The root of a tree is a vertex (or node) which has no parents. Strangely, perhaps, trees are usually represented pictorially with the root at the top. For example, the following diagram shows the expression `1/(2x)` represented in a binary tree:



The algorithms used for the creation of the binary trees, and hence the engine of the tool, can be broken down into the following topics:

4.3.1 Lexical Analysis

In order to give students feedback about the syntactical structure of expressions, it is important that the Input Tool understands the syntax of what the student has typed. Therefore, a one-line input must be broken down into tokens such as functions, constants, parentheses, real numbers, identifiers and operators. The elements of each set are shown in the following Turbo Pascal TYPE declaration statement:

```
TYPE
 FunctionType = (FTanh, FTan, FSqrt, FSqr, FSin, FSech,
   FSec, FLn, FLog, FFact, FExp, FCoth,
   FCot, FCosh, FCosech, FCosec, FCos, FArctanh,
   FArctan, FArcSinh, FArcSin, FArcCosh, FArcCos, FAbs);
 OperatorType = (Plus, Minus, Mult, Divide, Power, UnaryMinus);
 IdentifierType = (aa,bb,cc,dd,ff,gg,hh,ii,jj,kk,ll,mm,nn,oo,
   pp,qq,rr,ss, tt,uu,vv,ww,xx,yy,zz);
 DelimiterType = (OpenBracket, ClosedBracket);
 ConstantType = (FPi, ee);
```

The above declaration enables the Input Tool to understand the basic trigonometric and hyperbolic functions and simple constants and operators. This could naturally be extended to include, for example, operators such as the differential operator, or the partial differential operator. Just as easily, one could define aa, bb, cc, etc. to be constants rather than identifiers – however, for the purpose of display, this is not important.

All of these TYPEs can now be grouped together under one TYPE. Call this type ItemInfo and declare it as a VARIANT RECORD as follows:

```
TYPE

ItemInfo = Record
     CASE  NodeStatus:NodeType of
        FunctionNode : (WhichFunction   :FunctionType);
        OperatorNode : (WhichOperator   :OperatorType);
        IdentifierNode: (WhichIdentifier: IdentifierType);
        RealNode       :(WhichReal       :Real);
        DelimiterNode :(WhichDelimiter :DelimiterType);
        ConstantNode  :(WhichConstant   :ConstantType);
        NothingNode:();
   End; {ItemInfo}
```

That is, a particular token of type ItemInfo can be either a Function, an Operator, a Identifier, a Real, a Delimiter or a Constant. If the token is a function then it is called WhichFunction which is of type FunctionType. Similarly, if the token is an operator then it is called WhichOperator and is of type OperatorType $\cdots$ and similarly for the Identifiers, Reals, Delimiters and Constants.

Now assume that a typical one-line expression string can be broken down into an array of tokens whose elements can be one of the above types. For example, the expression `x^2` can be broken down into the array A where

```
A[1].NodeStatus=IdentifierNode, A[1].WhichIdentifier=xx,
A[2].NodeStatus=OperatorNode, A[2].WhichOperator=Power,
A[3].NodeStatus=RealNode and A[3].WhichReal=2.0.
```

Note that, for example, if the NodeStatus of a token is FunctionNode then all of WhichOperator, WhichIdentifier, WhichReal, WhichDelimiter and WhichConstant are all automatically undefined.

After establishing the need for such an array, it is important to know how to differentiate between the end of one token and the start of another. For example, it is vital that the software understands that `pisin(x)` means `pi*sin(x)` rather than `pi*s*i*n*(x)`.

The following pseudo-code illustrates the algorithm to do this.

```
Comment: We want to take a string called S and separate it
Comment:  into an array of tokens called A.
Comment:  Firstly, denoting S[i] to be  the ith character
Comment: in a string S,
Comment: check for unary minuses in the string S using
Comment: the following procedure and function;


Function Unary_Minus(WhichString:String; WhichChar:Char;
  Position:Integer):Boolean;
```

```
{
  Comment: This boolean function returns true if a particular
  Comment: character,
  Comment: WhichChar, at position i in the string
  Comment: (WhichString) is  a unary minus (denoted by % rather than -)

  Set Unary_Minus result to default FALSE;
  Comment:  Initialise the result of the function to FALSE
  If (WhichChar='-') and (Position=1) then set Unary_Minus to TRUE;
  Comment: The first character in string is a minus and therefore must
  Comment: be unary
  If (WhichChar='-') AND (WhichString[Position-1]='(') then
  Set Unary_Minus to be TRUE;
  Comment: any - sign after an open bracket must be unary
}

Procedure CheckForUnaries(VAR WhichString:String);

Comment: That is, this function takes a string called WhichString
Comment: and returns
Comment: it with unary minuses included  (% instead of -)

Declare i as an integer;
{
  For i:=1 to length(WhichString) do
    If the character at position i in WhichString is a UnaryMinus then
      set the character at position i of WhichString to be
      a '%' character;
}
```

```
Declare TempNumberCounter and TempStringCounter as INTEGER;

Comment:  These are counters used to count sequences of numbers
Comment: and characters respectively

Initialise TempNumberCounter and TempStringCounter to 1;
Initialise the Boolean variable ErrorInNumbers to FALSE;
Initialise ArrayCounter to 1;
Comment: Counts through the elements of A
Comment:  TempString and TempNumberString are used to build a temporary
Comment: string for sequences of characters and numbers
Comment: (including decimal points)
Comment: respecitively.

Comment: Denote S[i] to be the ith character of the string S
Comment: Denote A[i] to be the ith element of the array A

REPEAT
     If ( S[TempStringCounter] in ['a'..'z','A'..'Z'] ) AND
        ( TempStringCounter<=Length(S) )
      then
          {
           Append to TempString the character S[TempStringCounter];
           Increase TempStringCounter by 1;
          }
      else
          {
           TempString becomes S[TempStringCounter];
          {
UNTIL ( NOT ( S[TempStringCounter] in ['a'..'z','A'..'Z'] ) OR
           ( TempStringCounter>=Length(S) ) )

Comment:  That is, keep reading letters  until a
Comment:  non-letter character appears or the end of the input
Comment:  string S is reached

REPEAT
     If ( S[TempNumberCounter] in ['0'..'9','.'] ) AND
        ( TempNumberCounter<=Length(S) )
     then
         {
          Switch off internal type checking
          If NOT (MoreThanOneDecimalPlaceinString(TempNumberString))
          then
           Turn TempNumberString into a number called TempNumber
          Switch back on the internal type checking

          If TempNumber>MaxAllowableNumber or
           TempNumberString contains >1
            decimal places then set the Boolean variable
             ErrorInNumbers to TRUE
          If TempNumber<=MaxAllowableNumber AND
           (NOT ErrorInNumbers) then
           Append TempNumberString with the
            character S[TempNumberCounter];

          Increase TempNumberCounter by 1
         } else
            Set TempNumberString to be '-'  (or some indicator
            that it is  not a number)
UNTIL ( NOT (S[TempNumberCounter] in ['0'..'9','.'] ) OR
            (TempNumberCounter>Length(S) ) )



If TempNumberString='-' then
    {
     Comment: did not come across a number and hence need
     Comment: to check if it is
     Comment: predefined constant, function or just an identifier

     If TempString is a predefined function string then
        {
         A[ArrayCounter].NodeStatus:=FunctionNode;
         A[ArrayCounter].WhichFunction:=
          FunctionString2Type(TempString);
        } else
     If TempString is a predefined constant string then
        {
         A[ArrayCounter].NodeStatus:=ConstantNode;
         A[ArrayCounter].WhichConstant:=
         ConstantString2Type(TempString);
        } else
     If TempString is an identifier string then
        {
         A[ArrayCounter].NodeStatus:=IdentifierNode;
         A[ArrayCounter].WhichIdentifier:=
         IdentifierString2Type(TempString);
        } else
     If TempString is an Operator String then
        {
         A[ArrayCounter].NodeStatus:=OperatorNode;
         A[ArrayCounter].WhichOperator:=
     OperatorString2Type(TempString);
    } else
If TempString is a delimiter String then
    {
     A[ArrayCounter].NodeStatus:=DelimiterNode;
     A[ArrayCounter].WhichDelimiter:=
     DelimiterString2Type(TempString);
    } else
   {
    Comment:   The string TempString is in none of the
    Comment: above categories and therefore it must be
    Comment: a mixture of two (e.g. xsin)

    Comment:  Denote a stack as S.  Use the stack S to
    Comment: store (push) and
    Comment:  retrieve (pop) tokens (in string form) which are part
    Comment:  of the string TempString

    Comment: Now copy the string TempString to a string called
    Comment: MainString, say.  Also, call TString another
    Comment: temporary string which will
    Comment: be used to store tokens (in the form of strings)
    Comment: being read from the rear of MainString.
    Comment: A Boolean variable, ContinueSearch, will be used
    Comment: to signify whether or not the search for tokens should
    Comment: continue or not.

     Initialise the Stack S to contain no strings
     MainString:=TempString;

     REPEAT
```

```
Initialise TString to the null string  '';
 Initialise a BOOLEAN variable called
 ContinueSearch to TRUE;

REPEAT
   TString:=MainString[Length(MainString)-
    Length(TString)]+TString;
   Comment: That is, read from the rear of MainString and
   Comment: put the result in TString.
   If TString is a pre-defined constant or function then
    {
     If FunctionString2Type(TString) in [FCos,FSin,FCos,
        FTan,FSinh,FCosh,FTanh] then
     Comment: I.e. NOT an inverse function
      {
       Comment:Check to see the previous characters are
       Comment: NOT 'ARC'
       If (NOT(PreviousCharacters(MainString,
        Length(MainString)-
        Length(TString),Length(MainString)-
         Length(TString)-2))='ARC')
         then ContinueSearch:=FALSE;
      } else ContinueSearch:=FALSE;
    }
  UNTIL ( TString is a PreDefined Constant or Function) AND
        ContinueSearch=FALSE  OR
        (Length(TString)>=Length(MainString));
  If TString IS a PreDefinedFunction or Constant then
    {
     Push onto the stack S the string TString;
     Delete Length(TString) characters from the rear
```

```
              of MainString;
            } ELSE
            {
             Comment:  Must be just a single character variable
             Push last character of TString onto the stack;
             Delete the last character of MainString;
            }
       UNTIL (Length(MainString)=0)
       Comment:  The stack S has been filling up with
        tokens from the rear

Comment: of the string called MainString.  Now take off all the tokens
       Comment (i.e. pop the stack), filling up the array A as we go

       While The Stack S still has content DO
         {
          Pop the stack into a string called SString;
          if SString is a predefined function then
            {
              A[ArrayCounter].NodeStatus:=FunctionNode;
              A[ArrayCounter].WhichFunction:=
               FunctionString2Type(SString);
              If S still has content then Increase ArrayCounter by 1;
            } else
          If SString is a predefined Constant then
            {
              A[ArrayCounter].NodeStatus:=ConstantNode;
              A[ArrayCounter].WhichConstant:=
               ConstantString2Type(SString);
              If S Still has content then Increase ArrayCounter by 1;
            } else
```

```
      { Comment:  Must be an identifier in the stack S
        A[ArrayCounter].NodeStatus:=IdentifierNode;
         A[ArrayCounter].WhichIdentifier:=
           IdentifierString2Type(SString);
          If S Still has content then Increase ArrayCounter by 1;
        }
      }
} else
 {
   Comment: Encountered a number rather than an alpha-string -fill
    Comment: in the Array element corresponding to the real
     Comment: number that has been found.


      A[ArrayCounter].NodeStatus:=RealNode;
       A[ArrayCounter].WhichReal:=Number representation of
         TempNumberString;
     }

      If TempNumberString='-' then   Comment: not a number
         Delete from the string S the length of TempString else
          Delete from the string S the length of TempNumberString

        Increase ArrayCounter by 1;

      UNTIL (Length(S)=0) or ErrorInNumbers;
   }
```

The above process is perhaps best illustrated by example. Table 4.3 shows the values of the important variables in the algorithm and show the breakdown of the expression as the lexical analysis progresses.

C_1	C_2	C_3	C_4	C_5	C_6	C_7
1		1	0.123	6	0.123	'0.123/sin(pi*x)'
2	/	1	-	1	/	/sin(pi*x)
3	sin	4	-	1	SIN	sin(pi*x)
4	(	1	-	1	(	(pi*x)
5	pi	3	-	1	PI	pi*x)
6	*	1	-	1	*	*x)
7	x	2	-	1	x	x)
8	)	1	-	1	)	)

Table 4.3: Table Showing the Breakdown of an Expression During Lexical Analysis

The meaning of the columns $C_1 \cdots C_7$ are given in table 4.4 below:

Column	Meaning
C_1	ArrayCounter
C_2	TempString
C_3	TempStringcounter
C_4	TempNumberString
C_5	TempNumberCounter
C_6	A[ArrayCounter]
C_7	S

Table 4.4: Table Showing Column Names for Table 4.3

Now that this array has been created, there is a need to check for implied multiplication. For example, `xsin(x)` will have been separated into its tokens and stored in the array A as:

```
 A[1].NodeStatus=IdentifierNode, A[1].WhichIdentifier=xx,
A[2].NodeStatus=FunctionNode, A[2].WhichFunction=FSin,
A[3].NodeStatus=DelimiterNode, A[3].WhichDelimiter=OpenBracket,
A[4].NodeStatus=IdentifierNode, A[4].WhichIdentifier=xx,
A[5].NodeStatus=DelimiterNode, A[5].WhichDelimiter=ClosedBracket
```

However, there has been no provision made for implied multiplication. For example, xsin(x) really means x*sin(x) and it is important that the software understands this. The pseudo-code below takes the array A and inserts multiplication's where necessary.

This code uses two procedures. Procedure ImpliedMult tests for the need of Implied multiplication between two array elements of A and InsertMult inserts a multiplication node between two specified elements of A.

Note: In the following code, it is sometimes written, for example, `''If This.NodeStatus=OpenBracket''`. This should strictly be `''(This.NodeStatus=DelimiterNode) AND (This.WhichDelimiter=OpenBracket)''` but for simplicity, for former notation is sometimes used.

```
Procedure InsertMult(N1,N2:Integer; VAR A);
Declare i as an integer;
Declare MultNode as being of type ItemInfo;
{
 Comment: Procedure takes 3 parameters. Multiplication
 Comment: element will be inserted
 Comment: between elements N1 and N2 of the array A
 MultNode.NodeStatus:=OperatorNode;
 MultNode.WhichOperator:=Mult;
 Comment: create a multiplication node

 Comment: now shift up all nodes to the right of N1
 Comment: (N2 and higher) by one
 Comment: to make room for the multiplication node

 For i:=MaxElem-1 DownTo N2 do
  {
   A[i+1]:=A[i];
   Comment: where MaxElem is a constant equal to the maximum
   Comment: allowable elements in the array A; Note that this
   Comment:  assumes that there are less than
   Comment: <Maximum Elements in the Array> number of tokens in
   Comment: the array. If this is not true then an
   Comment: array overflow message will appear.  However, this is
   Comment: unlikely and tests can be made
```

```
    Comment so that this doesn't happen.
    Comment:  Now insert the multiplication node
    WhichArray[N2]:=MultNode
}


Procedure ImpliedMult(VAR A);
Declare i as an integer;
Declare P1 and P2 to be of type ItemInfo;
{
 For i:=1 to MaxElem -1 do
  {
   Comment: for each element in the array A do the following
   P1:A[i];  Comment: assign P1 to the ith element of A
   P2:=A[i+1];  Comment: assign P2 to the (i+1)th element of A
   If ( ( P1 is either a constant, Real or Identifier type ) AND
      (P2 is a Constant, Real, Identifier or Function) )  OR
      ( P2 is an OpenBracket) then
   {
     InsertMult(i,i+1,A);
     Comment: insert a Mult. node between i and i+1 in A
   } else
   If ( P1 is a Constant type) AND
      ( P2 is a Real, Identifier or Function type
       or an OpenBracket ) then
   {
    InsertMult(i,i+1,A);
    Comment: insert a Mult. node between i and i+1 in A
   } else
   If ( P1 is a ClosedBracket )  AND
      ( P2 is a Real, Identifier, Constant or Function type or
        OpenBracket ) then
    {
     InsertMult(i,i+1,A);
     Comment: insert a Mult. node between i and i+1 in A
    }
  }
}
```

The above algorithms and pseudo-code caters only for correctly formed expressions. Specifications for the Input Tool included the ability to update the expression display and the feedback boxes perpetually. That is, the tool should accept individual key presses and respond to them accordingly. The consequence of this is that students will frequently type syntactically incorrect expressions. Moreover, since the software appeals to the constructions of binary trees – an impossible task with incorrectly formed expressions – the expression must be syntactically correct. Therefore, provision must be made for these occurrences. The solution employed here is to transform incorrectly formed expressions into legal expressions by amending the input-String. Adding combinations of a special '?' item (declared as being a constant) and parentheses to the array of tokens A will not only create a legally constructed expression but will add to the quality of the feedback given to the student. For example, if the illegal expression `x^` is appended with the '?' constant then it will read `x^?`. This expression can now be displayed (along with a relevant feedback message) as follows:

$$x^?$$

Expecting something after the operator $\wedge$

To make the token array A legal, we iterate through all the elements of A, looking at the combinations and interactions of the adjacent elements. If the index of the iteration is called i, let THIS be the ith element, PREV be the (i-1)th element and NEXT be the (i+1)th element of A. Also, denote an element of A which has no content to be of type NothingNode. We can enforce rules on how to amend A according to the combinations of PREV, THIS and NEXT. These amendments are described in the following pseudo-code:

```
Procedure MakeLegalArray(VAR A:NodeArray);
Declare i as an integer;
Declare Prev, Next, This and TempItem to be of type ItemInfo (elements of
 array);
Comment: Now iterate through all the elements of A
For i:=1 to MaxElem do
 {
 Comment: Work out This, Prev and Next
 This:=A[i];
 If i=1 then Prev.NodeStatus:=NothingNode else Prev:=A[i-1];
 Comment: allow for the fact that if i=1 then there is no A[i-1]
 If i references the last non-empty element of A then
  Next.NodeStatus:=NothingNode else Next:=A[i+1];

 Case This.NodeStatus of Comment: What type of element is A[i] (This)?
    FunctionNode:
     {
     If Prev is a function then
     Comment: function must have argument in brackets - put a ? argument
     After Prev, insert three elements between Prev and this as (?);
     Comment: eg SINSIN -> SIN(?)SIN

     Case Next.NodeStatus of Comment look at function - Next combinations
        NothingNode, ConstantNode, IdentifierNode, RealNode,
        FunctionNode, OperatorNode or ClosedBracket:
         {
          Insert (?) after This Comment: e.g. SINPI -> SIN(?)PI
         }
 case end
IdentifierNode, ConstantNode, RealNode:
 {
 If Prev.NodeStatus=FunctionNode then
 Before This (and after function Prev) put (?);
 Comment: eg SINx -> SIN(?)x
 Comment: can have any Next after an identifier, constant or real
 }
OperatorNode:
 {
 Case This.WhichOperator of Comment: which operator is This?
    Plus, Minus, Mult, Divide, Power:
     { Comment: binary operators
     Case Prev.NodeStatus of Comment: Prev-BinOperator combinations
      OperatorNode: After Prev insert a ? Comment: eg ++ -> +?+
      FunctionNode: After Prev insert (?) Comment: SIN+ ->SIN(?)+
      OpenBracket: After Prev insert (?) Comment: eg (+ -> ((?)+
      NothingNode:
      {
       If This.WhichOperator=Minus then Comment: should be a unary
        Make This a UnaryMinus rather than Minus;
      }
     end case Comment: end of Prev.NodeStatus case
     Case Next.NodeStatus of Comment: This-Next combinations
      OperatorNode: Add after This ?; Comment: eg ++ -> +?+
      NothingNode: Add a ? after This;  Comment: eg ...+ -> ...+?
     end case; Comment: end Next.NodeStatus case
     }
    UnaryMinus:
     {
     If Prev.NodeStatus<>NothingNode then Make This a BinaryMinus;
Case Prev.NodeStatus of Comment: Prev-% combinations
 FunctionNode: Add (?) after Prev Comment: eg SIN% -> SIN(?)-
 OperatorNode: If Prev.WhichOperator<>UnaryMinus then
 {
  make This into a Binary Minus;
   Insert ? before This Comment: eg +% -> +?-
  }
 end case Comment: end Prev.NodeStatus case
 Case Next.NodeStatus of Comment: %-Next combinations
  OperatorNode: Add ? after This; Comment eg %+ -> %?+
  ClosedBracket: Add (?) after This; Comment: eg %) -> %(?))
  NothingNode:
  {
   Make This into a BinaryMinus;
    Insert ? after This; Comment e.g. ...% -> ...-?
   }
  end case; Comment: end Next.NodeStatus case
  } Comment: end of UnaryMinus:
end case; Comment: end This.WhichOperator case
} Comment: end OperatorNode case
NothingNode:
 {
 Case Prev.NodeStatus of
  OperatorNode: Insert ? after Prev; Comment: eg + -> +?
  FunctionNode: Insert (?) after Prev; Comment: eg SIN -> SIN(?)
  OpenBracket: Insert ?) after Prev; Comment: ( -> (?)
 end case; Comment: end of Prev.NodeStatus case
 } Comment: end of NothingNode case
 DelimiterNode:
  {
  Case This.WhichDelimiter of
```

```
        OpenBracket:
        {
         Comment: can have any Prev
         Case Next.NodeStatus of
          OperatorNode: Insert ? before Next; Comment: eg (+ -> (?+
          NothingNode: Add ?) after This; Comment: eg ( -> (?)
          ClosedBracket: Add ? before Next: Comment: eg () -> (?)
         end case; Comment: end of Next.NodeStatus case
        } Comment: end of OpenBracket case
        ClosedBracket:
        {
         Case Prev.NodeStatus of
          OperatorNode: Add ? Before This; Comment: eg +) -> +?)
          FunctionNode: Add (?) after Prev; Comment: eg SIN) -> SIN(?))
          NothingNode: Add (? before This; Comment: eg )... -> (?)...
          OpenBracket: Add ? After Prev; Comment: eg () -> (?)
         end case; Comment end Prev.NodeStatus case
        } Comment: end ClosedBracket case
      end case; Comment: end of This.WhichDelimiter case
    } Comment: end DelimiterNode case
end case; Comment: end This.NodeStatus case
} Comment: end of FOR loop

If Number of Closed brackets in Array A <> Number of Open then
{
 If number of closed > Number of Open then
   Insert in start of array the requisite number of Open brackets
 else
 If Number of open > number of Closed then
   Insert at end of array the requisite number of closed brackets
 Comment: since the number of closed<>number of open
```

```
  Comment: then put a ? at end
  Comment: of array to signify an error occurred.
  Comment: Otherwise it will look like
  Comment: no error occurred in the redisplay of the expression in 2-d
  Add ? to end of array;
 }
 Comment: after putting in requisite brackets, make sure there are no ()
 Comment: situations in the array
 FOR i:=1 to MaxElem do
 {
    If A[i] is an OpenBracket and A[i+1] is a ClosedBracket then
     Insert a ? between A[i] and A[i+1];
 }
End of procedure
```

The above code not only enables the construction of the binary trees but also gives rise to the ability of building in good feedback to students as to the current status of their expression. Throughout, the pseudo-comments have been used to clarify the meaning of the code. More importantly, it is now possible to provide students with quality feedback about the construction of their expression. The feedback messages for two wrongly adjoining tokens are shown in Table 4.5. Notice that Prev-This combinations encompass the same messages as for identical This-Next combinations. The Next column is included as the Next token may be blank (indicated by a dash) which needs to be accounted for. For example, a blank after a function is an illegal expression.

