Computing Project

Nathan Bartram

Table of Contents

Analysis

Background

Mathematics is considered to be a difficult subject, this has lead to the idea that a tool could be developed to help with the learning of maths. As a result Head of Maths teacher Rob Lamb requested that a learning aid should to be investigated into and developed to help maths students with their studies.

Problem Definition

The problem was discovered when Rob Lamb wanted students to have better opportunities and encourage them to study outside of lessons, this should help raise the standard of grades in the department. At home, a student may have little guidance with their revising, if they make a mistake when solving an equation it is usually difficult to backtrack and find the problem. Therefore, a solution must be found to solve this problem by creating a system that provides feedback to students and use this functionality to help with maths revision.

Investigation

An interview with Rob Lamb revealed that a system that can show the step by step solutions to maths problems will help students a great deal, particularly GCSE resit candidates. Some inconveniences are that some students may require extra tuition to help with their studies but due to time constraints not everybody can get the guidance they need. A tool that could provide feedback and support would help students achieve the qualifications they need to pursue a job career. To help his students, Rob decided to ask the computing department of Hartlepool 6th Form College where he works to determine if there is a computational solution.

Observation

During an observation of a maths lesson with Rob Lamb as the teacher in a typical lesson, students are initially taught the theory of a topic. To reinforce understanding an active example is solved which gives students the opportunity to ask questions and otherwise contribute to the discussion. The process continues with more examples of increasing difficulty, and then students are given an additional example to work out on their own to strengthen and test what they have learnt. If a student has a problem they can ask Rob for help to iron out problems, but this system is not without problems. One major concern is that Rob can only help a single student at any time because each person has individual problems and questions. Sometimes students have a global or general predicament which Rob will detect and stop the class to explain and clarify the dilemma, rather than repeat the same procedure several times. This is the leading factor to the decision that a system should be created to complement and work alongside the current one is because Rob has became the limiting factor on the performance of the current system. This is because of a bottleneck effect were the maximum number of students Rob can offer help to per lesson is much less than the quantity of students needing help. While students are waiting they could be using the system rather than doing nothing which will alleviate workload from Rob and allow students to have a greater amount of assistance. Not all students have the confidence to participate in discussions and as a result, dominate students receive a lot more attention that others. The new system will cater for both categories but shy students may benefit more from it because they may feel more comfortable using the system. The new proposed system should help users to practice and revise maths topics and provide feedback on performance as well provide all facilities that are required such as graph rendering and a calculator. The new system should not replace the old one because the knowledge of a human teacher allows dynamic two way communication, while a computerized system may not be as versatile. However, additional aid will benefit users when compared to basic revision methods because when both systems are is use the strengths of each will complement each other.

Interview

A lengthy discussion with Rob Lamb, the Head of Mathematics on 20th October 2010 revealed that students studying maths typically have many queries due to the nature of the subject. Rob stated that he spends a majority of lesson time helping students who are having difficulties and an aid to both help and free lesson time for other use would be a huge help. The aid should provide students with information on how to solve maths problems related to the topic they are struggling with, and should be tailored towards GCSE resit candidates. The reason behind the decision is that A-level students do not need as much help as GCSE students because they have a higher level of self motivation and dedication to their studies.

In addition to the basic functionality, more abilities will be required because providing all the required tools in one package will be convenient to the end user, and not all students remember to bring their calculators. An additional problem is graph drawing; college graphical calculators are slow to draw and are user unfriendly, there is a new model available but they are too expensive for the maths department to replace the current devices in use. The system must be able to draw graphs quickly because students who use the calculators for the first time become annoyed and frustrated as a result of its problems, a replacement must be created to solve this issue. Another problem with the graphical calculators is that data input is set on override rather than insert, which means if the user makes a mistake and deletes it, everything after the error must be re-entered. This is another contributing factor to why students dislike the graphical calculators. Also when entering large equations into scientific or graphical calculators, it is logical break it up into smaller parts, but because the device only supports two variables, students must write it down on paper and re-enter when needed. This can cause the final answer to be incorrect because of rounding errors if the degree of accuracy of the recorded number is not sufficient and it is very time consuming writing down numbers to ten significant figures. Past equations are stored in memory, but calculators have a very limited capacity for how many they can store which can result in unnecessary retyping. This is a clear indication that the proposed new system should be able to allow users to draw graphs quickly and provide dynamic variables as well as an insert editing of equations.