Since the above procedure introduced new constants ("?") and parenthesis, it is necessary to re-call the ImpliedMult procedure. For example, if the token array contained tokens from the expression `Sinx` then it will have been transformed into one containing tokens from `Sin(?)x`. This expression is a product of `Sin(?)` and `x` and will therefore need a multiplication node between them.

Prev	This	Next	Ex. Input	Ex. Output	Message
Fn	Fn	-	`SINSIN`	`SIN(?)SIN(?)`	SIN must have argument in brackets
BinOp	Fn	-	`-SIN`	`-SIN(?)`	SIN must have argument in brackets
Var	Fn	-	`xSIN`	`xSIN(?)`	SIN must have argument in brackets
Const	Fn	-	`PISIN`	`PISIN(?)`	SIN must have argument in brackets
%	Fn	-	`-SIN`	`-SIN(?)`	SIN must have argument in brackets
Delim. (	Fn	-	`(SIN`	`SIN(?)?`	Missing) to complete expression
Delim.)	Fn	-	`)SIN`	`??SIN(?)`	Need legal expression inside ( )
Fn	BinOp	-	`SIN-`	`SIN(?)-?`	Expecting something around -
BinOp	BinOp	-	`--`	`??-?-?`	Expecting something around -
Var	BinOp	-	`x-`	`x-?`	Expecting something around -
Const	BinOp	-	`PI-`	`PI-?`	Expecting something around -
%	BinOp	-	`--`	`??-?-?`	Expecting something around -
Delim. (	BinOp	-	`(-`	`(-?)?`	Missing) to complete expression
Delim.)	BinOp	-	`)-`	`??-?`	Need legal expression inside ( )
Fn	Var	-	`SINx`	`SIN(?)x`	SIN must have argument in brackets
Delim. (	Var	-	`(x`	`x?`	Missing) to complete expression
Delim.)	Var	-	`)x`	`??x`	Need legal expression inside ( )
Fn	Const	-	`SINPI`	`SIN(?)pi`	SIN must have argument in brackets
Delim. (	Const	-	`(PI`	`PI?`	Missing) to complete expression
Delim.)	Const	-	`)PI`	`??PI`	Need legal expression inside ( )
Fn	%	-	`SIN-`	`SIN(?)-?`	Expecting something around -
BinOp	%	-	`--`	`??-?-?`	Expecting something around -
Var	%	-	`x-`	`x-?`	Expecting something around -
Const	%	-	`PI-`	`PI-?`	Expecting something around -
%	%	-	`--`	`??-?-?`	Expecting something around -
Delim. (	%	-	`(-`	`(-?)?`	Missing) to complete expression
Delim.)	%	-	`)-`	`??-?`	Need legal expression inside ( )
Fn	Delim. (	-	`SIN(`	`SIN(?)?`	Need legal expression inside ( )
BinOp	Delim. (	-	`-(`	`(-?)?`	Need legal expression inside ( )
Var	Delim. (	-	`x(`	`x??`	Need legal expression inside ( )
Const	Delim. (	-	`PI(`	`PI??`	Need legal expression inside ( )
%	Delim. (	-	`-(`	`(-?)?`	Need legal expression inside ( )
Delim. (	Delim. (	-	`((`	`???`	Need legal expression inside ( )
Delim.)	Delim. (	-	`)(`	`????`	Need legal expression inside ( )
Fn	Delim.)	-	`SIN)`	`?SIN(?)`	Missing (in expression
BinOp	Delim.)	-	`-)`	`?(??-?)`	Missing (in expression
Var	Delim.)	-	`x)`	`?x`	Missing (in expression
Const	Delim.)	-	`PI)`	`?PI`	Missing (in expression
%	Delim.)	-	`-)`	`?(??-?)`	Missing (in expression
Delim. (	Delim.)	-	`()`	`?`	Need legal expression inside ( )
Delim.)	Delim.)	-	`))`	`???`	Need legal expression inside ( )

Table 4.5: Table showing error messages for incorrect inputs within the Input Tool
Note that BinOp stands for $+$, $-$, $*$, $/$ or $\wedge$

4.3.2 Postfix Notation – Dijkstra's Algorithm

In order to create a binary tree of the input expression (using the token array), it is easier to transform the token array from its current infix form to its corresponding postfix form. The infix form of an expression has operands on either sides of its operators whereas operators in postfix expressions appear after their operands. For example, the infix from of the expression `x^2` has corresponding postfix form `x2^`. This process also eliminates the need for parenthesis – for example `x^(2*x)` in postfix notation is `x2x*^`.

Dijkstra's Algorithm[1] uses a stack to store tokens at particular stages. It also uses the notion of the precedence of a token.

The precedence of the tokens used in the Input Tool are shown in Table 4.6 below. The table shows that Functions have the highest precedence whereas Delimiters have the lowest.

Delimiters	0
Identifiers, Reals and Constants	1
Operators Plus and Minus	2
Operators Mult and Divide	3
Operator Power	4
Operator Unary Minus	5
Functions	6

Table 4.6: Table showing the precedence of tokens

The following pseudo-coded procedure receives a token array A (tokens in infix form) and creates another token array B in postfix form:

```
Declare S as a Stack whose elements are Integers -
 the indices of the array;
Declare IC and OC as Integers;
Comment: used to count through the elements of
Comment: A and B respectively
Declare X to be an Integer;
```

[1]Dijkstra's Algorithm for converting an expression to postfix notation was obtained from unpublished Data Structures and Algorithms Lecture notes by Ian E. Aitchison, Department of Computing and Electrical Engineering, Heriot-Watt University, Edinburgh.

```
Initialise B;
Initialise IC and OC to 1;

While A[IC] has content do
 {
  If A[IC] is a Real, Identifier or Constant then
   {
    B[OC] is assigned to be equal to A[IC];
    Increase OC by 1;
   } else
  If A[IC] is a Delimiter then
   {
    If A[IC] is an OpenBracket then Push IC onto the stack S else
    If A[IC] is a ClosedBracket then
     {
        Pop the stack into the integer X;
        While A[X] is not a delimiter do
         {
          B[OC] is assigned to be equal to A[X];
          Pop the stack S into the integer X;
          Increase OC by 1
         }
     }
    } else
    {
        While the Stack S is non-empty AND the Precedence of
        A[IC] <=A[integer currently in the top of the stack S] do
         {
          Pop the stack S into the integer X;
          B[OC] is assigned to equal A[X];
          Increase OC by 1;
         }
        Push the value of IC onto the stack S;
   }
  Increase IC by 1;
 }
 While the stack is non-empty do
  {
        Pop the stack S into the integer X;
        B[OC] is assigned to equal A[X];
        Increase OC by 1;
  }
```

An example of this process is given in Appendix A.

4.3.3 Binary Tree Creation

An array of tokens has now been created in which the tokens appear in postfix order. From this array it is fairly easy to create a binary tree of the expression. The following pseudo-coded procedure shows how to construct a tree given an array of tokens in postfix order:

Within this code a stack is used. This time, however, the elements of the stack are themselves binary trees. Call the array sent to the procedure A and the binary tree that it produces T.

```
Declare IC as an Integer; Comment: used as an index counter for A
Declare T1,T2 and CTree to be Binary Trees;
Declare S to be a stack whose elements are binary trees;

Initialise the stack S;
Initialise the tree T;
Initialise IC to 1;

While A[IC] is non-empty do
 {
  If A[IC] is a non-unary-minus operator then
   {
    Let T1 equal the tree contained in the top of the stack S;
    Pop the Stack S -- no need to return the result;
    Let T2 equal the tree contained in the top of the stack s;
    Pop the Stack S -- no need to return the result;
    Let CTree be the tree with A[IC] at the root,
      T2 as the Left SubTree and T1 as the Right SubTree;
    Push CTree onto the stack S;
   } else
  If A[IC] is a unary minus or a function then
   {
    Let T2 equal the tree contained in the top of the stack S;
    Pop the Stack S -- no need to return the result;
    Let T1 be the nill tree (nothing on it)
    Let CTree be the tree with A[IC] at the root,
     T2 as the left SubTree and T1 as the Right SubTree;
    Push CTree onto the stack S;
   } else
   {
    Let CTree be the tree with A[IC] at the root and the nill tree as its
      left and right SubTrees;
    Push CTree onto the Stack S;
   }
  Increase IC by 1;
 }
 Let T be the tree contained in the top of the stack S
```

The method in which Binary Trees are created is illustrated in the following example.

Given the postfix expression `ab-2^` (representing $(a-b)^2$) then the following table gives the construction of the Binary Tree.

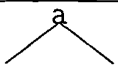
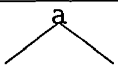
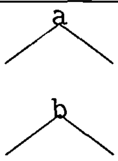
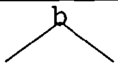
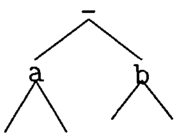
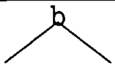
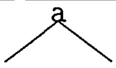
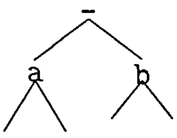
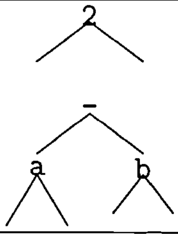
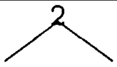
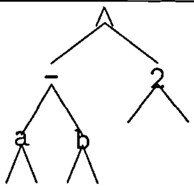
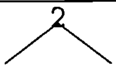
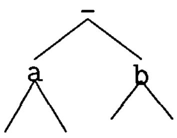
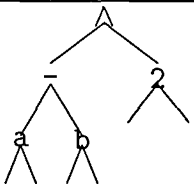
Ref	1	2	3	4	5
A	Symbol	Stack(S)	Tree (T1)	Tree(T2)	Tree (T)
B	a				
C	b				
D	-				
E	2				
F	∧				

Table 4.7: Table Showing Binary Tree Construction

The binary tree for the postfix expression (a-b)∧2 is created, using the pseudo code above as follows:

- Initialise the Stack (S)
- Initialise the Tree(T)
- Get the first character in the postfix expression (a)
- This is **not** an operator and hence construct whichtree with a as the root and Whichtree∧.LeftPtr = Whichtree∧.RightPtr = nil

- Push WhichTree onto the Stack S (Ref B2)
- Get the next character (b)
- This isn't an operator and hence add b to the top of the stack like we did with the character a (Ref C2)
- Get the next character (-)
- - is an operator and so assign T1 to be the top of S (Ref D3).
- Pop the stack (delete top element of stack and make next element the top)
- assign T2 to be the top of the new stack (Ref D4)
- Pop the stack again
- Assign WhichTree to be a Binary Tree with - as its root, T2 as its left SubTree and T1 as its right SubTree (ref D2)
- Get the next character (2)
- 2 is not an operator so push 2 onto the top of the stack S (giving the stack as shown in Ref E2)
- Get the next character ($\wedge$)
- $\wedge$ is an operator and so carry out a similar procedure as with - to get Ref F2.

Notice that at each stage, T is defined to be the top of the stack (Refs: B5, C5, D5 and E5).

4.4 Displaying the Expression

Now that the binary tree of the expression has been created, it can be used to display the expression in mathematical notation.

A first attempt to display an expression held in a tree was to create its corresponding LaTeX code [41]. LaTeX is a universal mathematics typesetting language which can be converted into a .DVI file which can then be viewed using a .DVI viewer. This process involved a recursive procedure which traversed the binary tree of an expression, creating LaTeX code.

Although LaTeX creates a very high quality typeset expression, the process of obtaining the picture of it is slow - in the order of 6 seconds on a PC 486 33MHz machine. This is particularly the case if the LaTeX process is to accommodate the continuous (as-you-type) re-display of expressions. This solution also requires the disk-space-consuming LaTeX software and fonts being on the computer. As the next chapter explains, students did find this re-display useful to check syntax of expressions but the following "in-house" solution has since proved to be more popular.

For this task, we will assume that each character takes up an equal amount of space on the screen and each token will calculated to be positioned at the point (x, y) from a fixed origin $(xOrigin, yOrigin)$. Now assume that associated with each node on a binary tree are the integer data fields *Horiz*, *Above*, *Below*, $TopLeftX$, $TopLeftY$ and *Bracket*. Here, *Horiz* stores the horizontal displacement needed to display the token on a particular node and *Above* and *Below* store the vertical displacements. *Bracket* acts as a boolean decision as to whether or not a node needs to be bracketed and $TopLeftX$ and $TopLeftY$ store the position to place the token on the screen.

The following pseudo-code calculates the values of *Horiz*, *Above*, *Below* and *Bracket* for the Tree T and each SubTree of T. To aid in describing the recursive nature of this code, call the procedure which calculates the items described CALC(VAR T:BTree).

```
Comment:  This pseudo-code describes a recursive procedure which
Comment: takes in a binary tree T
Comment: and returns it with information pertaining to the position of
Comment: the tokens stored within its nodes.


Comment: call the pointer which points to nothing Nil


Procedure CALC(VAR T:BTree);  Comment:  Return T which is a binary tree
```

```
Declare i, NumBrackets and w to be integers;
Declare Leaf1 and Leaf2 to be Binary trees;

If T does not point to nil (empty tree) then
 {
  Iterate i from 1 to (the number of SubTrees of the tree T)
   {
    If i = 1 (left SubTree) then
     {
        Assign Leaf1 to be the Left SubTree of T;
        Make a recursive call to CALC with argument as Left SubTree of T;

        If Left SubTree of T needs to be bracketed then
         Assign the Bracket data
          field on Leaf1 to be 1 otherwise assign it to 0;
        On Leaf1 and all its SubTrees, shift to the right the tokens
          by <bracket> spaces (increase TopLeftX by 1 if bracket=1
           or 0 if bracket=0);
        Assign the Horiz data field of T to be: Itself + (2*Bracket);
     } else
    If i = 2 (right SubTree) then
     {
        Assign Leaf2 to be the Right SubTree of T;
        Make a recursive call to CALC with argument as
         Right SubTree of T;
        If Right SubTree of T needs to be bracketed then
         Assign the bracket data
          field of Leaf2 to be 1 otherwise assign it to 0;
        On Leaf2 and all its SubTrees, shift to the right the tokens
          by <bracket> spaces;
        Assign the Horiz data field of T to be:  Itself + (2*Bracket);
      }
    } Comment: end iterate

Assign w to be the number of characters that the token on T needs to be
  displayed (i.e the width of T);
Assign the TopLeftX, TopLeftY data fields on T to be XOrigin and YOrigin
  respectively;

If T is a Constant, Identifier or Real then
        {
         Assign T.Above data field to be 0;  Comment: On one line
         Assign T.Below data field to be 1;  Comment: takes up one line
         Assign T.Horiz data field to be w;
        } else
If T is an Operator then
        If T is a Plus, Minus or a Multiplication then
                {
                 Assign T.Above to be the Max of
                  Leaf1.Above and Leaf2.Above;
                 Assign T.Below to be the Max of
                  Leaf1.Below and leaf2.Below;
                 Assign T.Horiz to be the sum of
                  Leaf1.Horiz and Leaf2.Horiz;
                 On Leaf2 and SubTrees of Leaf2, shift to the right the
                  positions of tokens by Horiz+w places;
                 On Leaf1 and SubTrees of Leaf1, shift downwards the
                  positions of tokens by T.Above-Leaf1.Above places;
                 On Leaf2 and SubTrees of leaf2, shift downwards the
                  positions of tokens by T.Above-Leaf2.Above places;
                } else
```

```
If T is a UnaryMinus then
        {
         Assign T.Above to be Leaf1.Above;
         Assign T.Below to be Leaf1.Below;
         Assign T.Horiz to be w+Leaf1.Horiz;
         On Leaf1 and SubTrees of Leaf1, shift to the right the
          positions of tokens by w places;
        } else
If T is a Divide then
        {
         Assign T.Above to be Leaf1.Above+Leaf1.Below;
         Assign T.Below to be Leaf2.Below+1;
         Comment: +1 to allow for divide sign
         Assign T.Horiz to be the Max of Leaf1.Horiz and
          Leaf2.Horiz;
         Comment:  divide sign must be long enough to encapsulate
         Comment: either the numerator or denominator
         Comment: whichever is the bigger.
         On Leaf2 (denominator),and SubTrees of Leaf2, shift down
          the positions of tokens by Leaf1.Above +
            Leaf1.Below +1;
         On Leaf1 and SubTrees of Leaf1, shift right the
          positions of tokens by
           Round(Int((1+T.Horiz-Leaf1.Horiz)/2);
         Comment:  centre the numerator
          above the horizontal divide sign
         On leaf2 and SubTrees of Leaf2, shift right the
          positions of tokens by
           Round(Int((1+T.Horiz-Leaf2.Horiz)/2);
         Comment:  centre the denominator
          below the horizontal divide sign
```

```
                } else
        If T is a Power then
                {
                 Assign T.Above to be Leaf2.Above+
                  Leaf2.Below+Leaf1.Above;
                 Comment: i.e. to be How much space the power needs
                 Comment: for above
                 Comment:, how much power needs for below
                  and how much base needs
                 Comment: for above
                 Assign T.Below to be Leaf2.Below;
                 Assign T.Horiz to be Leaf1.Horiz+Leaf2.Horiz;
                 Comment: horizontal spacing is spacing for base
                 Comment: + for power
                } else
If T is a function then
        {
         Assign T.Above to be Leaf1.Above;
         Comment:  Above space needed for function is the above space
         Comment: needed for argument of function (left SubTree)
         Assign T.Below to be Leaf1.Below;
         Comment: Below space needed for function is the below space
         Comment: needed for argument of function (left SubTree)
         Assign T.Horiz to be Leaf1.Horiz+w;
         Comment: Horizontal space is the horizontal space needed for
         Comment: the argument of the function + the space needed to
         Comment: display the function name itself
        }
}
```

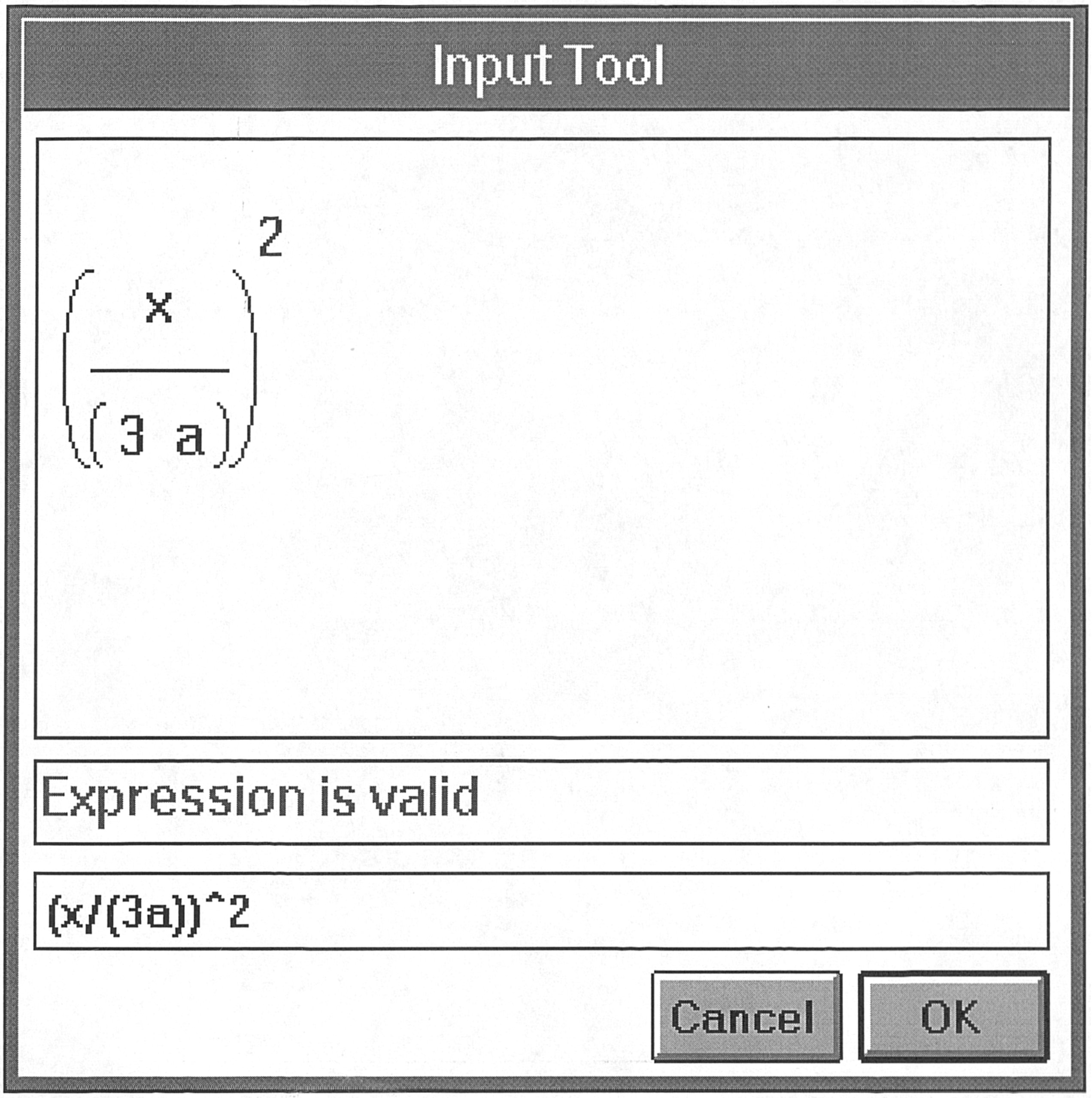
From the information contained in the tree, the co-ordinates $(xPos, yPos)$ and

$(x1Pos, y1Pos)$ can be calculated. Here, $(xPos, yPos)$ is the co-ordinate position of any token on the screen. For bracket tokens, $(xPos, yPos)$ defines the starting position of the bracket and $(x1Pos, y1Pos)$ defines the finishing position. Similarly, if the token is a divide then $(xPos, yPos)$ defines the starting position of the divide and $(x1Pos, y1Pos)$ define the finishing position The following pseudo-code produces an array A where each element of A contains the value of $(xPos, yPos)$, $(x1Pos, y1Pos)$ and a string S for the contents of the elements.

Again, because of the recursive nature of the following code, it will be described as a procedure. This procedure, called CalcArray, has three parameters; the array A which needs to be created, the tree T holding the expression and corresponding information and C, a counter used to iterate through the elements of A.

```
Declare ItemString as a String;
Declare x,dx,y,dy and i as integer;

Comment: denote the pointer which points to nothing as Nil;

If T<>Nil then
    {
     Shift:=0;
     If T is a Plus, Minus or Mult operator then
        Assign Shift to be the horiz field of the left SubTree of T;
     Assign A[C].XPos to be T.TopLeftX+Shift;
     Assign A[C].YPos to be T.TopLeftY+T.Above;
     Assign A[C].S to be the String representation of T.Item;
     If T was a divide then
        {
         Comment: work out start and finishing point of the divide
         Assign A[C].X1Pos to T.TopLeftX+Shift+<length of A[C].S>-1;
         Assign A[C].Y1Pos to T.TopLeftY+T.Above;
         Assign A[C].S to be '/' sign;
        }
        If T.Bracket=1 then
{
 Increase C by 1;
 Assign X to be T.TopLeftX-1;
 Assign dx to be T.Horiz;
 Assign y to T.TopLeftY;
 Assign dy to T.Above+T.Below;

 If (y-(y+dy-1))=0  then  Assign A[C].S to the '(' character else
     Assign A[C].S to the '[' character;
 Assign A[C].XPos to x;   Assign A[C].YPos to y;
 Assign A[C].X1Pos to x;  Assign A[C].Y1Pos to  y+dy-1;
 Increase C by 1;
 If (y-(y+dy-1))=0 then  Assign A[C].S to the ')' character else
     Assign A[C].S to the ']' character;
 Assign A[C].XPos to x+dx-1;   Assign A[C].Ypos to y;
 Assign A[C].X1Pos to x+dx-1;  Assign A[C].Y1Pos to y+dy-1;
}

If the left SubTree of T is not Nil then Increase Count by 1;
Call CalcArray recursively with the same array A, the left SubTree of T
      and the value of C;
If the right SubTree of T is not Nil then Increase Count by 1;
Call CalcArray recursively with the same array A, the right SubTree of T
      and the value of C;
                  }
```

For example, the tree of the expression `(x/(3a))^2` can be processed to produce the displayed expression shown in figure 4.3.



$$\left(\frac{x}{(3\ a)}\right)^2$$

Expression is valid

```
(x/(3a))^2
```

Figure 4.3: A Typical Expression displayed in the Input Tool

Table 4.8 gives the values of $(xPos, yPos)$ and $(x1Pos, y1Pos)$ co-ordinates of the tokens. Note square brackets has been used for the outer braces to distinguish them from those surrounding $3a$.

Token	xPos	yPos	x1Pos	y1Pos
[	1	2	1	4
x	4	2	4	2
divide	2	3	6	3
(	2	4	2	4
3	3	4	3	4
a	5	4	5	4
)	6	4	6	4
]	7	2	7	4
2	8	1	8	1

Table 4.8: table showing positions of tokens for $\left[\frac{x}{3a}\right]^2$

4.4.1 Platform Specific Considerations

Now that the algorithms for the engine of an Input Tool have been established, a decision was needed to be made on how to deliver it to the students / users. Chapters 5 and 6 will show that such an Input Tool would be useful in many pieces of CAL software and with this in mind it was important to seek a method of delivery which could port into these software's. The standard way of creating code and/or applications which many differing applications can use is that of DLLs (see [12]). A DLL (Dynamic Link Library) is code which can be loaded dynamically (during run-time) by a number of programs. This loading is often via a simple function call. It was decided, therefore, to create an Input Tool DLL which interfaced with Microsoft Windows.

Chapter 5

DRIFT

5.1 Introduction

This chapter describes an educational experiment designed to gain formative evaluation on a student-computer communication within a mathematics environment. The experiment used, as its base, a simple differential calculus package which enabled students[1] to test themselves in applying the rules of differentiation.

It is important to note at the outset that this chapter, because of small the number of students involved (8 students in all), makes no attempt to verify statistically any findings. It is included as a stepping stone to the following chapter in that observations made during the DRIFT experiment have proved invaluable in the development of the Input Tool.

A Pre-Post Test evaluation was considered but the small number of students and the affect on their education prevented this.

In addition, the work in this chapter describes the problems of marking answers when students set their own questions.

5.2 The Experiment Procedure

Eight students were, for six fifty-minute sessions, given expressions to differentiate. The problems were similar to those found in the Scottish Higher mathematics paper.

[1]the students referred to in this chapter were from St. Kentigern's Academy in Blackburn, West Lothian

A student typed the expressions (i.e. the questions) into the computer. The student could do their working on paper but the answers were input into the DRIFT package. During the sessions there were 550 mathematical expressions input into the computer which were subsequently analysed. The way in which this was done will be explained as this chapter progresses.

5.3 An Overview of the DRIFT package

5.3.1 The Front-End

DRIFT (Differentiation Rules Intelligent Feedback Tool) is a package designed to help students use the "Drill and Practice" technique of learning and applying the rules of differentiation.

The version of DRIFT used for the experiment employed a very simple DOS based menuing system. This menu is shown below:

```
*********************************************************************
       Differentiation Rules - Intelligent Feedback Tool  (DRIFT)
             Last Updated: 17-03-94     (c) David G Wild
*********************************************************************
        0 :     Input Your Personal Details
        1 :     Create / Change Expression
        2 :     View Expression
        3 :     Differentiate the Expression
        4 :     HELP   (about the program)
        B :     Background Theory of differentiation + examples
        C :         Change the Help Level
        5 :     Quit  DRIFT



   No expression installed - use option 1


Enter a number and press RETURN :
```

This menu encapsulated all the features which were available in the software. Using Option 0, students typed in their names and an assigned computer number which was used to identify them for the ensuing formative analysis.

Options 1 and 2 allowed students to install an expression into the software and see their input expression in a more meaningful way. The precise way in which this was done will be described in more detail later in this chapter.

Option 3 prompted the student to input the derivative of the their inputs. This option also gave the student feedback as to whether they differentiated the expression correctly and displayed a method of tackling the differentiation. This will also be explained in more detail later in the chapter.

Option 4 gave help on how to use the package. Option B displayed some background theory on the different rules of differentiation together with some illustrative examples.

Option C enabled the student to change the degree of help given in option 3 and option 5 quit the package.

5.3.2 Recording Student Activity

DRIFT was programmed to gather details of student activity. Every key that a student pressed was noted and written to a file. This enabled the information to be gathered on what the students were doing and when whilst using the computer. This information, together with what the student wrote on paper, was used to give some evaluation of the Input Tool.

5.3.3 Differentiating Student Inputs

The expressions input by students are differentiated by DRIFT. The algorithms to do this appear in pseudo code below. These algorithms are used to provide students with a method of differentiating their expression.

```
Comment: Due to the recursive nature of the following code, we use
Comment: the following recursive procedure, called
Comment: Differentiate(VAR T1:BTree; T2:BTree);
```

```
Comment: Declare many trees such as u, v, du, dv, udv, vdu etc...
Comment: The left subtree of T2, say, will be denoted T2^.LeftPtr
Comment: and similarly the right subtree of T2, say,
Comment: will be denoted T2^.RightPtr

Comment: MakeTree(NewTree,LeftTree,Root,RightTree) is a procedure
Comment: which creates the tree NewTree with root called Root,
Comment: Left Subtree called LeftTree and Right
Comment: Subtree called RightTree



Procedure Differentiate(VAR T1:BTree; T2:BTree);
{
If T2<>Nil then
        {
         If root of T2 is an operator then
        {
        CASE T2.WhichOperator of
        Plus:
                {
                 Assign tree u to T2^.LeftPtr;
                 Assign tree v to T2^.RightPtr;
                 Differentiate(du,u);
                 Differentiate(dv,v);
                 MakeTree(T1,du,+,dv);
                 Comment: put together to make
                 Comment: summation rule D(u+v)=D(u)+D(v)
                }
         Minus:
                {
```

```
          Assign tree u to T2^.LeftPtr;
          Assign tree v to T2^.RightPtr;
          Differentiate(du,u);
          Differentiate(dv,v);
          MakeTree(T1,du,-,dv);
          Comment: put together to make
          Comment: difference rule D(u-v)=D(u)-D(v)
         }


Mult:
         {
          Assign tree u to T2^.LeftPtr;
          Assign tree v to T2^.RightPtr;
          Differentiate(du,u);
          Differentiate(dv,v);
          MakeTree(udv,u,*,dv);
          Comment: create the tree called udv with
          Comment: u as left subtree,
          Comment: dv as right subtree and * as root
          Differentiate(du,u);
          Comment: differentiate u and get du
          MakeTree(vdu,v,*,du);
          Comment: create the tree called vdu with v as left,
          Comment: du as right and * as root
          MakeTree(T1,udv,+,vdu);
          Comment: put together to make the product rule
         }
Divide:
         {
          Assign tree u to T2^.LeftPtr;
```

```
                Assign tree v to T2^.RightPtr;
                Differentiate(du,u);
                Differentiate(dv,v);
                MakeTree(vdu,v,*,du);
                MakeTree(udv,u,*,dv);
                MakeTree(Numerator,vdu,-,udv);
                MakeTree(Denominator,v,^,2);
                MakeTree(T1,Numerator,/,Denominator);
                Comment: put together into quotient rule
                Comment: D(u/v)=(vdu-udv)/v^2;
              }
Power:
              {
                Assign tree u to T2^.LeftPtr;
                Assign tree v to T2^.RightPtr;
                Differentiate(du,u);
                MakeTree(PowerMinus1,v,-,1);
                MakeTree(vTimesPowerMinus1,v,*,PowerMinus1);
                MakeTree(T1,vTimesPowerMinus1,*,du);
                Comment: put together using chain rule as :
                Comment: D(u^v)=vu^(v-1)du
              }
}} Comment: end case and operator
else if root of T2 is a Constant then MakeTree(T1,nil,0,nil)
                Comment: D(c)=0;
else if root of T2 is a Real then then MakeTree(T1,nil,0,nil)
                Comment: D(Real)=0;
else if root of T2 is a Variable then MakeTree(T1,nil,1,nil)
                Comment: D(Variable)=1;
else if root of T2 is a function then
              {
                    Case T2^.WhichFunction of
                    FSIN:
                           {
                            Assign tree u to T2^.LeftPtr;
                            Comment: argument of sin function is
                            Comment: left subtree
                            Differentiate(du,u);
                            MakeTree(Cosu,u,cos,nil);
                            MakeTree(T1,Cosu,*,du);
                            Comment: put together using chain rule
                            Comment: D(sin(u))=cos(u)du
                           }
                 } Comment: end case
                 } Comment: end of T2 is a function
         }
}
```

The above pseudo-code is a sample of the code for the differentiation procedure. Here, only differentiation of one function (SIN) is shown and the code which caters for situations like 4x (a product of a constant and a variable) or x/4 have been left out. However, from this simple abstraction, the reader can see how the symbolic differentiation is performed.

Some examples of DRIFT output for particular student inputs are given below:

Your input was

$$x^4$$

<u>Differentiate</u> x^4

Use $\frac{\mathrm{d}}{\mathrm{dx}}(u^n) = nu^{n-1}$

I.e $\frac{\mathrm{d}}{\mathrm{dx}}\left(x^4\right) = 4 \cdot (x)^{4-1}$

So the derivative of x^4 is

$$\underline{\underline{\underline{4 \cdot x^3}}}$$

Help Level was 3

Press Q to QUIT the viewer

Your input was

$$\cos\left(4 \cdot x^3\right)$$

Differentiate $\cos(4 \cdot x^3)$

Use the CHAIN RULE $\frac{\mathrm{d}}{\mathrm{dx}}(\cos(u_1)) = -\sin(u_1)\frac{\mathrm{d}}{\mathrm{dx}}(u_1)$

where $u_1 = 4 \cdot x^3$

Differentiate $4 \cdot x^3$

Since 4 is a constant, use $\frac{\mathrm{d}}{\mathrm{dx}}(cf) = c\frac{\mathrm{d}}{\mathrm{dx}}(f)$

I.e $\frac{\mathrm{d}}{\mathrm{dx}}(4 \cdot x^3) = 4\frac{\mathrm{d}}{\mathrm{dx}}(x^3)$

Now, $\frac{\mathrm{d}}{\mathrm{dx}}(x^3) = 3 \cdot x^{3-1} \qquad = 3 \cdot x^2$

Therefore $\frac{\mathrm{d}}{\mathrm{dx}}(4 \cdot x^3) = 4\frac{\mathrm{d}}{\mathrm{dx}}(x^3) = 4 \cdot 3 \cdot x^2$

So $\frac{\mathrm{d}}{\mathrm{dx}}(u_1) = -(4 \cdot 3 \cdot x^2 \cdot \sin(4 \cdot x^3))$

So the derivative of $\cos(4 \cdot x^3)$ **is**

$$\underline{\underline{-\left(4 \cdot 3 \cdot x^2 \cdot \sin\left(4 \cdot x^3\right)\right)}}$$

Help Level was 3

Note that a "dot" notation was used for display rather than multiplication being implied. This was requested by the students and also enabled multiplication of real numbers to be more clear. For example, without some multiplication symbol, 4×5 would appear as 45.

5.3.4 Help Levels

Before the experiment took place, students reported that some of the help that DRIFT gave was too "rigorous". That is, DRIFT gave a detailed description of all the elements of the differentiation - including the more simple parts within a complicated problem. For example, if a student was differentiating $(\sin(x^2))^2$ using the chain rule then including help on differentiating x^2 was, for some students, not necessary.

The concept of a Help Level arose in order to help students who found solutions too detailed. Here, students could decrease the help level so that the differentiation of the inner expressions (in this case x^2) was assumed and therefore omitted.

5.4 Mathematical Input

Chapter 4 highlighted the difficulties encountered when students are required to input expressions into the computer. The algorithms described were used to create the first version of the Input Tool - a DOS based program. This version used a parsing routine which translated student input into LaTeX code. This code was then compiled and viewed using suitable rendering software.

The example below shows how a one-line input of `(x^2+1)/(x-1)` would be displayed to the student using LaTeX.

Your input was : `(x^2+1)/(x-1)`

which is represented as

$$\frac{x^2+1}{x-1}$$

A pseudo-code routine which shows the translation from one-line to LaTeX code is provided below:

```
Procedure LaTeX(var whichtree:Btree);
VAR
 NumberString:String;
{
 If whichtree<>nil then
  {
   CASE NodeStatus of the root of the tree of
    OperatorNode: CASE WhichOperator on root of WhichTree of
            divide:{
                Comment: a/b in LaTeX is \frac{a}{b}
            AppendLaTeXby('\frac{');
                LaTeX(Left SubTree of WhichTree);
                AppendLaTeXby('}{');
                LaTeX(Right SubTree of WhichTree);
                AppendLaTeXby('}');
               };
         Comment: LaTeX the left tree (numerator)
         Comment: and right tree (denominator)
         power: {
             Comment: check if base and power needs to be bracketed
             Comment: and use the syntax (a)^{b} if they do - otherwise
             Comment: simply miss out the brackets
             If (Left SubTree of WhichTree is a Function) or
                   (Left SubTree of WhichTree is an Operator ) then
                        {{need extra brackets}
                         AppendLaTeXby('\left(');
                         LaTeX(Left SubTree of WhichTree);
               AppendLaTeXby('\right)^{');
                 LaTeX(Right SubTree of WhichTree);
                 AppendLaTeXby('}');
                } else
                {
                 If NeedLeft(WhichTree) then
                    {
                     AppendLaTeXby('\left(');
                     LaTeX(Left SubTree of WhichTree);
                     AppendLaTeXby('\right)^{');
                    } else
                    {
                     LaTeX(Left SubTree of WhichTree);
                     AppendLaTeXby('^{');
                    };
         LaTeX(Right SubTree of WhichTree);
         AppendLaTeXby('}');


        };
      }; Comment: Power
minus: {
Comment: check if left/right trees needs to be bracketed
Comment: and use the syntax (a)-(b) if they do - otherwise
Comment: simply miss out the brackets

    LaTeX(Left SubTree of WhichTree);
    If NeedRight(WhichTree) then
    {
     AppendLaTeXby('-\left(');
     LaTeX(Right SubTree of WhichTree);
```

```
       AppendLaTeXby('\right)');
      } else
      {
       AppendLaTeXby('-');
       LaTeX(Right SubTree of WhichTree);
      };
    }; Comment: Minus
UnaryMinus:
    {
Comment: check if left tree needs to be bracketed
Comment: and use the syntax -(a) if it does - otherwise
Comment: simply miss out the brackets

      If NeedLeft(WhichTree) then
      {
       AppendLaTeXby('-\left(');
       LaTeX(Left SubTree of WhichTree);
       AppendLaTeXby('\right)');
      } else
      {
       AppendLaTeXby('-');
       LaTeX(Left SubTree of WhichTree);
      };
    }; {Unary Minus}
Mult: {
      If NeedLeft(WhichTree) then
      {
       AppendLaTeXby('\left(');
       LaTeX(Left SubTree of WhichTree);
       AppendLaTeXby('\right)');
      } else
```

```
              {
               LaTeX(Left SubTree of WhichTree);
               AppendLaTeXby(times);
              };
              If NeedRight(WhichTree) then
              {
               AppendLaTeXby('\left(');
               LaTeX(Right SubTree of WhichTree);
               AppendLaTeXby('\right)');
              } else
              {
               LaTeX(Right SubTree of WhichTree);
              };
             }; Comment: Mult
         Plus: {
              LaTeX(Left SubTree of WhichTree);
              AppendLaTeXby('+');
              LaTeX(Right SubTree of WhichTree);
             }; {plus}
        };
ConstantNode: {
         case Constant on the root of the Tree of
           ee:{
              LaTeX(Left SubTree of WhichTree);
              AppendLaTeXby('e');
              LaTeX(Right SubTree of WhichTree);
             };
          FPI:{
              LaTeX(Left SubTree of WhichTree);
              AppendLaTeXby('\pi ');
              LaTeX(Right SubTree of WhichTree);
```

```
            };
         };
        };
FunctionNode: {
        Case Function on the root of the Tree of
          FSQRT  :{
               AppendLaTeXby('\sqrt{');
               LaTeX(Left SubTree of WhichTree);
               LaTeX(Right SubTree of WhichTree);
               AppendLaTeXby('}');
               };  Comment: square rooting
          FSQR   :{
                If NeedLeft(WhichTree) then
                {
                AppendLaTeXby('\left(');
                LaTeX(Left SubTree of WhichTree);
                AppendLaTeXby('\right)^{2}');
                } else
                {
                LaTeX(Left SubTree of WhichTree);
                AppendLaTeXby('^{2}');
                };
               }; Comment: squaring
          FABS   :{
               AppendLaTeXby('\left|');
               LaTeX(Left SubTree of WhichTree);
               AppendLaTeXby('\right|');
               }; Comment: absolute
          FFACT  :{
                If NeedLeft(WhichTree) then
                {
```

```
            AppendLaTeXby('\left(');
            LaTeX(Left SubTree of WhichTree);
            AppendLaTeXby('\right)!');
            } else
            {
            LaTeX(Left SubTree of WhichTree);
            AppendLaTeXby('!');
            };
          }; Comment: Factorials
      FEXP   :{
            If NeedLeft(WhichTree) then
            {
            AppendLaTeXby('e^{\left(');
            LaTeX(Left SubTree of WhichTree);
            AppendLaTeXby('\right)}');
            } else
            {
            AppendLaTeXby('e^{');
            LaTeX(Left SubTree of WhichTree);
            AppendLaTeXby('}');
            };
          }; Comment: Exponentiation
      FTANH,FTAN,FSINH,FSIN,FSECH,FSEC,
      FLN,FLOG,FCOTH,FCOT,FCOSH,FCOSECH,
      FCOSEC,FCOS,FARCTANH,FARCTAN,FARCSINH,
      FARCSIN,FARCCOSH,FARCCOS:
          {
          AppendLaTeXby(CorrespondingTexCommand(
          FunctionType2String(Function on root of
           WhichTree))+'\left(');
          LaTeX(Left SubTree);
```

```
                    AppendLaTeXby('\right)');
                    };
             };
            };
    IdentifierNode:{
             LaTeX(Left SubTree of the Tree);
             AppendLaTeXby(IdentifierType2String(
              Identifier on root of WhichTree));
             LaTeX(Right SubTree of the Tree);
             };
    RealNode:   {
             If abs(frac(WhichTree^.item.WhichReal))<0.001 then
               Comment:  treat it as an integer
             Str(WhichTree^.item.WhichReal:0:0,NumberString) else
               Comment: treat it as a real to 5 decimal places
               Comment: (5 decimal places is usually accurate enough
               Comment: but the accuracy could be passed to the
               Comment: input tool from a question)
             str(WhichTree^.item.WhichReal:0:5,NumberString);
             LaTeX(Left SubTree of WhichTree);
             AppendLaTeXby(NumberString);
             LaTeX(Right SubTree of WhichTree);
            };
   };
  };
};
```

The problems with this method of input translation were two-fold. Firstly, it required the use of the LaTeX software. Although these programs were available on the public domain and were consequently free, they could not be bundled as part of other softwares, such as DRIFT. The LaTeX software could also be seen working in the background - something that may confuse the students. Secondly, and most importantly, this method used a lot of computing power and was therefore slow. The whole process of translation and display took around 12 seconds[2]. These concerns of speed and student useability prompted the next version of the software - the DOS version 2 of the Input Tool (see chapter 4).