Rob stated that the GCSE maths has two options which are the higher and foundation tier, in order to cater for both the system should be able to set the difficulty of a problem based on which tier a student is taking. This will target questions that are relevant to the current user and will help focus students to practice what they will be tested on. Also, Rob pointed out that there are different methods in which students learn, most maths students are visual learners indicating that the new system must cater for such users, but others much not be neglected because this will limit the number of potential users. At the end of the interview Rob suggested that the solution should comply with the following objectives:

Suggested objectives

1 Revising a topic

  1.1 Set difficultly of topic

  1.2 Guided though topic using concise presentation

  1.3 Questions are generated by computer

  1.4 To be able to be shown the solution

2 Extra features

  2.1 Expression evaluator

  2.2 Dynamic user defined variables

  2.3 Draw graphs

A prototype was developed and then was sent to Rob, he replied stating that he wanted to improve and modify the objectives which can be seen in the objectives section below. The letter is in the appendix.

Questionnaire Results

Questionnaires were issued to a class of thirty maths students, a copy and the quantitative results are presented in the appendix. The questionnaire was constructed based information from Rob in the interview in order to get the thoughts and opinions of the participation students. Confidence in maths students follows a normal distribution that centres on the medium mark meaning that there is a clear indication that there is room for improvement. The majority of students revise between two and eight hours which on a weekly basis which could be used for substantial use of the system. Rob was correct in stating that most maths students are visual learners because over two thirds stated they use visual revision methods. There were no auditory learners and as a result nobody used such type of revision and there was also a lack of kinaesthetic learners. Over half of the class used computer based e-learning usually in the form of interactive websites and other software as well as relying on cramming before an exam to improve their performance. A moderate percentage of students improve by using an experiential tactic and others use flash cards to revise indicating that these two groups will have a medium priority when implemented into the proposed system. Almost eighty percent of student believe that they would benefit from extra revision material with reasoning including that it will allow both practice and accommodate for a greater range of question variations in order best tackle the final paper.

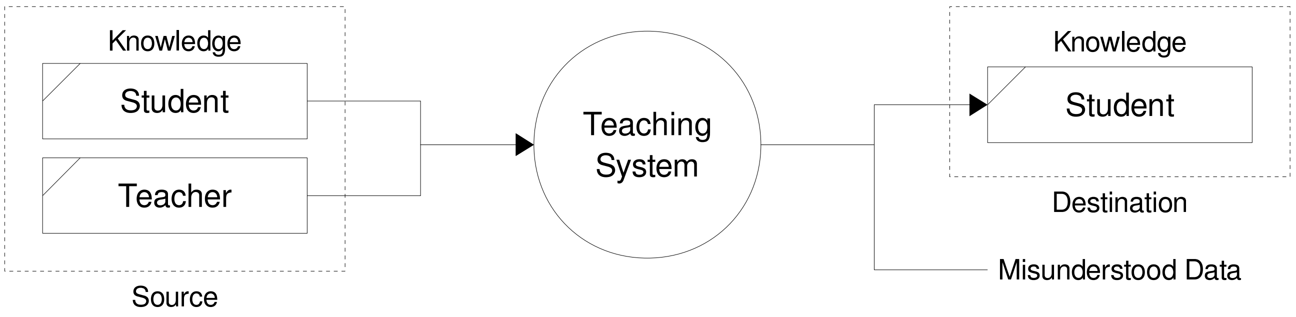
All students own a computer with an internet connection and have a good understanding on how to use them, eighty percent of candidates believe that they would use computer based revision system. The questions about what revision methods users would like to see in the system may seem like a repeat of an earlier question on a similar topic but some paper based methods do not work well in a computer based environment. Most liked the variable difficulty and graph rendering but also suggested a number generator and unit converter to help with questions about number patterns and units of measurement because using a paper based method is error prone and tedious.

The final question aimed to determine whether the students thought that a replacement to scientific and graphical calculators were necessary by asking them about the faults of the device. Many students expressed that graph drawing was frustrating on graphical calculators while there was an even opinion between the number of variables and the screen size being either too small or adequate. While the majority found using the calculators difficult, only three people found it easy showing that the devices need to be improved or replaced. After analysing the results of the questionnaire objective have been set based on them so that the system can help meet the needs of GCSE maths students.