The three different input methods which the students used were compared. The results, which are presented in table 5.9 , are quite striking begin to show the importance of communication between computer and student.

Input Method	Simple one-line	Using LaTeX Interpreter	Input Tool
Number of Input Errors (%)	17	13	1

Table 5.9: Errors Using Different Input Methods

In the table, an Input Error is a situation where an answer, be it correct or incorrect, is input wrongly in terms of either syntax or structure. For example, inputting `1/x-1^2` instead of `1/(x-1)^2` is deemed an input error.

This shows that, on average, a student would, without the use of the Input Tool, make a mistake in input 17 times out of 100. On these occasions, therefore, it is possible that a student would be marked as wrong when they should have been marked right.

Many examples during the experiment confirm the success of the input tool. The following examples reflect this and are shown in Table 5.10.

Intended Input	Key Sequence	Input Tool Used?
$x \wedge (1/2) - cos(x)$	$x(1/2) \Leftarrow\Leftarrow\Leftarrow\Leftarrow\Leftarrow \wedge(1/2) - cos(x)$	Yes
$x \wedge (1/2) - cos(x)$	$x(1/2) - cos(x)$	No
$cos(x) - 4sin(x) - x \wedge (-3/4)$	$cos(x) - 4sin(x) - x(3/4)$	No
$-3x \wedge (-5/2) + 1/4sin(x)$	$-3x(-5/2) + 1/4sin(x)$	No
$2x \wedge (2/3) - 1/4cos(x)$	$2x \wedge 2/3 \Leftarrow\Leftarrow\Leftarrow (2/3) - 1/4cos(x)$	Yes
$1/2x \wedge (1/2) + 2sin(x)$	$1/2x \wedge 1/2 + 2sinx$	No

Note: The $\Leftarrow$ signifies the pressing of the DELETE key

Table 5.10: Examples of the Effectiveness of the Input Tool whilst using DRIFT

The examples in Table 5.10 are by no means isolated cases. Almost all the students who did not have access to the Input Tool made similar mistakes as shown

[2]measured on 386 SX 16 MHz machines

here. Mistakes like these were almost completely eradicated for those students who used the Input Tool. To show the importance of efficient communication with the computer, we look at the following sequence of events which took place in the experiment:

1. Student inputs $sqrt(x) - cos(x)$ (from a question sheet) and wishes to differentiate it
2. Student answers with $1/2x(-1/2) + sin(x)$
3. Student is marked with the message "WRONG ANSWER"
4. Student looked at the solution on screen from 10:17:13 until 10:18:48
5. Student changes input to $x(1/2) - cos(x)$
6. Student answers with $1/(2x \wedge (1/2)) + sin(x)$
7. Student is marked with the message "WRONG ANSWER"
8. Student attempts to answer again with $1/2x \wedge (-1/2) + sin(x)$
9. Student is marked with the message "WRONG ANSWER"
10. Student attempts to answer again with $1/(2sqrt(x)) + sin(x)$
11. Student is marked with the message "WRONG ANSWER"
12. Student attempts to answer again with $1/2x \wedge (-1/2) + sin(x)$
13. Student is marked with the message "WRONG ANSWER"
14. Student attempts to answer again with $1/2(x \wedge (-1/2)) + sin(x)$
15. Student is marked with the message "WRONG ANSWER"
16. Student attempts to answer again with 9
17. Student is marked with the message "WRONG ANSWER" (The student has obviously given up)

18. Student decided NOT to re-attempt answer nor was the solution viewed.

Here, this student did not have access to the Input Tool. The question was firstly input correctly but the answer had a $\wedge$ missing and therefore was marked as being WRONG. The student then looked at the solution which explains the transformation from *sqrt* to $\wedge(1/2)$. This prompted the student to reenter the expression using $\wedge(1/2)$ notation. However, the $\wedge$ was again missing and so the answer was marked incorrectly. The student then tried various forms for the correct answer, all of which would have been valid if the initial input was correct.

The frustration of this student was shown in the questionnaire at the end of the session when he wrote:

> **the computer kept on marking my answer wrong but I couldn't understand why.**

5.5 Marking answers

Since it was required that students were given feedback on answers, there must be provision for the software to evaluate and mark the students. The uses of CALM Eval and CALM Compare (see Chapter 3) are limited to scenarios where the programmer / teacher has compile-time knowledge of the answers to problems. From this knowledge a suitable range, tolerance, failure rate and variable list can be easily specified – therefore limiting comparison of two answers to a "safe" range. The problem with evaluating and comparing answers in DRIFT is that they are only known at run-time and therefore the range cannot be pre-set. The problem is, therefore, how can two answers be compared if their "safe" ranges are unknown?

Possible solutions to this question are given below:

5.5.0.1 String Comparison

Chapter 2 highlighted the obvious reason why straightforward string comparison is, by itself, unusable. For example, the two expressions $x^2 - 3x + 2$ and $(x - 1)(x - 2)$ are mathematically equivalent and could therefore both be a correct answer to a problem. String comparison would deem these expressions to be different - an incorrect judgement.

5.5.0.2 Algebraic Comparison

An algebraic comparison of two mathematical expressions is a process where these expressions are re-written in some canonical form and then compared.

A great deal of work has been done on algebraic comparison of two expressions. Matz [24] states that

> **... even though mathematicians knowledge has an established notation in terms of axiomatic-deductive frameworks, these formalisms alone do not make a working problem-solver nor do they capture the knowledge of mathematics underlying human competence. '**

In this paper, Matz quotes Minsky [27] who says:

> **Minds are complex, intricate systems that evolve through elaborate developmental processes. To describe one, even at a single moment of that history, must be very difficult. ... Only a good theory of the principles of the mind's development can yield a manageable theory of how it finally comes to work.**

These two quotes encapsulate the difficulties of algebraic expressions and show that, even with the large amount of research done in this area, there is no reliable way of comparing student answers with a true answer algebraically. They also epitomise the unpredictable nature of students - particularly in the way that they give answers to questions.

Although algebraic comparison of some simple expressions is possible , certain examples give unpredictable results - see [38].

Since this experiment Waterloo Maple[3] have released software which gives programmers (and therefore teachers) access to very sophisticated computer algebra routines. However, these routines have yet to be suitably tested in an educational

[3]Waterloo Maple is the company responsible for the Maple computer algebra system

environment. These routines are now a centre of a great deal of work within the United Kingdom Mathematics Courseware Consortium. Within this project, work on answer judging and answer comparison is being tested.

5.5.0.3 Numerical Comparison

Notwithstanding the above, it has been shown that, in general, symbolic and algebraic comparisons of expressions do not yield a reliable way of comparing expressions. It has also been pointed out that there is no pre-knowledge of a safe range over which to compare two expressions. A way forward, perhaps, is an attempt to find a safe range over which both true and student answers can be compared.

A numerical method can be used to compare most expressions and therefore a detailed description of how this was achieved will be given below. Firstly, however, the following simple but interesting scenario will show that numerical comparison is not totally infallible:

Say a student types the expression $x\sqrt{a-b}\sqrt{b-a}$ (where a and b are general constants) and wishes to differentiate it. This yields, because of the square roots, conditions that $(a-b) \geq 0$ and $(b-a) \geq 0$. I.e. $a \geq b$ and $b \geq a$. This can only be true if $a = b = 0$ and therefore the original expression is equivalent to 0. If the numerical comparison routine employed found these conditions on a and b [4] then a derivative of 0 would be marked as correct. Moreover, if a student answers with $\sqrt{a-b}\sqrt{b-a}$ then, by the same argument, it would be marked as correct. This could be considered as both an advantage and a disadvantage of a numerical comparison algorithm. It is quite simple to build in the capability to point out these situations to the student and hence provide appropriate feedback. However, the situation where non-real valued functions are input should be catered for - perhaps by telling the students that an input is not defined on the real line.

Sometimes, a straightforward numerical comparison is not possible. For example, the expression $\sqrt{a-2} + \arccos(a)$ has two contradicting conditions on the value of a. The $\sqrt{a-2}$ imposes that $a \geq 2$ and $\arccos(a)$ means that $a \leq 1$. It is impossible,

[4]As will be seen later in this chapter, this depends heavily on the method on which the values of the identifiers are chosen. In particular, because of the method in which DRIFT compares answers, it will depend on the value of dx

therefore, for a numerical routine to choose a suitable value for a.

The simple example above shows how difficult it is to get a 100% reliable comparison algorithm. The following "solution", which can compare the majority of student answers, uses both string and numerical comparisons along with some simple algebraic manipulations.

The work done on expression trees can be used to provide greater meaning as to the construction of expressions. From such an expression tree, conditions and restrictions can be made on identifiers (constants and variables) so that comparison is made safe. In particular, one must be wary of the divide operator, square roots and trigonometric functions. For example, given an expression $\frac{1}{a}$, a condition on the identifier a is that its value is sufficiently far from 0 to avoid "blow-up". Also, in order to evaluate $\sqrt{a}$, one must insure that the value of a is greater than or equal to 0.

Two condition lists (an ordered list of conditions) can therefore be created - one each from the trees of the true and student answers. Call these condition lists CT and CS. This compare algorithm also needs to know a list of the identifiers that appear in both the student answer and true answer – called VL. The way in which these lists are constructed is described in this section.

The following flow chart (figure 5.4) describes the algorithm that DRIFT uses to check student answers. Note that within each node ($\diamond$, $\square$ or $\subset\supset$) on Figure 5.4 has an associated piece of text; these are shown shown in Table 5.11.

The flow starts at the oval root ($\subset\supset$ 1) where the binary trees for the student and true answers are acquired. If there are extra identifiers[5] ($\diamond$ 3) or not enough identifiers ($\diamond$ 2) (but not including the constants π or e) then the student answer is marked wrong ($\square$ 24) else they are checked for string equivalence ($\diamond$ 4). If they are symbolically equivalent then the answer is marked right ($\square$ 22) else the answers are simplified ($\square$ 5) (see 5.5.0.4) and checked again for string equality ($\diamond$ 6). Again, if they are now equivalent then the answer is marked right ($\square$ 22) otherwise the flow advances to ($\square$ 10). Here, a lists of conditions, called CS and CT are created from the student and true answer trees respectively. Then ($\square$ 9) creates a list of

[5]An identifier is either a symbolic constant or variable (i.e. "A" through to "Z" and "a" through to "z")

$\diamond$, $\square$ or $\subset\supset$ Number	Associated Text
$\subset\supset$ 1	Get True and Student Answer Trees
$\diamond$ 2	Are there enough identifiers in the Student Answer?
$\diamond$ 3	Are there extra iden. in Student Answer (Compared to True Answer)?
$\diamond$ 4	Are the Student and True Answers symbolically equal?
$\square$ 5	Use the simplification routine on both True and Student Answers
$\diamond$ 6	Are the Student and True Answers symbolically equal?
$\square$ 7	Assign the values of the variables contained in the variable list VL
$\square$ 8	Set the Boolean variable `OkAtPoints` to `FALSE`
$\square$ 9	Create VL – a list of variables appearing in Student and True Answers
$\square$ 10	Create condition lists for Student and True Answers (CS and CT)
$\diamond$ 11	Are the condition lists CS <u>and</u> CT empty?
$\square$ 12	Set the Boolean variable `OkAtPoints` to be `TRUE`
$\diamond$ 13	Is the conditioin list CS empty?
$\diamond$ 14	Do the variables in VL contravene CT?
$\square$ 15	Set the Boolean variable `OkAtPoints` to `FALSE`
$\diamond$ 16	Have all possible values on VL been used?
$\diamond$ 17	Is the condition list CT empty?
$\diamond$ 18	Do the variables on VL contravene the conditions on CS?
$\diamond$ 19	Do the variables on VL contravene CS or CT?
$\square$ 20	Set the Boolean variable `OkAtPoints` to be `TRUE`
$\diamond$ 21	Are Student and True Ans. numerically equal using the values in VL?
$\square$ 22	The Student Answer is equal to the True Answer
$\square$ 23	The Answers are numerically incomparable (with this algorithm)
$\square$ 24	The Student Answer is NOT equal to the True Answer

Table 5.11: Text for nodes on Figure 5.4

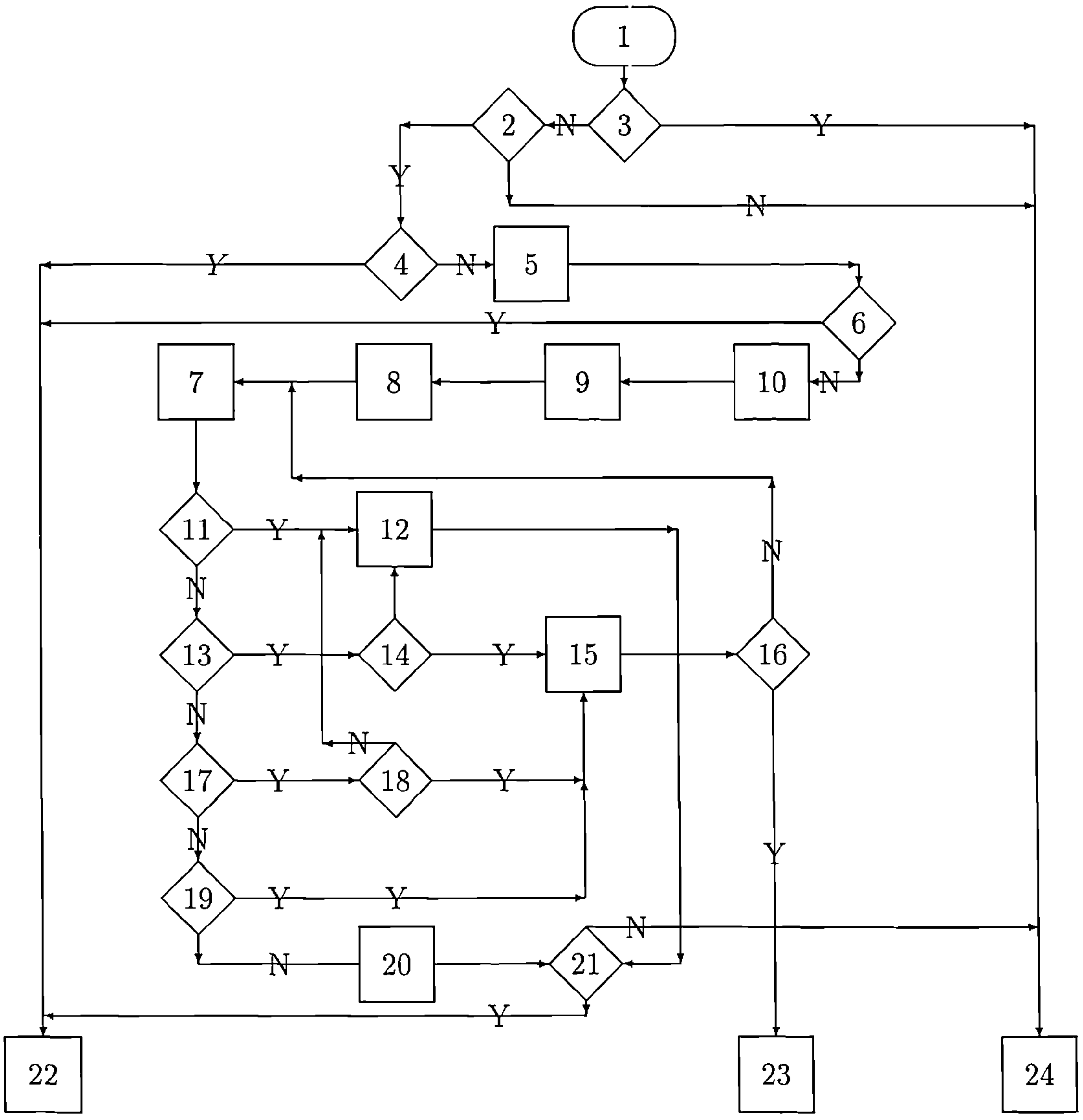


Figure 5.4: Flow Chart showing the Numerical Comparison Algorithm

identifiers, VL, which appear in both the student and true answer trees. A Boolean variable `OkAtPoints` is initialised to `TRUE`. This variable is used to flag whether or not values for the identifiers have been found which do not contravene the conditions contained in both CS and CT. The flow advances to (□ 7) where the values of the identifiers contained in VL are set. Note that the values of these identifiers are advanced every time the flow reaches (□ 7). How the identifiers are advanced will be explained later.

Now check to see if both lists CS and CT are empty (◇ 11). If they are empty then there are no problems in numerical comparison, so set `OkAtPoints` to be `TRUE` (□ 12), numerically compare the two answers (◇ 21) and either mark the answer correct (□ 22) or incorrect (□ 23) accordingly. If either CS or CT are non-empty then check if either CS (◇ 13) or CT (◇ 17) are empty. If CS is empty then check if the identifiers in VL contravene the conditions in CT (◇ 14). If they do then set `OkAtPoints` to `FALSE` (□ 15) and check if all the possibilities of values of the identifiers in VL have been used. If they have then the answer cannot be numerically compared (□ 23) otherwise, choose other values for the identifiers in VL (□ 7). Similarly, if CT is empty then check if the identifiers in VL contravene the conditions in CS (◇ 18). If they do then set `OkAtPoints` to `FALSE` (□ 15) and take the same path as before either to (□ 23) or (□ 7).

If neither CS or CT are non-empty then check if the identifiers in VL contravene either CS or CT (□ 10). If they do then set `OkAtPoints` to `FALSE` (□ 15) and check if all the possible values of the identifiers on VL have been used (◇ 16). If they have then return to (□ 7) and re-assign values otherwise the answers cannot be compared numerically (□ 23). If the identifier values do not contravene the condition lists CS and CT then set `OkAtPoints` to be `TRUE` and numerically evaluate each expression to see if they are equivalent. If they are then mark the student answer correct (□ 22) otherwise mark it wrong (□ 24).

The most complex parts of this algorithm are those contained at stages (□ 7) and (◇ 16). Methods of deciding what values the identifiers should take and whether all possible combinations of values for the variables have been exhausted are now considered.

The method which DRIFT currently adopts to choose the values of the identifiers on the list VL is to iterate from `-MaxValue` to `MaxValue` in steps of `dx` where `MaxValue` and `dx` are real values. Here, setting `MaxValue` sufficiently large and `dx` small will create a large number of combinations for the values of the identifiers. This method obviously provides an easy way of knowing if and when all the values of the identifiers have been exhausted. There is a problem using this limited range when an expression is not defined within $-$`MaxValue` $\cdots$ `MaxValue`. For example, it is impossible, if `MaxValue`$= 100$, to mark the expression $\sqrt{a - 101}$.

One solution would be to examine all the conditions imposed on the identifiers and choose the value of `MaxValue` to be sufficiently high.

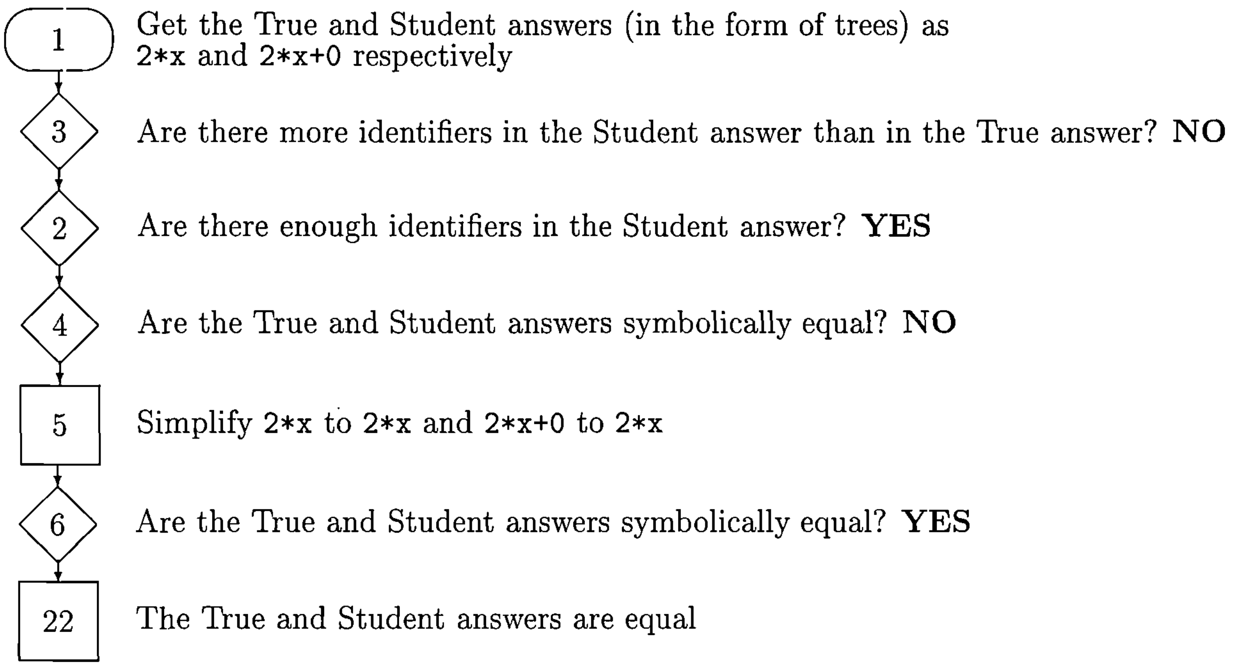
The value of `dx` can also be critical in comparing some expressions. For example, going back to the expression $\sqrt{a-b}\sqrt{b-a}$, it was noted that the conditions gave $a = b = 0$. Here, if `dx` is large (perhaps in the order of 1) then iterating through possible values of a and b may miss out $a = b = 0$. For example, if `MaxValue` $=$ 100 and `dx` $= 0.3$ then the values of a and b, because of the iteration, would be $-100, -99.7, \cdots, -0.7, -0.4, -0.1, 0.2, \cdots, 100$.

Two examples showing how the algorithm proceeds follows. The first example (see Figure 5.5) shows how simple algebraic simplification can yield a faster comparison while the second (see Figure 5.6 shows two algebraically different expressions can be compared.

More sophisticated methods of finding values of the identifiers could be sought, perhaps by simultaneously solving the conditions imposed. Another possible method, which is more efficient and yet easy to implement, is described in figure 5.7 below whose labels are given in table 5.12 :

◇, □ or ⊂⊃ Number	Associated Text
⊂⊃ 1	Make a list called CList - a combination of CS and CT
□ 2	Choose values for the variable list VL
□ 3	Remove conditions from CList that aren't contravened by VL values
□ 4	Create new VL for revised cond. in CList (previous values remain)
◇ 5	Have all possible values for iden. in VL been considered?
◇ 6	Is CList empty?
□ 7	The Answers are numerically incomparable (with this algorithm)
□ 8	Set the Boolean variable `OkAtPoints` to `TRUE`

Table 5.12: An Improvement in Identifier Selection



Get the True and Student answers (in the form of trees) as 2*x and 2*x+0 respectively

Are there more identifiers in the Student answer than in the True answer? **NO**

Are there enough identifiers in the Student answer? **YES**

Are the True and Student answers symbolically equal? **NO**

Simplify 2*x to 2*x and 2*x+0 to 2*x

Are the True and Student answers symbolically equal? **YES**

The True and Student answers are equal

Figure 5.5: Example 1 Describing the Numerical Comparison Routing in Figure 5.4

1 Get the True and Student answers (in the form of trees) as `x+1` and `(x^2-1)/(x-1)` respectively

3 Are there more identifiers in the Student answer than in the True answer? **NO**

2 Are there enough identifiers in the Student answer? **YES**

4 Are the True and Student answers symbolically equal? **NO**

5 No simplification done. The simplification routine cannot factorise.

6 Are the True and Student answers symbolically equal? **NO**

10 Create CS as $x - 1 \neq 0$ and CT as *Empty* (no conditions on CT)

9 Create VL just containing the one variable x

8 Set OkAtPoints to FALSE. (Assume cannot compare using values in VL)

7 Assign values of the variables on VL to $x = -maxvalue = -100$, say.

11 Are CT <u>AND</u> CS empty? **NO**

13 Is CS empty? **NO**

17 Is CT empty? **YES**

18 Do the values on VL contravene CS? **NO** (since $-100 - 1 \neq 0$)

12 Set OkAtPoints to be TRUE

21 Are the True and Student answers numerically equal using $x = -100$? **YES**

22 The True and Student answers are equal

Figure 5.6: Example 2 Describing the Numerical Comparison Routing in Figure 5.4

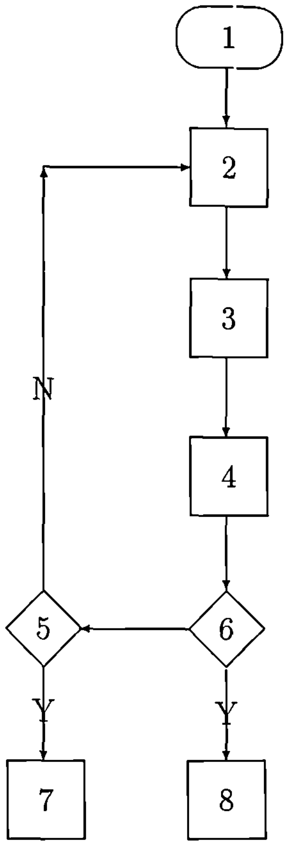


Figure 5.7: An Improved Method of Finding Identifier Values

This algorithm replaces parts of the earlier one by first merging CS and CT into one condition list called CList (⊂⊃ 1). Again, values for the identifiers in VL are now chosen (□ 2). This time, however, they are chosen so that the values of certain identifiers (those which don't contravene CList) don't change. Then, the algorithm removes all the conditions from CList that aren't contravened by the values of the identifiers in VL (□ 3) and then chooses a new VL from the conditions left in CList (□ 4). If CList is empty of conditions (◇ 6) then comparison is now valid using the values of the variables in VL, otherwise, check if all possible values of identifiers in VL have been considered (◇ 5). If they have then the student and true answers cannot be marked numerically (□ 7) otherwise choose other values (□ 2).

This algorithm is more efficient since once "safe" values for certain identifiers have been found, they are taken out so the conditions which they satisfy are not re-considered later. The only problem with this method is that the very identifiers taken out at an early stage may be required to be reconsidered if stage (7) is reached. The following simple example will illustrate this point:

Say one of the answers to be compared (either true or student answers) is $\frac{1}{a} + \frac{1}{1-a}$ then the conditions will be $a \neq 0$ and $1 - a \neq 0$. The algorithm will take the first condition away from CList (as long as $a \neq 0$ and fix the value of a. However, if this value of a was fixed as 1 then this will contravene the later condition $1 - a \neq 0$.

One way to avoid this is to take conditions away from CList only if the variables which they contain do not appear in the remaining conditions. For example, if an answer was $\frac{1}{b} + \frac{1}{1-a}$ then taking out the condition $b \neq 0$ from CList and fixing the value of b would have no effect on the remaining condition $1 - a \neq 0$.

The problem of how to choose values for the identifiers in VL (2) still remains. However, with the above improvement, simple iteration through the permutations of values for identifiers will be more efficient. DRIFT currently chooses the values in VL by looking at the permutations of the possible values for the identifiers. I.e. it uses the permutations of the variables each with values from -MaxValue to MaxValue.

5.5.0.4 Expression Simplification

The numerical routine described above makes use of a basic simplification subroutine. This enables simple expressions to be simplified and therefore possibly easier to compare. For example, the expression $x + 0$ is simplified to x, $x + x$ is simplified to $2 * x$ and $\frac{3}{4} \times \frac{5}{6}$ is simplified to $\frac{5}{8}$.

5.5.0.5 The Condition Lists

The conditions used in section 5.5.0.3 are now described. So far only the logical operators $\geq$, $\neq$ and $=$ have been used to specify a condition on an identifier or expression. The logical operators used for marking answers all have an associated binary tree (representing the expression) and a numerical value. The operators used are given in the following Turbo Pascal TYPE declaration:

```
TYPE
  CondType = (LessEqual,Less,GreaterEqual,Greater, Equal,
                        NEqual, ProductOf, NProductOf);
 (*  Comment:
      the N in front of Equal and ProductOf indicate
      NOT  (i.e NotEqual and NotProductOf)
  *)
```

Examples of possible conditions are given in table 5.13:

Tree to Compare	Condition Tree	Log. Operator	Value
$\frac{1}{a}$	a	NEqual	0
$\frac{1}{\sqrt{a}}$	$\sqrt{a}$	NEqual	0
	a	GreaterEqual	0
$\tan(x)$	$x - \frac{\pi}{2}$	NProductOf	π
$\arccos(x)$	x	LessEqual	1
	x	GreaterEqual	-1

Table 5.13: Examples of Conditions on Expressions and Identifiers

To conclude this chapter, it is pointed out that there is access to the DRIFT package on the diskette provided with this thesis. This is included to give demonstration of both the Input Tool and the marking procedure which are described in the text. It is not provided as a demonstration of DRIFT's facilities or functionality.

In order to accommodate the Input Tool, a **demonstration** version of DRIFT is included which runs in Microsoft Windows on IBM PC compatible computers[6]. To run DRIFT (for Windows), simply double click on the DRIFTWIN.EXE file in the Windows' File Manager.

[6]this has superceded the DOS version of DRIFT

Chapter 6

Formally Assessing Mathematical Ability by Computer

6.1 Introduction

This chapter describes an educational experiment which took place at the Heriot-Watt University in Edinburgh during the Autumn Term 1994. The experiment set out to perform 40 % of the assessment for a mathematics module taken by 82 first year engineering students. The experiment was designed so that the computer would set and mark exam questions while the students were sitting at a computer keyboard. The chapter first describes the method of question delivery and explains the process of preparation for such a technically difficult event. The important educational issues which arose during the experiment will be discussed – such as partial credit, input, examination security and software design. The chapter concludes with an evaluation of the success of the computer assessment and implications for future research.

A computer's ability to enhance the process of learning is well documented (for example, [34], [15], [3], [4], [7], [31] and [40].) Howevcr, it is important to show that a computer can have a broader role in education in that it can be used to reduce some of the chores of teaching.

In particular, the computer can be used to automatically mark hundreds of computer-set exam scripts at the end of a university term. Moreover, a computer's speed brings with it other advantages such as a more flexible approach to testing (i.e. an examination is available either more frequently or perhaps when a student is ready to be tested). A computer examination brings objectivity in that the student's name is irrelevant. An important feature of computer examinations is that they give *instant* results.

Naturally, there are drawbacks to the use of the computer for assessment. There are important questions which teachers must consider. How does the computer compensate in giving partial credit and how significant is students' familiarity with technology? Another issue, as discussed in chapters 3 and 4, is that there is a mismatch of notations between writing mathematics on the computer and on paper. This concern is magnified when one sets computer examinations which are not of the multiple choice format – as is the case here. All of these concerns are discussed throughout this chapter.

6.2 The Mathwise Testing Mechanism

To deliver mathematics exam questions over the computer successfully requires a robust, well structured, and *easy to use* software package. The method of question delivery used for the experiment will now be described.

The Mathwise [1] test mechanism was written within the Mathwise project to provide an assessment methodology to accompany modules written across the whole of the TLTP[2] project.

6.2.1 Banking Questions

It employs a method of question delivery which groups questions into banks. Banking questions in this way enables a higher flexibility in terms of which questions can be delivered to the students. For example, a particular mid-term examination may require the testing of one particular topic or a selection of topics. Therefore the banks of questions can be made to coincide with parts of the syllabus which are to

[1]Mathwise is one of the three mathematical consortia funded during the first phase of the Teaching and Learning Technology Programme (TLTP) underway in British Universities 1992-5

[2]TLTP = Teaching and Learning Technology Program

be tested.

For the purpose of this experiment, the questions were picked randomly from all banks, which covered the syllabus, to make up a whole computer examination "paper".

6.2.2 Browsing the Questions

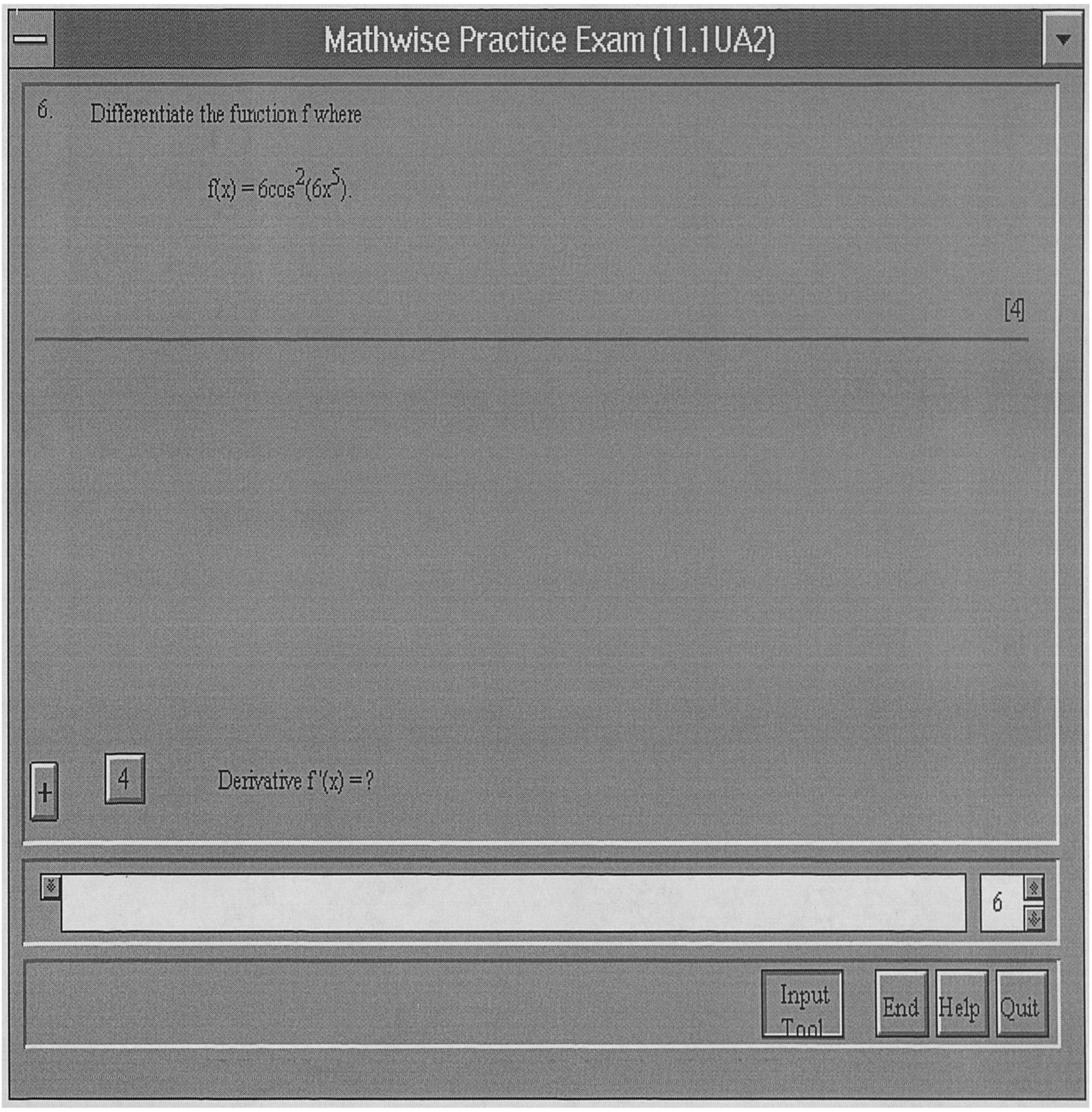
Traditional examination papers usually allow students to answer questions in any order. Therefore, a *browsing* facility was included in the testing mechanism which enabled the students to choose which questions could be answered first. Educationally this is very important since confidence during an exam is often lost or found whilst answering the first question. It was observed during the experiment that students employed the browsing mechanism often, using it to pass on the harder questions.

6.2.3 The Marking Scheme and Sub/Key-Parts

The marking scheme was a very simple one. The number of marks available for any one question is the number of parts it contains – one mark for each part.

The individual questions are designed to contain up to 12 parts with each part worth one mark. However, for the purposes of the experiment, questions contained either two or four parts. Any number of these parts are then deemed to be "key" or "sub", depending upon their importance within the question.

Figure 6.8 shows a question on differentiation within the test mechanism. The question asks to differentiate the function f where $f(x) = 6\cos^2(6x^5)$ and displays part 4 which is a key-part. This key-part simply asks for the derivative of f, assuming that student can apply the chain rule to both $6x^5$ and to the square of the cosine function. If a student cannot answer this part then they can press on the "+" button and reveal the sub-parts 1,2 and 3. Figure 6.9 shows these revealed sub-parts, giving a break down of the question. Note that if sub-parts are revealed then they must be answered, along with their key-part, to gain full marks. Each key-part is worth 1 plus the value of its subparts.

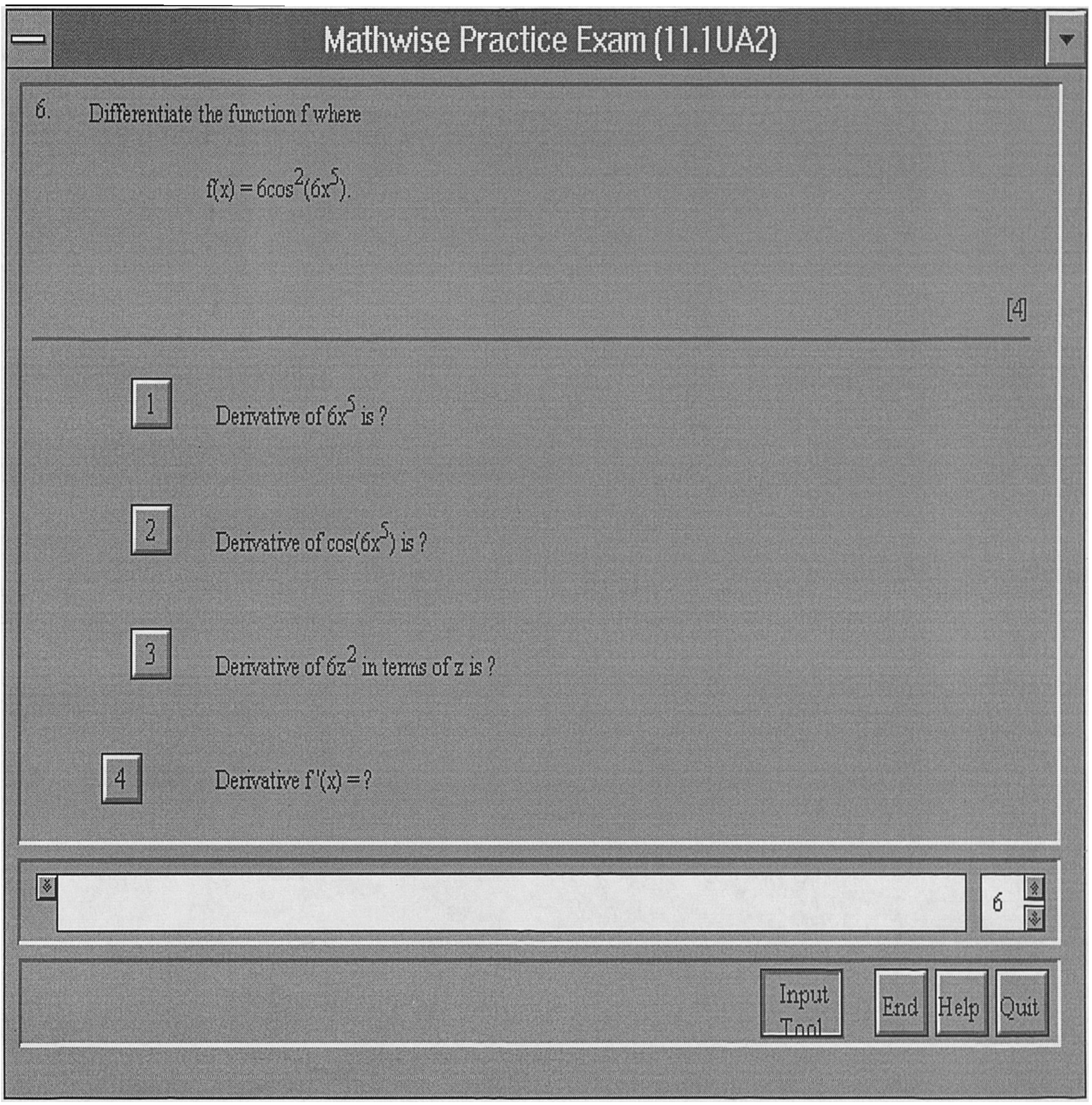


6. Differentiate the function f where

$$f(x) = 6\cos^2(6x^5).$$

Derivative f '(x) = ?

Figure 6.8: Figure showing Key-Parts


Mathwise Practice Exam (11.1UA2)

6. Differentiate the function f where

$f(x) = 6\cos^2(6x^5)$.

[4]

1 Derivative of $6x^5$ is ?

2 Derivative of $\cos(6x^5)$ is ?

3 Derivative of $6z^2$ in terms of z is ?

4 Derivative $f'(x) = ?$

Figure 6.9: Figure showing revealed Sub-Parts

6.2.4 Randoms and Test Re-Usability

Questions can contain random parameters. That is, constants, coefficients, powers etc. can be randomised in order to change the question for subsequent examination sittings. In the example above (in figure 6.8), the two 6s, the power 2 and the 5 were random integers from 2 through to 9. This concept is vital for such a testing mechanism for if the test is to be re-used in subsequent years, and therefore justify the effort in the creation of the test, then the questions must be different each time they are displayed or used.

Once a question is completed, by pressing the *End* button, it is taken out of the bank and the student returns to browse mode[3].