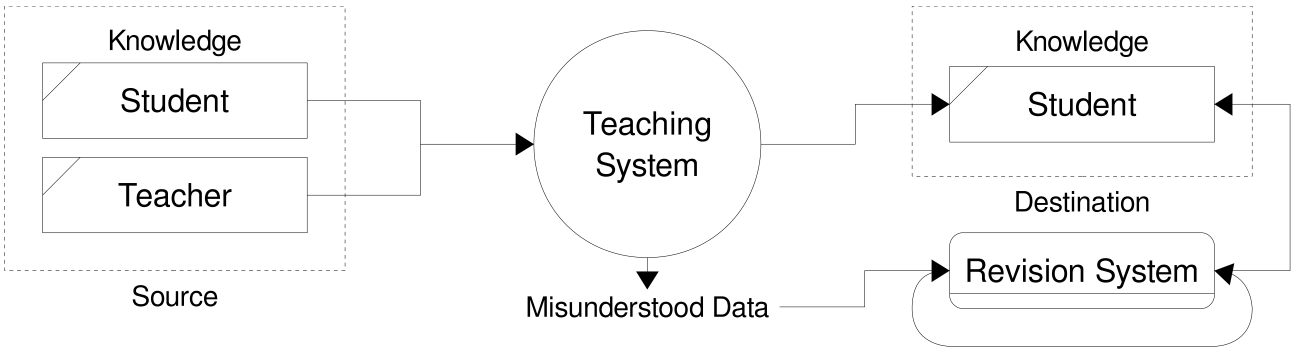
1 Revising a topic
   - 1.1 Wide variety of maths topics available to the user
   - 1.2 Users can set the difficultly of a topic
   - 1.3 Questions are generated by computer
   - 1.4 Users can be able to be shown the solution

2 Extra features
   - 2.1 Dynamic user defined variables
   - 2.2 Ability to draw graphs
   - 2.3 Unit converter utility
   - 2.4 Number generator

Context Diagrams



Some problems occur when the system has completed a cycle and the misunderstood data does not have a destination. This in turn creates a gap in pupil knowledge; as a result this could lead to a decrease in potential pupil performance.



The proposed new system will have an added component that will allow users to be able to learn topics that were not previously understood. It will also enable users to be able to practice topics that they already comprehend so that they will rise to a higher standard of performance.

Data Flow Diagram

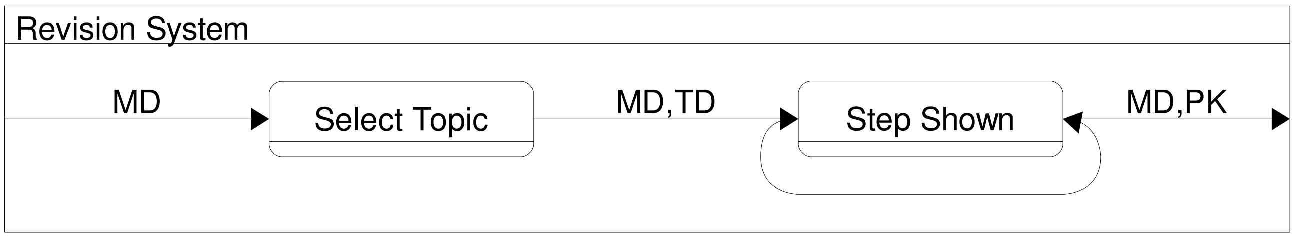
Data Definition

MD: Misunderstood Data - Topics that a student does not understand

TD: Topic Data – Data needed for question generation

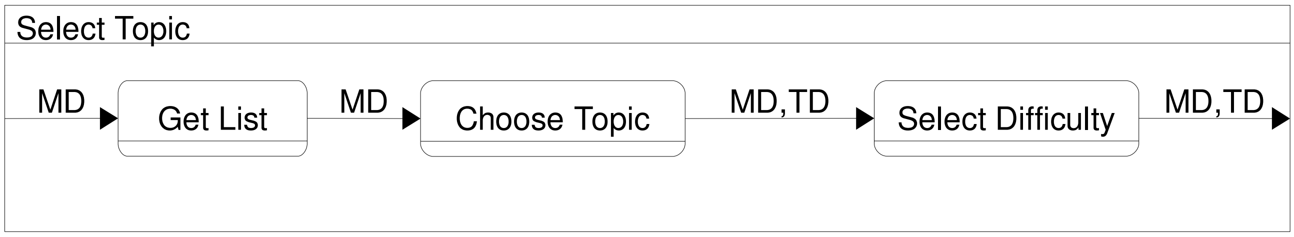
PK: Pupil Knowledge – Topics that a student understands