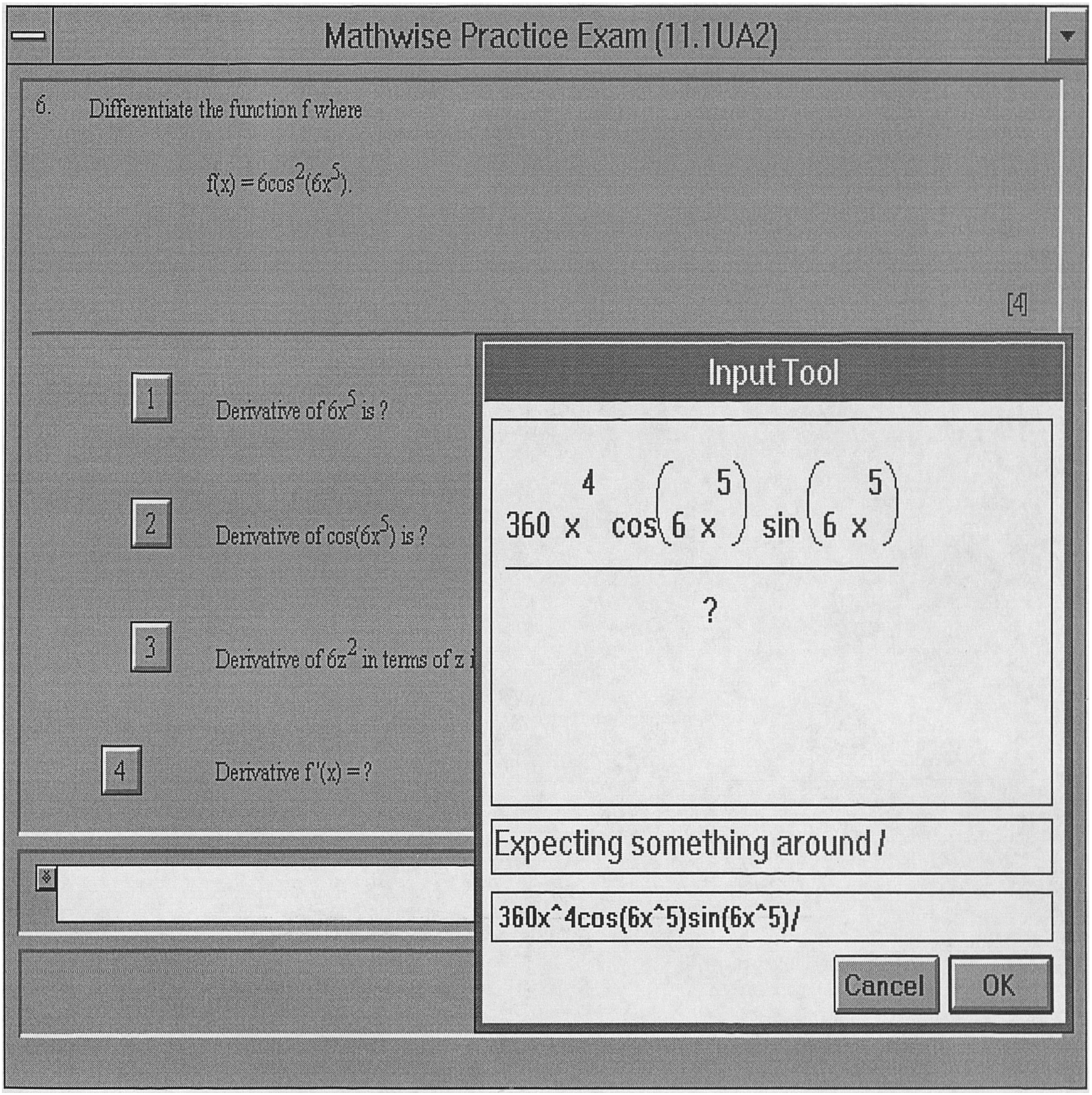
6.2.5 Levels of Feedback

The mechanism contains 4 levels of feedback. Level 4, *help level*, provides students with maximum feedback. Here, students are marked after each part with a tick or a cross or are allowed to reveal the answers (perhaps, therefore, helping them in subsequent parts). Level 3 provides slightly less help. Students are not allowed to reveal the answers to questions but still get the parts to the questions marked as they go. This mode is useful to monitor the moderate students. Level 2 provides marking at the end of the question only and there is no revealing of answers allowed. Here, the main use could be for revision for the final level, level 1. This level is termed *exam mode* and, as the name suggests, is useful for the use in formal assessment as students receive no marking throughout the test. Teachers can decide if students marks should be displayed at the end of the test. During the run-up to the experiment, the students practised using the Mathwise mechanism in feedback level 2 but for the actual experiment examination mode was used.

[3]problems associated with taking out the question from the test at this stage will be considered later

6.2.6 Methods of Input

The IBM-PC version of the testing software [4] includes a button for the "Input Tool". When this button is pressed in (on), input is via the Input Tool as described in chapters 3 and 4 otherwise, it is via a one-line input in the white rectangular box at the bottom of the screen. The screen layout, showing the position of the Input Tool, is shown in figure 6.10.


Mathwise Practice Exam (11.1UA2)

6. Differentiate the function f where

$f(x) = 6\cos^2(6x^5)$.

[4]

1 Derivative of $6x^5$ is ?

2 Derivative of $\cos(6x^5)$ is ?

3 Derivative of $6z^2$ in terms of z i

4 Derivative f'(x) = ?

Input Tool

$$\frac{360\,x^4 \cos\left(6\,x^5\right) \sin\left(6\,x^5\right)}{?}$$

Expecting something around /

```
360x^4cos(6x^5)sin(6x^5)/
```

Figure 6.10: The Mathwise Mechanism using the Input Tool

[4]The Mathwise test software is available on both Apple Macintosh and IBM-PC

6.2.7 Marking Answers

The Mathwise Test Mechanism uses the CALM compare software to mark the students' answers. However, because of the restrictions in CALM compare, which are identified in 3.3.2.4, extra facilities are provided to enable the setter to encourage the student to give the correct form of the desired answer.

6.2.7.1 MaxLength and MinLength

Sometimes, it will be required that the number of characters contained in a student answer needs to be restricted. The following example illustrates this point: "Factorise $x^2 - 1$". The software should not accept `x^2-1` as a correct answer; rather it should force the student to factorise the quadratic. A simple way to solve this would be to insist that the student answer contains at least 10 characters (MinLength=10). Although this does mean that students cannot type just `x^2-1` as an answer, `((((x^2-1))))` would be marked as correct.

A way around this would be the use of **Must-Have** and **Not-Allowed** strings.

6.2.7.2 Must-Have and Not-Allowed Strings

In the example above, the problem of what to allow as student answers can be solved by insisting that they must contain the "(" character AND must NOT contain either the "((" string or the "^" string. This type of answer restriction is very useful, particularly in questions which ask students to expand or simplify answers.

So as not to waste student input time, the Input Tool could be made aware of MaxLength, MinLength, Not-Allowed String and Must-Have String for a particular answer and advise the student about them as they type. This would be quite a simple and yet powerful extension to the Input Tool.

Take, for example, the question: "What is 0.5 as a fraction ?." Because CALM compare does not make a distinction between fractions and decimals, an answer of 0.5 or .5 would be marked as correct.

6.2.8 Recording Marks and Student Information

As the software is used, two files are generated and amended. The first, a .mrk file, contains the student's name (identification), the date, the title of the examination, the number of questions that were selected from each bank, the total number of parts and the percentage mark and the status at the end of the test. A sample .mrk is now given:

```
David, 10/07/95, 11.1UA1 EXAM, 1, 1/LU1, 1/LU2, 1/LU3, 1/LU4, 1/LU5,
 1/LU6, 1/LU7, 1/LU8, 24, 42, quit
```

The second file, with extension .tst, contains more detailed information about each question and the answers given by the student. The test file corresponding to the above .mrk file is shown below:

```
Results File
********************
Start Test Data
====================
TestStartDate    10/07/95
TestStartTime    13:57
TestTitle        11.1UA1 EXAM
SectionsTested   1/LU1, 1/LU2, 1/LU3, 1/LU4, 1/LU5, 1/LU6,
 1/LU7, 1/LU8,
NumQuestions     8
QuestionSet      10
18
29
38
43
57
64
80


Randoms
```

```
====================

(10)    -22,-33

(18)    5,-3,5,9

(29)    4,-4

(38)    3,2,5,8

(43)    4,6

(57)    4,1,2

(64)    9

(80)    5,8

--------------------

Question 43

============

SeqNo   5

StartDate       10/07/95

StartTime       13:57:59

ID      43

Parts   2

Keys    2

Rands

MaxScore        2

StudScore       2

TimeIn  477

Tries   2

FinDate 10/07/95

FinTime 14:06:00

----------------------------------------

PartNo  1       2

Tans    6       4-6

Vset

Range   1,2,5   1,2,5

Tolfr   0.001,0.0       0.001,0.0
```

```
MaxLen   100      100

MinLen   0        0

MaxScore          1        1

Studans 6         -2

Tries    1        1

Done     0        0

Emode    R        R




+++++++++++++++++++++++++++++++++++++++

********************

Quit Test at 14:10 on 10/07/95

********************End Test Data

====================

Date     10/07/95

Time     14:10

Time in Test

Score    42%
```

A description of all the elements in this file is shown in table 6.14.

6.3 Preparing for the Computer Examination

Two groups of Computer Science and Electrical Engineering students were chosen as an appropriate pilot class from within the whole of the first year. From these classes, totaling 82, five students agreed to assist with feedback as term and our preparations progressed.

The main student concerns were isolated through a series of meetings with the small group and from follow-up questionnaires to the whole class.

Key	Description
SelectionsTested	List of the number of questions selected from each bank
NumQuestions	Identifies the number of questions in the examination
QuestionSet	The set of questions which were picked from the banks
Randoms	Lists, for each question, the value of the randoms
SeqNo	The number of the question as it appears to the student
ID	Identification number of the question
Parts	The number of parts that the question contains
Keys	Indicates the numbers of the parts which were "key"
Rands	Find out what this is
MaxScore	The max. score for this question (= number of parts)
StudScore	The score achieved by the student on this question
TimeIn	The time taken for the question
Tries	Number of tries at all the parts on the question
PartNo	The number of the part being attempted
Tans	The "True" answer (correct answer) to a PartNo
Vset	The "variables" contained in the question (identifiers)
Range	The "safe" range and number of points for compare (see Chapter 2)
Tolfr	Tolerance and Failure Rate
MaxLen	Maximum allowable characters in answer
MinLen	Minimum allowable characters in answer

Table 6.14: Table to Explain the .mrk and .tst Files

6.3.1 Class Questionnaires

The questionnaires given out, along with the responses received are now described.

Questionnaire on December Computer Test

1 What advantages to do you see in using the computer for part of your assessment in mathematics?

2 What concerns or worries can you identify in using the computer for part of your assessment in mathematics?

3 Could we have improved the information about the test? Yes/No. If Yes can you suggest what would have been more helpful?

4 It may not be possible but if it is would you want to know your mark immediately at the end of the test? Yes/No

Figure 6.11, below, shows the responses to question 1 on the questionnaire. Here, 26 students from group 1 and 31 from group 2 returned questionnaires and listed possible advantages of taking assessment on the computer. Figure 6.12 alters the Group 1 response frequencies (multiplying by $\frac{31}{26}$) to allow for the different number of questionnaires returned from each group. This gives greater pictorial clarity of the likeness between the questionnaire responses to question 1 and is equivalent to using a relative frequency scale (note that this does not alter the correlation figure). The students identified 15 advantages which are labelled in table 6.15.

Figures 6.11 and 6.12 show that, overall, the main considered advantages (of using the computer for assessment in mathematics) of both groups were similar. The product-moment correlation coefficient (see Appendix B) between group 1 and group 2 responses was 0.743 which clarifies this.

Predicted possible advantage of computer assessment	Label
Made a nice change	1
Faster results	2
Improve Comp. literacy	3
Good prep. for written exam	4
Less writing	5
Allows easy resit	6
Handwriting isn't important	7
More at ease at computer	8
Questions vary for individuals	9
Working need not be shown neatly	10
Marking is more consistent (fairer)	11
More Time	12
Less work needed by teachers	13
Splitting questions into sub-parts	14
Final mark doesn't depend on just one exam	15
No advantages	16

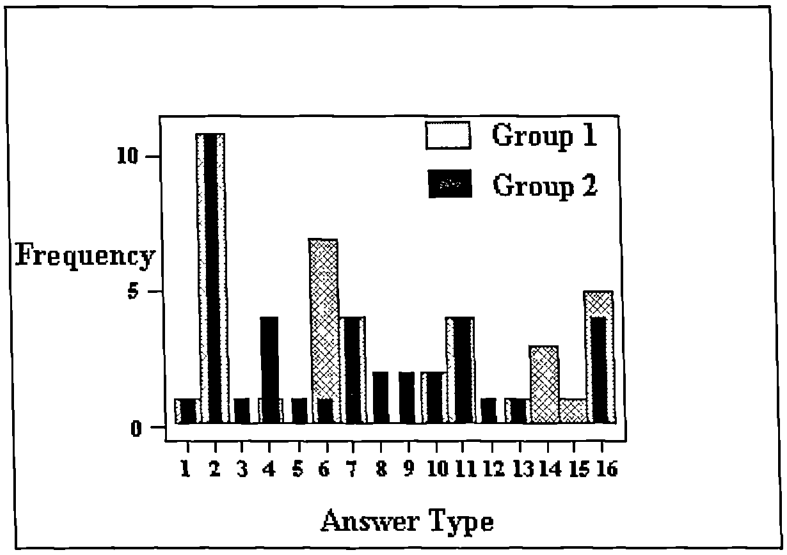


Figure 6.11: Responses to Questionnaire Question 1

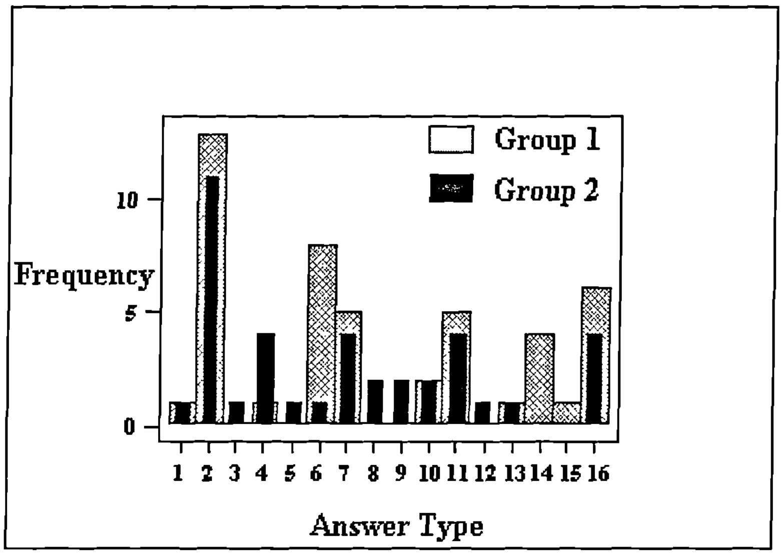


Figure 6.12: Group 1 Scaled Responses to Questionnaire Question 1 (adjusted)

A very interesting observation from the responses appears in this analysis. Group 2, who filled in their questionnaires on a later day from those in Group 1, reported advantage 6 (resits) far more heavily than students in group 1. This advantage would perhaps be less obvious unless a teacher pointed it out (which is, in fact, what happened). In other words, it appears that word of mouth (concerning the computer examination) played a part.

The questionnaire also showed general agreement concerning students' apprehensions of the computer exam. Again two figures are given, 6.13 and 6.14 (figure 6.14 shows compensations for differing size groups 1 and 2), which give the responses to question 2 on the questionnaire. Here, the Answer Types are described in table 6.16, below.

Predicted problems of computer assessment	Label
Computer doesn't give partial credit	1
Not sure about strategy for computer examination	2
Computer Exam is ergonomically uncomfortable	3
Correct answers will be marked as being incorrect	4
Mistakes in the questions	5
inexperience with the Mathwise Test Mechanism	6
Problems inputting answers	7
Can't re-enter a question once it is finished	8
Time constraints	9
One set of questions may be harder than another	10
Network collapse	11
None	12

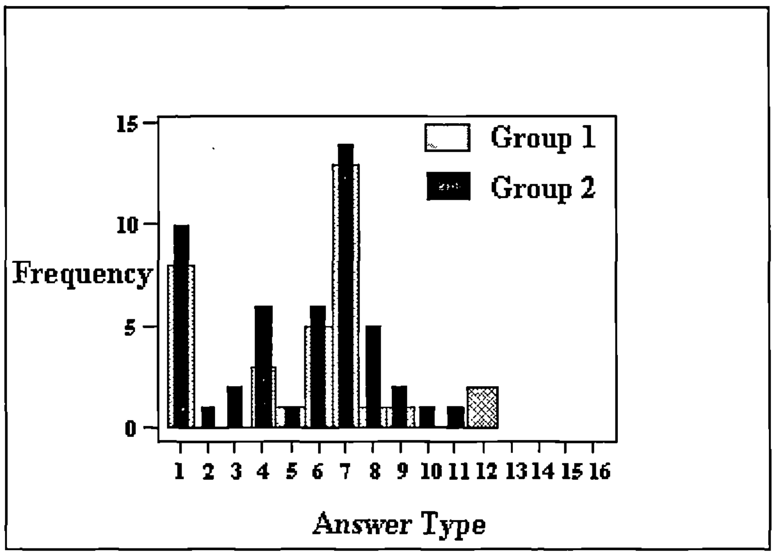


Figure 6.13: Responses to Questionnaire Question 2

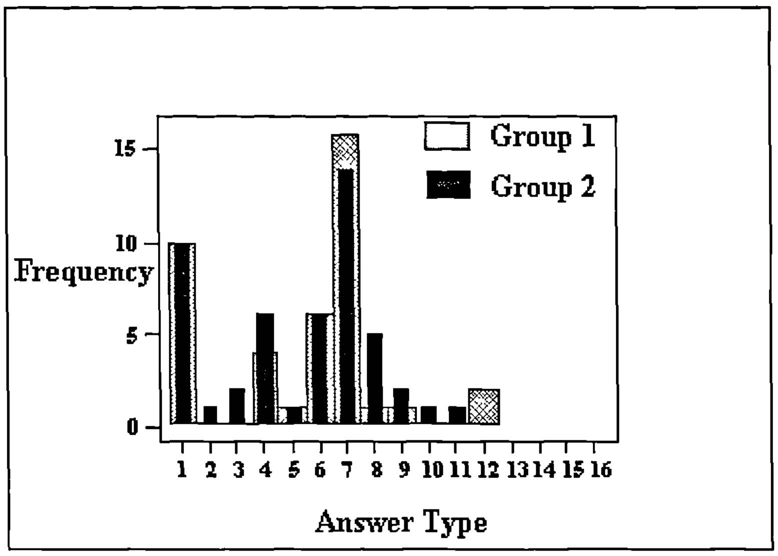


Figure 6.14: Responses to Questionnaire Question 2 (adjusted)

It is interesting to note again that the two groups had very similar worries concerning the computer examination. This time, the correlation coefficient between the groups 1 and 2 is 0.939.

The results of question 3 were very encouraging. Out of the 57 questionnaires returned, 47 students said there we couldn't have improved the information about the computer examination. Eight students from the remaining 10 students thought that more practice for the examination should have been available. However, it was made clear to all the students, both verbally and in the written hand-outs that the practice version of the Mathwise exam was available from 9am until 10pm, Monday to Friday for 2 weeks prior to the actual examination. There were also two practice sessions[5] in the students' tutorial times but it was possible that the 8 students who didn't have enough practice were those students missing from the tutorial.

The small discussion group was chosen to decide if students' marks were to be displayed at the end of the examination. This group unanimously chose **not** to get the marks immediately and so this strategy was adopted within the software. It is surprising, therefore, that $\frac{48}{57} = 84.2\%$ of students chose *Yes* to question 4 on the questionnaire. This seems to indicate that there is an initial attraction in knowing a result immediately but, in practice, students prefer time for reflection after the

[5]The practice tutorial sessions were designed to give the students experience with the Mathwise mechanism. The questions were not the same as those seen in the examination.

examination.

6.3.2 Concerns Identified by the Discussion Group

The main educational issues which arose from the small discussion group were found to be similar to those found in the questionnaires; they were:

1. that screens are visible to neighbouring students presenting a security problem;

2. how the computer marking could cope with partial credit; and

3. the mis-match between mathematical and computer notations.

For the computer examination, the Mathwise Test mechanism was loaded with 8 banks of questions. With random parameters also present in most of the questions, the number of different questions in any particular examination and / or bank increases enormously so that the possibility of any two students receiving the same question is very low. In this way, we sought to minimise the difficulty identified in 1 above – why look at a neighbour's screen when your questions are different? The questions for the examination were constructed in a way which strove to attain a similar standard across the eight banks, each bank containing ten questions.

Moreover, the design of the questions tried to minimise the problem of partial credit. Questions were staged in either two or four parts with key-parts enabling good students to move more quickly through a question. Weaker students, who generally opted to take their questions with all the parts on view, appeared not to suffer too much from lack of partial credit but some of the better students, who felt they could go directly to an answer through key-steps, occasionally failed to score the marks that a human teacher might have awarded. Students therefore need to strike a balance between speed obtained by just answering the key-parts and making some progress by clicking for more steps. This balance is for the individual student to judge and is a new examination strategy for the student to consider and practice in advance. The precise effect of key/sub-parts is analysed later in this chapter.

The problems that students have with mathematical input have already been discussed in depth in chapters 3 and 4. Figure 6.15, below, shows how the Input Tool integrates with the Mathwise Test. The pre-examination questionnaires highlighted the need for the Input Tool. In particular, students expressed concern that they would make syntactical errors as they input their answers and be unaware that the computer had mis-judged their answer. Questionnaires after the examination showed that the Input Tool dramatically reduced students' concerns about input, with comments like:

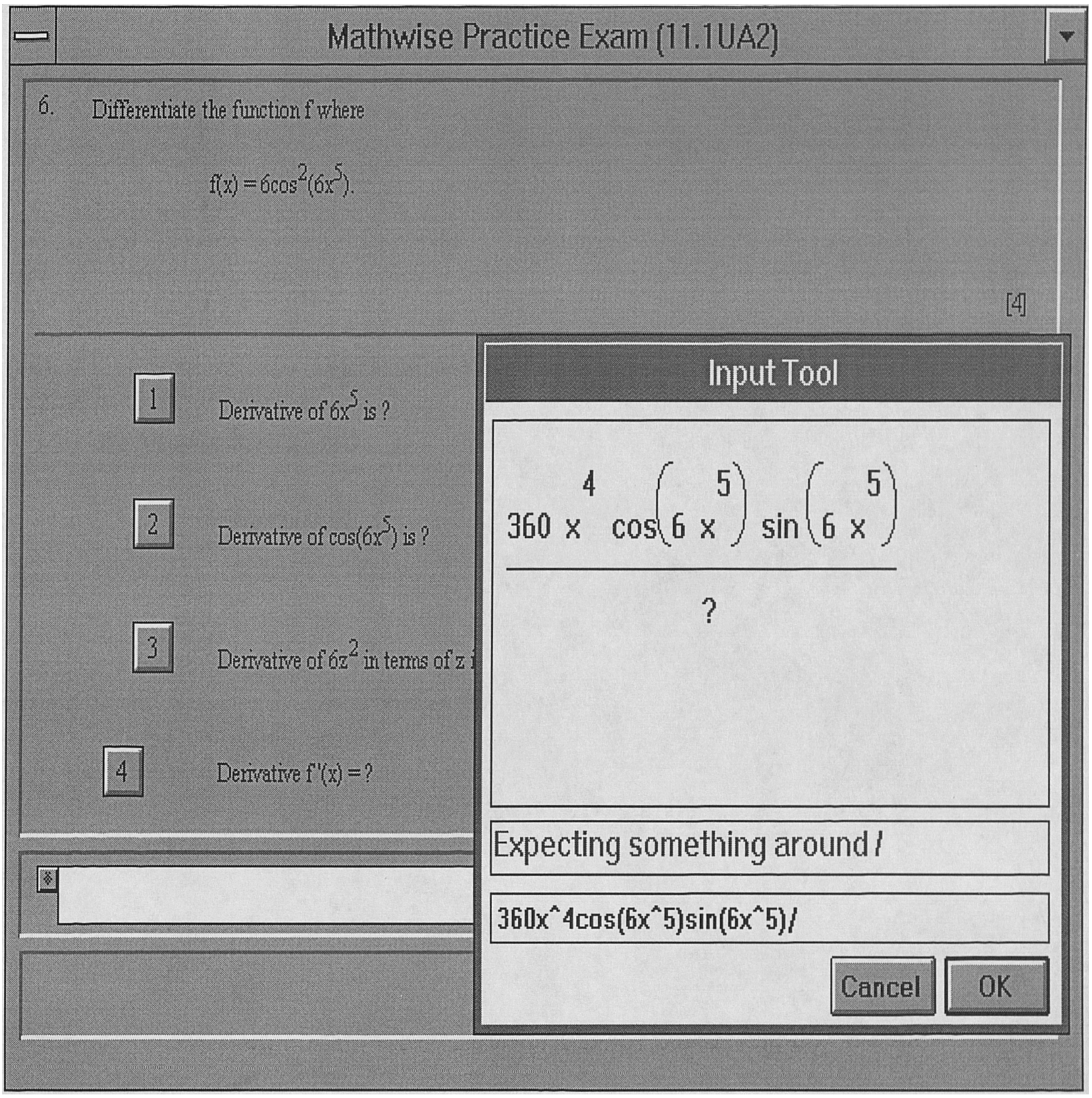
- "The Input Tool helps you see how the computer understands what you have written";
- "It made sure that I didn't make any input errors"; and
- "I was always sure that the answer was being constructed in the way that I wanted."

The effectiveness of the Input Tool is discussed more thoroughly later in this chapter.

6.4 Post-Examination Analysis

The most important criterion to ensure that the experiment was successful is that it provides consistent results when compared with more conventional ways of grading student ability. This section attempts to evaluate the effectiveness of the computer examination by means of some simple statistical analysis. It compares and contrasts the differences between computer-based and traditional written examinations. In addition, a report on information gathered through questionnaires and interviews is provided. Before this, however, some further detail is needed about the structure of the written and computer examinations.

For several years, December the written examination paper in Service Mathematics at Heriot-Watt University has been in two sections – Section A containing questions on Algebra (Trigonometry and Complex Numbers) and Section B on Calculus (Limits and Differentiation). In the last two years, this structure has evolved further so that each section divides into two parts with part 1 in each section having short questions worth 2 - 4 marks and a part 2 with longer questions worth 10


Mathwise Practice Exam (11.1UA2)

6. Differentiate the function f where

$f(x) = 6\cos^2(6x^5)$.

[4]

1 Derivative of $6x^5$ is ?

2 Derivative of $\cos(6x^5)$ is ?

3 Derivative of $6z^2$ in terms of z i

4 Derivative $f'(x) = ?$

Input Tool

$$\frac{360\,x^4\cos\left(6\,x^5\right)\sin\left(6\,x^5\right)}{?}$$

Expecting something around /

360x^4cos(6x^5)sin(6x^5)/

Figure 6.15: The Mathwise Mechanism using the Input Tool

marks each. In this way, 40 % of the written paper can be scored in the compulsory part 1 questions and a further 60% is available in the longer part 2 questions. It was envisaged that the computer could take over the assessment of the part 1 shorter questions and it is this analogy which this computer assessment experiment investigates.

It is important to point out that the students taking the computer examination were told it was an experiment and that their results would not count against their grade in the written examination. We were anxious not to disadvantage this group by asking them to do more work than the others in their year. Therefore, to re-dress the balance and give us a genuine comparison, the University gave permission for the results of the computer examination to be used in borderline cases. This incentive appeared to be enough to ensure that the bulk of the pilot group approached the computer examination in the right spirit though not always with the right preparation.

6.4.1 Partial Credit

After the examination, the students' .MRK and .TST files were analysed to see how the key-parts and sub-parts had been used. It was found that, as expected, the better students were answering key-steps whereas the weaker ones were revealing sub-parts. On a few occasions, these students would attempt to answer a key-part when perhaps they would have fared better if they had attempted the sub-parts first. The effect of this was sometimes very significant. This indicates, as with written examinations, that students need to practice their strategies for the computer examination. As part of the analysis, students' files were examined and credit was given to those students who answered a key-part wrongly but would have gained marks for the sub-part(s). This process was also aided by looking at the students' rough working on paper. Table 6.17, below, contains two rows; the first shows student marks before key-part credit was awarded and the second after credit was awarded.

Before	58	33	29	42	50	54
After	71	42	42	67	79	75

Table 6.17: Before and after awarding partial credit marks

This table shows the only significant changes due to partial credit. The two results of most concern would be those who passed only because of the human-added partial credit marks (i.e. the second and third columns). However, in combination with the written paper, such border line cases may disappear but if not then a human check may be necessary. Indeed, the inclusion of partial credit did not change the overall picture though it is believed that the original design of four part questions with only one key-step (i.e. the final answer in one step) should not be repeated. One strong recommendation from the results is that there should be at least two key-steps in any four part questions. In this way, the problem of partial credit can be minimised. From the work described in chapter 3, it may also be possible, in the future, to analyse wrong answers and award a proportion of marks for those with only one error. It is important to stress the need for students to practice their examination strategies in this novel mode of assessment.

6.4.2 The Input Tool

As described earlier in this chapter, the Input Tool was designed to decrease student concern about typing mathematical expressions into the computer. Because the Input Tool could be easily turned off during the examination, it is important to note how many students preferred to use it. Within the .TST files (described earlier in this chapter), a note is made whether the input tool was on or off. These files showed that with the class of 82 students, 80 preferred to use it. These files also indicated that there were very few occasions where students input incorrectly formed expressions (in terms of mathematical meaning, syntax or structure). In fact, those student who did not use the Input Tool had each made mistakes in input. The post-examination questionnaire asked the question:

- "Was the Input Tool useful and if so Why?"

Students were requested to grade their answer from 1 (useful) to 5 (useless). Some examples of the responses, which were the consensus, to this question have already been quoted and the result of the grading follows in table 6.18:

Response	1	2	3	4	5	Abstain
Frequency	26	19	28	4	3	2

Table 6.18: Responses to grading of Input Tool

The computer marked students' answers according to the rules described in Chapter 2 and a computer grade was recorded. This record was subsequently compared to the written results derived from a conventional pen and paper examination as described earlier.

The following diagram (Figure 6.16) gives an overall picture of the relationship between the students' computer mark and their corresponding part I written mark:

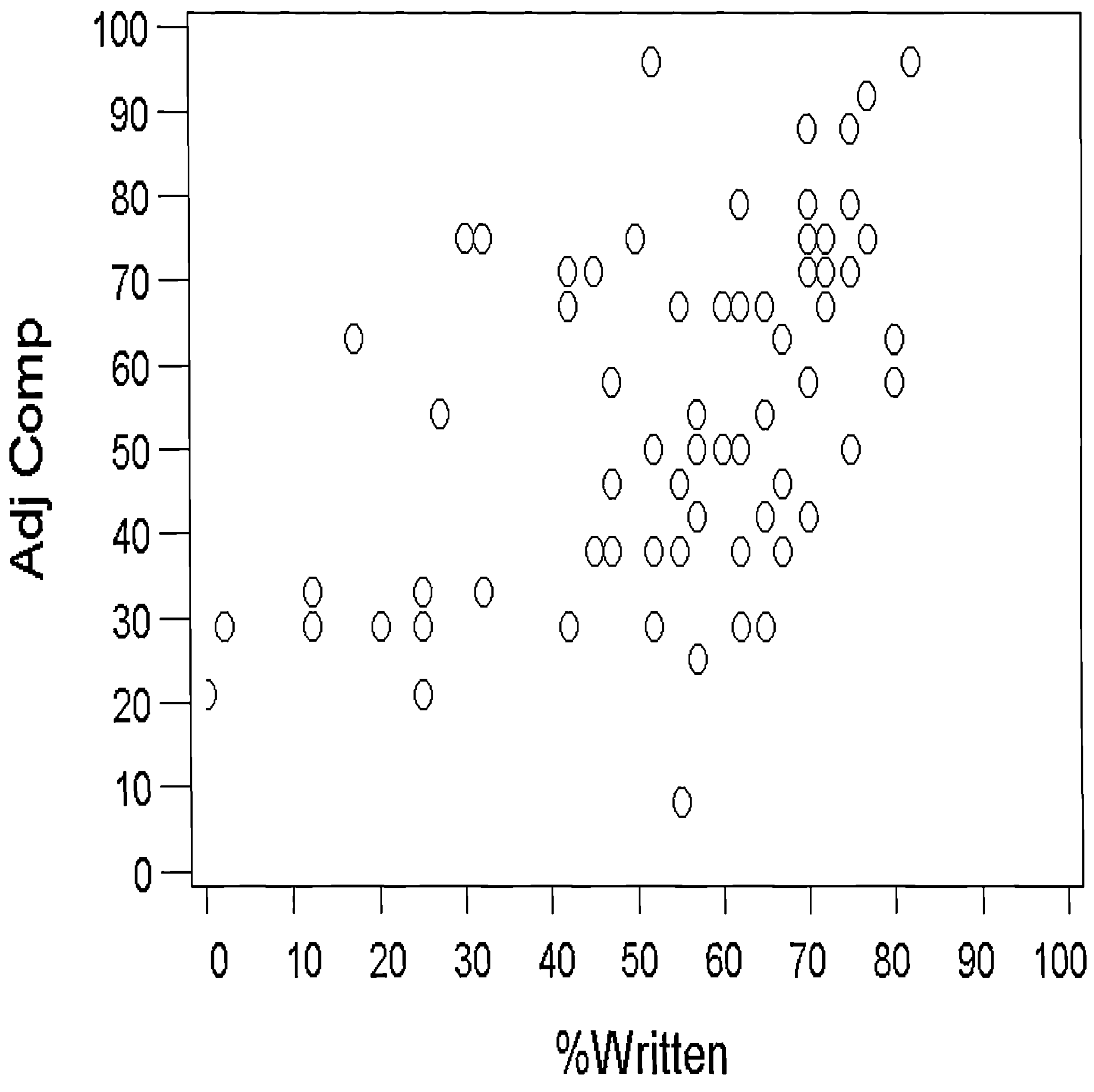


Figure 6.16: Relationship between computer and written marks

It can be seen from the diagram that there are some students with conflicting marks for the computer and written examinations. In order to explain why this was so, students with both similar and differing marks were interviewed. The vast majority of those students with differing marks were accounted for when they reported that they had either not revised or had not practiced using the computer software but had done their revision in the week in between the two examinations. Generally, the students found the tutorial room (where the computer examination took place) to be less daunting as the larger written paper examination halls This fact may account for the few students who scored higher marks on the computer. These students also found the computer conducive to better performances as they "related to computers more than examination papers".

Table 6.19 gives product-moment correlation coefficients for the computer examination and varying sections and parts of the written paper and Table 6.20 gives the correlation between the computer marks and the total part 1 marks (i.e. section A part 1 and section B part 1).

	Written Mark to Section A Part 1
Written Mark to Section B Part 1	0.579
Written Mark to Section A Part 2	0.563
Written Mark to Section B Part 2	0.437
Computer Mark (accounting for partial credit)	0.503

Table 6.19:

	Written Mark to Part 1 (Sections A and B)
Computer Mark (accounting for partial credit)	0.503

Table 6.20:

Looking at the Table 6.20 above, it can be seen that the correlation between the part I written and computer examination was 0.503. Although this is regarded as "highly significant" at a 1% significance level, it was thought, at first, to be disappointing. However, it is important to put this result in the context of other parts of the analysis. By looking at the correlations between the parts in the two sections of the written examination (see Table 6.19), it can be seen that there is a natural variability in the data. For example, the correlation between the marks scored by students in the part 1's of sections A and B is 0.579 and between part 2 of sections A and B the correlation is 0.563 — only slightly higher than for computer against written part 1. Taking this natural variability into account, the correlation of 0.503 can be regarded as satisfactory.

In order to improve the assessment strategy for future examinations, it is hoped that particularly significant parts of the computer examination can be assessed. For example, the use of the Input Tool has been discussed and the following graphs illustrate some other considerations, including "Ease of use" of the computer software,

"Amount of time" available in the examination and "Difficulty" of the examination. The symbol 2 which appear on these graphs show a point where 2 students' responses are plotted on the same point on the graph.
Students were asked whether they thought the software was easy to use. Figure 6.17, below, shows students' computer marks plotted against their corresponding written marks. The points are marked with a grading letter from A to E. An A indicates a student response of "easy to use" through to E which indicates that it was "difficult to use".

```
          -                                    C                   A
        90+                                                B  BC
          -
  Adj Comp-                     DB           D       2     2C 22
          -                             B C                DC A
          -           B                D        D  2C C B  C    D
        60+                               D                C     C
          -                  C                   C    C
          -                               E   B CB 2E   B    A
          -                                      E    2   B
          -        A        C   D        DC   E C   C   D
        30+  D      A    D   *          B      C     A *
          - C                B                    C
          -
          -
          -                                     E
         0+
            +---------+---------+---------+---------+---------+---------+------%Written
            0         15        30        45        60        75
         N* = 19
```

Figure 6.17: Ease of use

Looking at figure 6.17, there are no particular patterns in the data. This shows that the software (in terms of useability) did not discriminate against students' mathematical ability. This is, of course, an extremely important consideration when using computers to grade students.

Figure 6.18 shows students' responses to the question "Was there enough time to complete the computer examination?". Again, the points are plotted using a key — this time A represents an answer of "not enough time" and E represents "plenty of time". Looking at the data shows two distinct groups. The first, marked 1, indicates a group of able students (those who gained high marks in the computer examination) who had enough time to complete the examination. The second group, marked 2, shows that the weaker students (those who gained lower marks on the computer) reported that they had too little time. Again, this is a result which one might expect.

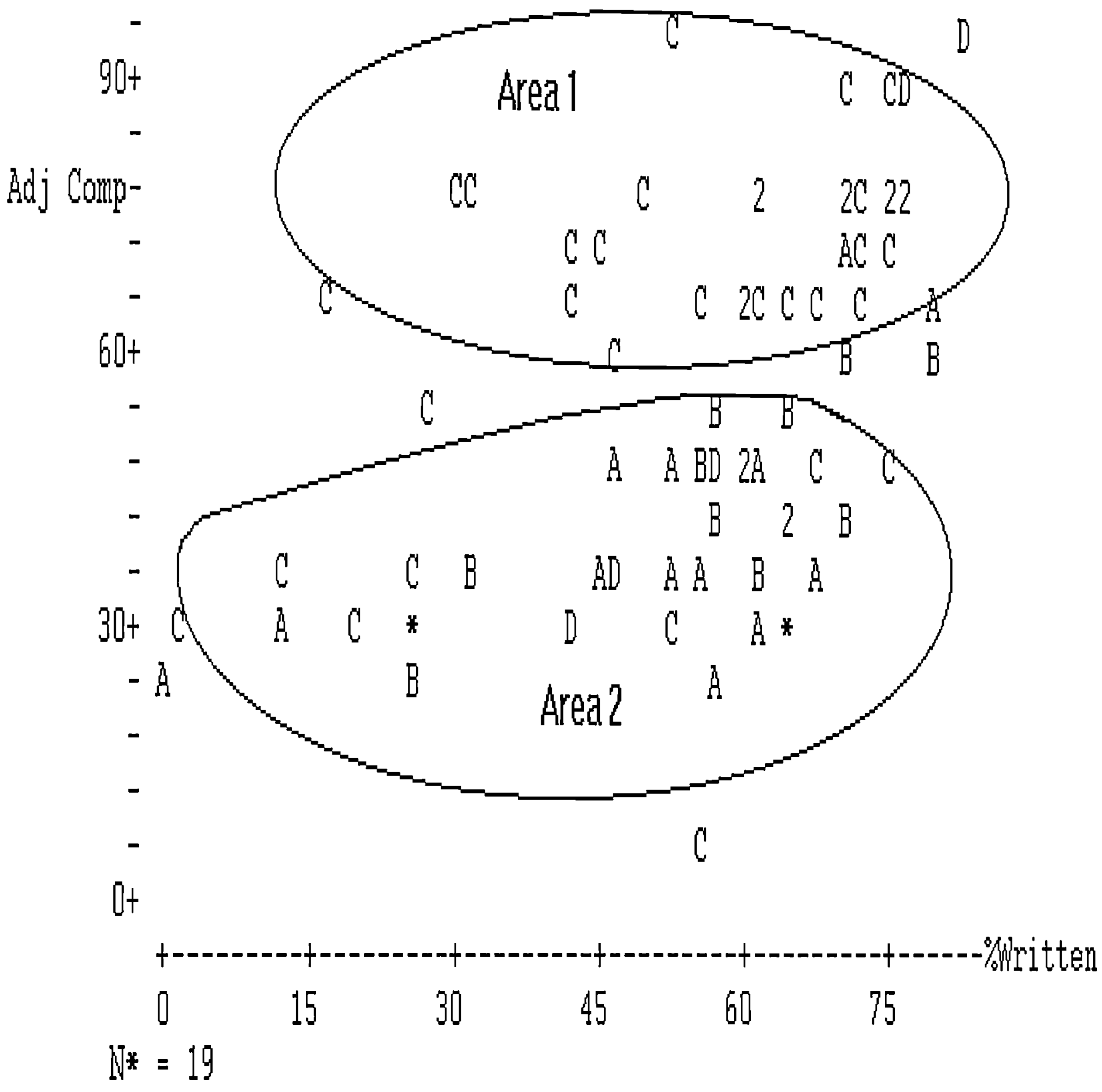


Figure 6.18: Enough time

6.5 Conclusions

Overall, the experiment can be considered a success. The experiment has given insight as to how to improve the testing mechanisms and, with the benefit of hindsight, these improvements can now be addressed. Notably, a re-design of the four part one key step questions will be carried out and a facility to enable students to update their answers at any stage during the examination will be included. However, the accuracy of the computer examination to predict mathematical ability seemed clear with 5 out of 82 failures in the computer examination and the same number in the written examination. They were not the same five students in each case but there were 3 students who failed on both the computer examination and the written paper.

It is important, of course, to justify the use of computers in this way and to highlight benefits that an automatic testing mechanism can provide from both teachers and students' viewpoints. The advantages for students are that the computer is consistent; that is, it gives the same marks for identical correct or incorrect answers for each student. The computer does not look at the front of the answer booklets, and therefore does not favour particular students. Also, with the advent of modular courses and student centred learning, sitting examinations more frequently becomes increasingly important. The computer offers countless different examination papers in one package because of the in-built randomness and therefore offers re-sit examinations on request. In fact, two students who narrowly failed were offered a resit examination at the start of the following term. One student passed the re-sit sufficiently well to carry their total marks for that module to a pass. This flexibility in the delivery of re-sit examinations is a genuine bonus for modern students.

This leads naturally to what advantages there are for staff. As the last paragraph describes, easier to set re-sits means that staff have to spend less time in setting such papers. Of course, this is true for setting examination papers in general and the time saved on staff setting and marking examination scripts can be spent on helping students learn. The real benefit to teachers appears in the greater flexibility that the computer examination brings. There is a real prospect of letting the students decide when they are ready to take the grading assessment for the course. Indeed, such a system enables an approach like that of the vehicle driving test to be adopted in education with the concomitant advantages of forcing down the failure rate.

The other obvious advantage to teachers is that of time saved in marking traditional examinations. It has been estimated that over a period of 5 years, over 350 person hours can be saved for a group of 400 students per year for the type of examination considered in the experiment.