Level 1

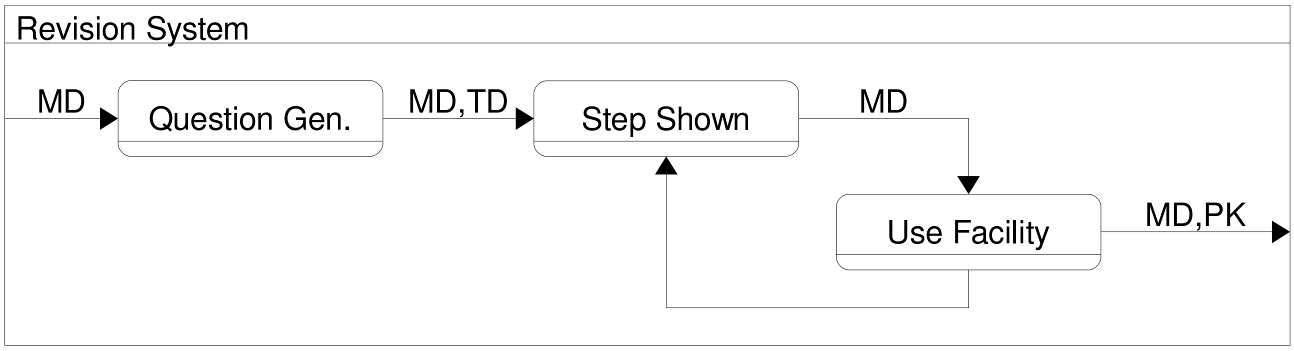


The revision system can be broken into two separate groups. First a topic is chosen which triggers a question to be generated which leads to the solution being shown step by step via user input. If there is still a misunderstood topic the system can be repeated by looping the Revision System until the misunderstood data has diminished.

Level 2



By extracting the struggling topics from misunderstood data the user can select which topic they want to work on. After a topic has been selected, topic data is created which is then passed to the next process. This is used to give a difficulty range, based on how confident and which tier they are taking will influence on which difficulty setting is chosen.



The topic data from the select topic procedure is used to produce a valid question that complies with what the user specified. Then the user is presented with a step by step overview of how to solve the generated question. The use facility process is the use of another system feature such as the unit converter or the infix evaluator. Once all steps have been shown, hopefully most of the misunderstood data is now well understood.

Data Dictionary

Name	*Data type*
Input expression	Text
Expression result	Number
Variable name	Text
Variable value	Number
Graph expression	Text
Graph rendering	Image
Number generator start value	Number
Number generator end value	Number
Number generator patterns	List of text
Number generator result	List of numbers
Unit converter unit type	List of text
Unit converter input number	Number
Unit converter result	Number
Revision select topic	List of text from tree
Revision select difficulty	List of text
Revision miscellaneous	Number/text/image

Data Volumes

Not appropriate.

Objectives of the System

The objectives are compromised from the interview with Rob Lamb and the questionnaires by merging them into one group so that the system has a unified direction to the problem it is going to solve.

1 Revising a topic

1.1 Wide variety of maths topics available to the user

1.2 Users can set the difficultly of a topic

- 1.3 Users are guided though topic using concise presentation
- 1.4 Questions are generated by computer
- 1.5 Users are able to be shown the solution

2 Extra features

- 2.1 Expression evaluator
- 2.2 Dynamic user defined variables
- 2.3 Ability to draw graphs
- 2.4 Unit conversion utility
- 2.5 Number generator to understand patterns

A prototype was created based on these objectives with focusing on primary functionality, and then it was presented to Rob along with the new objectives. He used and distributed the prototype to a small number of students who then evaluated the pros and cons and got back to Rob with their opinions. Rob replied with a letter marking changes to the objectives so that it can accurately reflect student's needs and correct problems that users were having. Here are the final objectives that the new proposed system will follow.

General

The system will allow students of all learning types to advance and improve in their maths tests. This will be achieved by the use of a graphical and descriptive method of portraying data in a clear concise format.