6.5.1 A Year On

After reporting the above conclusions, the use of the Computer for formal examinations was encouraged by the University. Withstanding this 268 students sat, as part of their formal assessment, a computer examination at the end of a term's module. The examination took the same form as the experiment in the previous year. The data gathered from this second examination, although slightly less detailed, was then used to judge if the conclusions and inferences made above were justified.

Firstly, with the increase in population size, any conclusions drawn were more statistically valid and therefore give a more accurate measurement of any findings. This second analysis starts with some product moment correlations for the different elements of the examination - shown in Table 6.21.

-	Section B (%)	Section A (%)	Computer Exam (%)
Section A (%)	0.627	-	-
Computer Exam (%)	0.552	0.521	-
Written Paper Total (A and B) (%)	0.888	0.915	0.595

Table 6.21: Product Moment Correlations

First notice the figure of 0.595 which shows the correlation between the written and computer examinations. This figure, although slightly higher than for the first experiment is still surprisingly low. However, as before it is important to compare this with the correlation between the different sections on the written paper - 0.627. These results back up the conclusion from the first experiment in that there is large natural variability with this sort of data. However, one might expect more variability since the written paper consisted of just the longer type questions. More importantly, it re-emphasises the need to address the traditional methods of assessing students ability. Figures 6.19 6.20 and 6.21 give a good pictorial representation of this variability. These graphs plot marks for Section A against Computer, Section B against Computer and Total Written against Computer respectively.

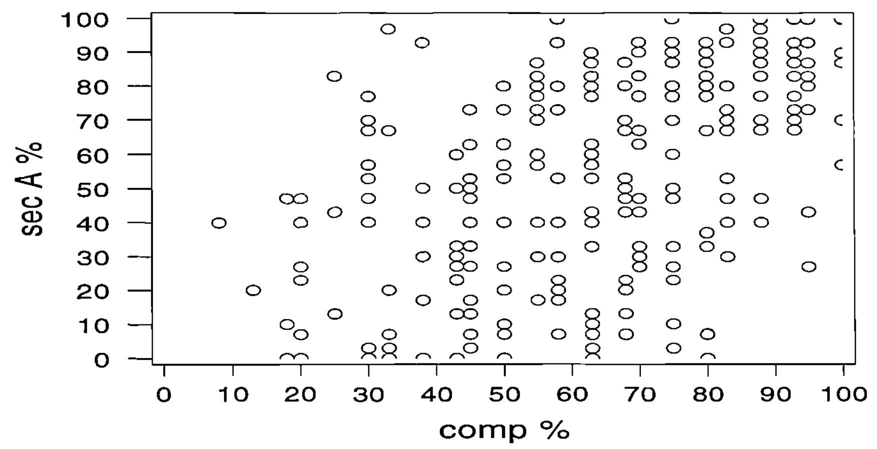


Figure 6.19: Graph Showing the 'natural variability' between Section A of the written paper and the Computer examination

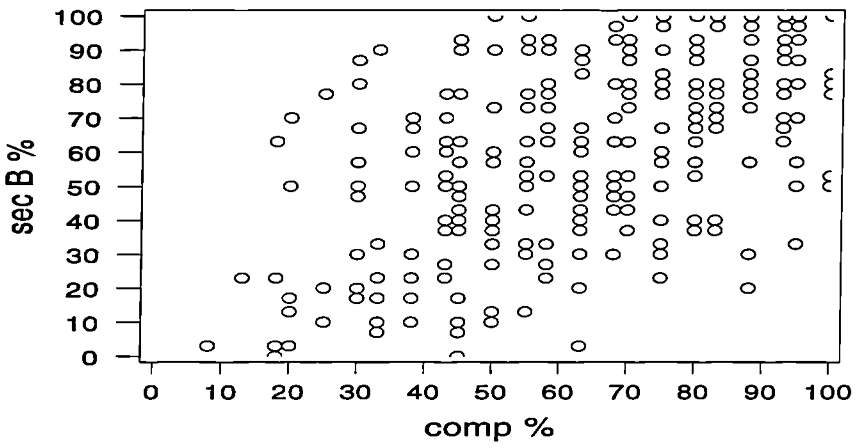


Figure 6.20: Graph Showing the 'natural variability' between Section B of the written paper and the Computer examination

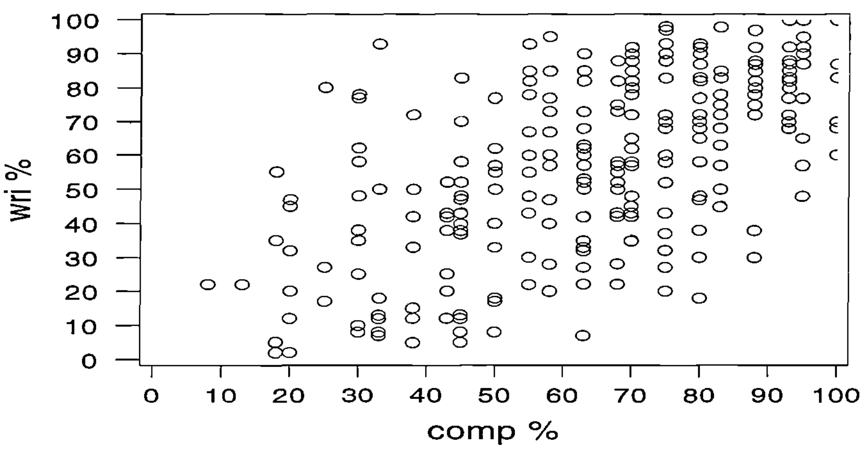


Figure 6.21: Graph Showing the 'natural variability' between the total written paper and the Computer examination

This second analysis has therefore shown that, overall, the computer examinations have given just as a reliable picture of students' mathematical ability as the written examinations. Moreover, the computer, again, brought forward benefits to the student that the written papers couldn't - the flexibility of re-take examinations. In fact, out of the 268 students who sat the examinations, 21 students attempted the computer examination a short time after the first sitting. From these 21, 6 students gained enough marks to increase their score to the pass mark of 40%[6]

[6]To be fair to those students who had already passed the module, the maximum mark for re-take students was a 40% pass.

Chapter 7

Future Research and Developments

The work described in this thesis has provided foundations for future research and development within the field of Computer Assessment of Mathematics and in particular the CALM Project. These developments are described below.

7.1 An Improved Answer Comparison Routine

Improved numerical algorithms, along with their associated data structures, are currently being developed . These routines will continue using the work with trees, whilst implementing N-ary structures [1] thus providing the ability to compare more answers such as vectors, matrices, lists and sets whilst enabling mathematical properties such as associativity and commutativity to be recognised. It uses an extendible grammar which, along with these new structures, enables future integration of different answer types and different methods of comparison.

[1]an N-ary structure can have N children as a sub-structure. Therefore an N-ary Tree is a tree where any node can have N children, each being an N-ary Tree

7.2 A New Input Tool

After realising the importance of the Input Tool, a new version is being constructed which implements several improvements. Characters can be displayed using a variable size font and special symbols, such as π, $\surd$ and Greek letters have been instrumental in the development of the tool. The Input Tool window can change size as an expression expands, therefore allowing far larger and more complex expressions to be input. This work may lead towards more complex answers (such as those containing comparison operators) being asked within a mathematics question.

7.3 A New Assessment Mechanism

The work highlighted in Chapter 6 has spurred the evolution of another assessment mechanism with the fuelling of the ongoing TERCENT (Test Enhancement Research - Computer Examination Network Trials) proposal[2]. An AIM (Assessment In Mathematics) engine has been created which will be used to deliver mathematics testing over a Local Area Network (LAN) and the Internet. This will enable computer examinations to be sat outside of the university and at school level throughout Scotland. The results and conclusions from Chapter 6 will provide a basis for the educational testing of the network trials.

[2]this proposal won the first Bank of Scotland Tercentenary Award to Education

Appendix A

An Example Using Dijkstra's Algorithm from Chapter 4

Given the expression `x+SIN(y/3)` $= x + \sin\left(\frac{y}{3}\right)$ the lexical analysis procedure would produce and array of tokens A as: `A[x,+,SIN,(,y,/,3,)]`. Note that the elements of A are stored, for example, as A[1].NodeStatus=IdentifierNode and A[1].WhichIdentifier=xx but for simplicity they will sometimes be written in the former notation.

Assume that the postfix array in which tokens are being stored (in postfix order) is called B. As the example progresses the values of OC (the Output Counter used to iterate through the elements of B) and IC (the Input Counter used to iterate through the elements of A) will be given.

The stack used is called S.

OC=1, IC=1 Get the token A[IC]=A[1].

A[1].NodeStatus = IdentifierNode A[1].WhichIdentifier = xx

Is A[1].NodeStatus in [RealNode, IdentifierNode, ConstantNode]? **ANSWER - YES**

Hence B[OC]:=A[IC] i.e. B[1]:=A[1]

Therefore the array B becomes `B[xx,....]`

OC=2, IC=2 Get the token A[IC]=A[2].

A[2].NodeStatus = OperatorNode A[2].WhichOperator = Plus

Is A[2].NodeStatus in [RealNode, IdentifierNode, ConstantNode]? **ANSWER - NO**

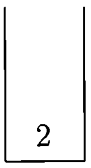
Is A[2].NodeStatus = DelimiterNode? **ANSWER - NO**

TEST CONDITION $WHILE\ \underbrace{(\ NOT\ \underbrace{(EMPTY(S\ ))}_{TRUE}\ )}_{FALSE}$

(ignore inside the WHILE loop)

Default to: PUSH IC (=2) onto the stack S so that S becomes

S=



OC=2, IC=3 Get the token A[IC]=A[3].

A[3].NodeStatus = FunctionNode A[3].WhichFunction = FSIN

Is A[3].NodeStatus in [RealNode, IdentifierNode, ConstantNode]? **ANSWER - NO**

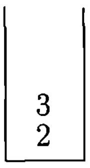
Is A[3].NodeStatus = DelimiterNode? **ANSWER - NO**

TEST CONDITION

$$WHILE\ \underbrace{\underbrace{(\ NOT\ \underbrace{(EMPTY(S\ ))}_{FALSE}\ )}_{TRUE}\ AND\ \underbrace{(\underbrace{Prec(A[3])}_{100} \leq \underbrace{Prec(A[Top(S)])}_{2})}_{FALSE}}_{FALSE}$$

(ignore inside the WHILE loop)

Default to: PUSH IC (=3) onto the stack S so that S becomes

S=


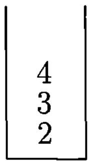
OC=2, IC=4 Get the token A[IC]=A[4].

A[4].NodeStatus = DelimiterNode A[4].WhichDelimiter = OpenBracket

Is A[4].NodeStatus in [RealNode, IdentifierNode, ConstantNode]? **ANSWER - NO**

Is A[4].NodeStatus = DelimiterNode? **ANSWER - YES**

Since A[4].WhichDelimiter = Open bracket then PUSH IC (=4) onto the stack S so that S becomes

S=


OC=2, IC=5 Get the token A[IC]=A[5].

A[5].NodeStatus = IdentifierNode A[5].WhichIdentifier = yy

Is A[5].NodeStatus in [RealNode, IdentifierNode, ConstantNode]? **ANSWER - YES**

Hence B[OC]:=A[IC] i.e. B[2]:=A[5]

Therefore the array B becomes `B[xx,yy,....]`

OC=3, IC=6 Get the token A[IC]=A[6].

A[3].NodeStatus = OperatorNode A[3].WhichOperator = Divide

Is A[6].NodeStatus in [RealNode, IdentifierNode, ConstantNode]? **ANSWER - NO**

Is A[6].NodeStatus = DelimiterNode? **ANSWER - NO**

TEST CONDITION

$$\underbrace{WHILE\ (\ \underbrace{NOT\ \underbrace{(EMPTY(S\ ))}_{FALSE}\ )}_{TRUE}\ AND\ \underbrace{(\underbrace{Prec(A[6])}_{3} \leq \underbrace{Prec(A[Top(S)])}_{0})}_{FALSE}}_{FALSE}$$

(ignore inside the WHILE loop)

Default to: PUSH IC (=6) onto the stack S so that S becomes

$$S = \begin{array}{|c|} 6 \\ 4 \\ 3 \\ 2 \\ \hline \end{array}$$

OC=3, IC=7 Get the token A[IC]=A[7].

A[7].NodeStatus = RealNode A[7].WhichReal = 3.0

Is A[7].NodeStatus in [RealNode, IdentifierNode, ConstantNode]? **ANSWER - YES**

Hence B[OC]:=A[IC] i.e. B[3]:=A[7]

Therefore the array B becomes `B[xx,yy,3.0,....]`

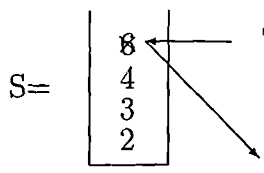
OC=4, IC=8 Get the token A[IC]=A[8].

A[8].NodeStatus = DelimiterNode A[8].WhichDelimiter = ClosedBracket

Is A[8].NodeStatus in [RealNode, IdentifierNode, ConstantNode]? **ANSWER - NO**

Is A[8].NodeStatus = DelimiterNode? **ANSWER - YES**

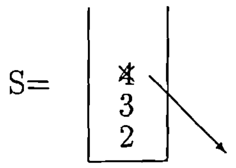
Since A[8].WhichDelimiter = Closed Bracket then keep popping the stack S and filling the array B until a delimiter is found at the top of the stack.



Top of the stack is popped

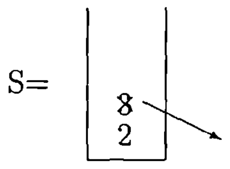
So B[OC]=B[4]:=A[6]="/"
i.e. the array B becomes `B[xx,yy,3.0,/,....]`

OC=5, IC=8 Now the stack S is popped again as shown below:



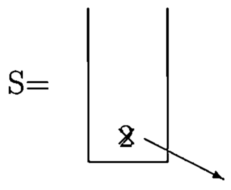
This element is popped. However, since this is a delimiter (since A[4].NodeStatus=DelimiterNode) then it is not output

OC=5, IC=9 Since A[IC] is empty (i.e. the end of the input array A has been reached) continue by popping what is left in the stack S, whilst outputting to the array B. This is shown in the diagram below:



A[3] = "SIN" so let B[OC]=B[5]:="SIN".
Hence the array B becomes `B[xx,yy,3.0,/,SIN,....]`

OC=6, IC=9 The last element in the stack S is now popped as shown below:



A[2] = "+" so let B[OC]=B[6]:="+".
Hence the array B becomes `B[xx,yy,3.0,/,SIN,+,....]`

The algorithm finishes and the output array B has been created as `B[xx,yy,3.0,/,SIN,+]` which is the postfix form of the input array A.

Appendix B

The Product-Moment Correlation Coefficient

Correlation is defined (see [13]) as:

> **the extent of correspondence between the ordering of two RANDOM VARIABLES. It is a** *positive correlation* **when each variable tends to increase or decrease as the other does, and a** *negative* **or** *inverse correlation* **if one tends to increase as the other decreases**

which leads to the product moment correlation coefficient begin defined as:

> **a statistic that measures the linear relationship between two variables in a sample and used as an ESTIMATE of the CORRELATION, ρ, in the whole population.**

The correlation coefficient between the two variables x and y, say, is usually denoted r and can be calculated as

$$r = \frac{S_{xy}}{S_x S_y} = \frac{\frac{1}{n}\sum(xy) - \overline{x}\,\overline{y}}{\sqrt{\frac{1}{n}\sum x^2 - \overline{x}^2}\sqrt{\frac{1}{n}\sum y^2 - \overline{y}^2}}$$

where S_{xy} is the covariance and S_x and S_y are the standard deviations of the x and y variables respectively.

References

[1] Abas S.J. "Rapid Turbo Pascal Graphics Tutor", *Institute of Physics Publishing ISBN 0 471 63777 7* (1992)

[2] Abas S.J., Mondragon J.R. "Pascal: An Interactive Text" *ISBN 0 75030 020 5* (1990)

[3] Beevers C.E., Foster M.G., Korabinski A.A., McGuire G.R. "The Use of Computer Aided Learning in Mathematics" *Collegiate Microcomputers Vol. 11 No. 3 pp. 199-204* (1993),

[4] Beevers C.E., Cherry B.S.G., Clark, D.E.R. Foster M.G., McGuire G.R., Renshaw J.H. "Software Tools for Computer Aided Learning in Mathematics", *International Journal of Mathematics Science Technology. 20 561-569* (1989)

[5] Beevers C.E., Foster M.G., McGuire G.R. "The CALM Before the Storm!" *Computers and Education Vol 12. pp 43-47* (1988)

[6] Beevers C.E. "Assessment Aftermath" *International Conference for Teaching in Mathematics Technology - conference proceedings pp. 46-58. Napier University* (March 1996).

[7] Beevers C.E., Foster M.G., McGuire G.R., Cherry B.S.G "Software Tools for Computer Aided Learning in Mathematics", *ISBN 1 85628 172 8* (1991)

[8] Beevers C.E. "Motivating mechanics", *IMA Journal of Mathematics Teaching* pp. 52 - 56 (1985)

[9] Beevers C.E., Foster M.G., McGuire G.R. "Some Problems of Mathematical CAL" *Computers and Education Vol. 18 No. 1/3 pp. 119* (1992)

[10] Beevers C.E. "Mechanics with a micro" *Proc. 2nd. Int. Conf. on Math. Modelling, Exeter University* (1985)

[11] Beevers C.E., McGuire G.R., Stirling G., Wild D.G. "Mathematical Ability Assessed by Computer" *Computers Educ. Vol. 25, No. 3, pp. 123-132, Pergamon Press* (1985)

[12] Borland Turbo Pascal for Windows Manuals (Programmer's Guide) pp. 215-220 (1991)

[13] Borowski E.J., Borwein J.M. "Dictionary of Mathematics", *Collins Reference ISBN 0 00 434347 6*(1989)

[14] Brydges S., Hibberd S., "Construction and Implementation of a Computer-based Diagnostic Test", *CTI newsletter (Maths and Stats)* (Aug 1994)

[15] Bull J. "Using Technology to Assess Student Learning" *The UK Universities' Staff Development Unit and The Universities of Kent and Leeds ISBN 1 85889 091 8* (December 1993)

[16] Burch J.G., Grudnitski G. "Information Systems - Theory and Practice (5th Edition)", *Wiley ISBN 0-471-61293-6*

[17] Crowder, N.A "Automatic tutoring by means of intrinsic programming", *Automatic Teaching: the State of the Art*, New York: Wiley. (1959)

[18] Falchikov N., Boud D. "Student Self-Assessment in Higher Education: A Meta-Analysis." *Review of Educational Research 59, 395-430*

[19] Kane D., Sherwood B. "A Computer-Based Course in Classical mechanics" *Computers and Education. Vol 4, pp. 15-36*

[20] Kimball, R.B. "A self-improving tutor for symbolic integration", *Intelligent Teaching Systems*, London: Academic Press. (1982)

[21] Kulhavy, R.W. "Feedback in Written Instruction", *Revue of Educational Research, 524-44*(1977)

[22] J. Lomax "Individually Parameterised Assessments" *Software for Engineering Education (CTI)* (Autumn 1995)

[23] Mann K.J., "International Computerised Testing in Mathematics", *Proceedings Fifth Annual International Conference on Technology in Collegiate Mathematics, Chicago, Illinois, USA, Nov 12-15, (1992)*

[24] Matz M. 'Intelligent Tutoring Systems' *Process Model for High School Algebra Errors* Massachusetts Institute of Technology

[25] McCabe M. 'Designer Software for Mathematics Assessment' *Maths and Stats Quarterly Vol. 6 No. 1 pp 11-16*

[26] . McCracken D.D. "A Second Course in Computer Science with Pascal" *Wiley ISBN 0 471 63777 7* (1987)

[27] Minsky M. "Plain Talk About Neurodevelopmental Epistemology." *MIT AI memo 430*

[28] Moi H.D.N. "An Interactive Space for Mathematics" *Ph.D Thesis, Heriot-Watt University*

[29] Moi H.D.N. Bulletin of the Institute of Mathematics and its Applications, Vol. 31, Nos. 11/12, pp. 163-166 November/December (1995).

[30] Neill N.T. "Examining Mathematics in a Laboratory" *Bulletin of the Institute of Mathematics and its Applications Vol. 31, Nos 11/12 Nov,Dec 1995 pp 163-166*

[31] O'Shae, T & Self, J "Learning and Teaching with Computers", *Artificial Intelligence in Education ISBN 0 7108 0665 5* (1983)

[32] Palmer, B. G. and Oldehoeft, A.E. "The design of an instructional system based on problem-generators", *Int. J. Man-Mach. Stud., Vol. 7, pp. 249-71*

[33] Pritchett N., Zakrzewski S. "Computerised Formal Assessment" *British Journal of Educational Technology Vol. 26 No. 2* (May 1995)

[34] Harding R.D., Lay S.W., Moule H. and Quinney D.A. "A Mathematical Toolkit for Interactive Hypertext Courseware: Part of the Mathematics Experience within the Renaissance Project." *Computers and Education Vol. 24 No. 2 pp. 127* (1995)

[35] Sherwood B.A. "The TUTOR Languagc". *Am. Phys - Communications of the ACM* (1972)

[36] Sherwood B.A "The TUTOR Language" *Eagan, Minnesota: Control Data Education Company*

[37] Skinner "The Technology of Teaching", New York: Appleton Century-Crofts

[38] Strickland P. CTI "The SLOTH System for Computer Aided Assessment" *Maths and Stats Quarterly Vol. 5 No. 4*

[39] Tenenbaum A.M. and Augenstein M.J. "Data Structures Using Pascal" *Prentice Hall, Inc. ISBN 0 13 196501 8* (1981)

[40] "Teaching and Learning in an Expanding Higher Education System", *The Committee of Scottish University Principals ISBN 0 9519377 1 5*

[41] Wild D.G. "Assessment in Mathematics - a Problem and its Possible Solution" *TLTP Workshop Papers on Assessment pp. 37 - 40, Sheffield* (1993)