Specific

1 Revising a topic

- 1.1 Relaxed calm environment for users
- 1.2 Clean graphical user interface
- 1.3 Quick and responsive program
- 1.4 Variety of maths topics accessible to the user
- 1.5 Users can set difficultly of a topic
- 1.6 Guided though topic using concise presentation
- 1.7 Questions are generated by computer

1.8 Users are able to be shown the solution

2 Extra features

2.1 Expression evaluator

2.1.1 Complies with BIDMAS

2.1.2 Supports parenthesis, +, -, *, /, ^ operators

2.1.3 Computes quickly

2.1.4 Past calculations are stored in history

2.1.5 Dynamic user defined variables

2.2 Users can draw graphs

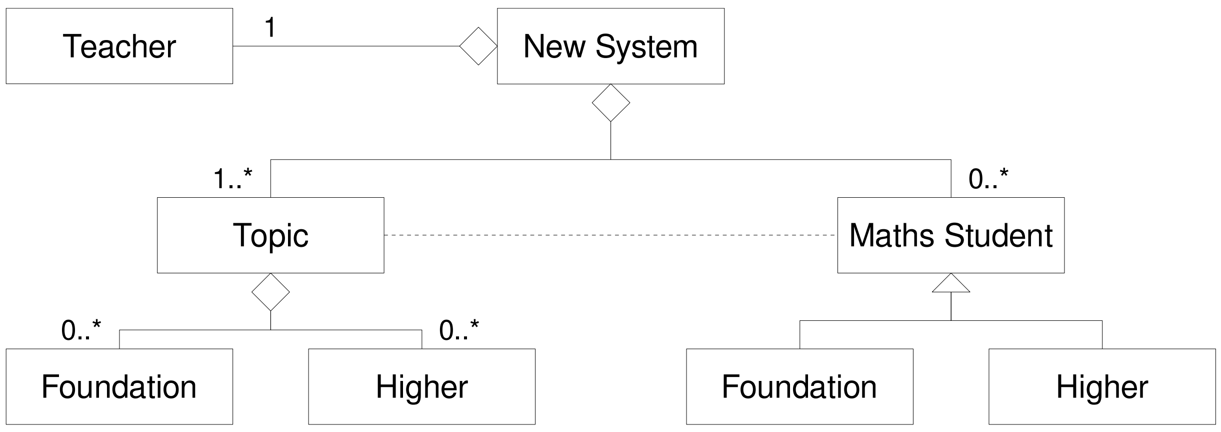
2.3 Unit converter to help understand units

2.4 Number generator to clarify patterns

2.5 Portable program

2.6 Platform-independent

Object Analysis Diagrams



The system will have two types of students; they can be foundation or higher tier to comply with the options of entry in GCSE maths. This is needed because each type of student will need questions tailored towards the exam they will be sitting. This will enable them to practice questions that are similar to the ones that will be on the exam which better prepare users for the big day. Similarly, there will need to be two levels of difficulty for each topic to match what students will be tested on. Both the higher and foundation topics will be available to users giving them the choice to which one they will use, putting them in control of their learning. There is an association link between a maths student and a topic because although they are not related, a maths student uses a topic. Finally, each class has one teacher which is still part of the system because they were kept from the existing system because they are still a vital part.

Constraints

The solution will be subject to hardware, software and time constraints and must be tailored to the computing skill of prospective users. Rob Lamb explained in the interview that his students have a greater understanding of computers that himself. Although his use of the solution will be small because it will be designed for the students, it must still be suitable for low ability users to make it usable by all potential candidates. However, the solution presume users have a basic knowledge of how to operate standard peripherals, such as a mouse and keyboard. This is based on that most students acquire a Key Skill qualification which has an Information technology section showing that users will have the ability to operate the system.

The solution will have no dependencies which will help to make it portable, and if users wish they may install it on their computer at home. Users need Linux, Mac OS X or Windows operating system in order to execute the respective program for each operating system. The Free Pascal compiler will be used and because it is a multi-platform technology that can used to compile for potentially more operating systems such as FreeBSD and many hardware architectures such as x86-64 and ARM.

Hardware requirements are very low, but the minimum recommended requirements are: 200Mhz CPU, 16MB RAM, 2MB disk space, 800 x 600 desktop resolution and standard peripherals e.g. mouse, keyboard and monitor.

Time constraints are an issue and as a result only the fundamental features of the solution will be developed, if successful more will be added later to offer users a more versatile program.

Limitations

The first version of the solution will have the basic, essential functionality because the current model allows the potential to add more abilities easily because of a proposed modular design. As the GSCE syllabus has a lot of content there will be a great time spend adding modules and testing them, so only topics that are not well understood in general should be implemented first. As a result only simultaneous equations will be available to the user in the early stages of development but more can obviously be added later.

Users will have full access to the program and there will not be any user accounts. However, in the future teachers could become users and have their own area where they can print questions and set them for homework or even formulate tests. In order to achieve this user accounts will need to be set up which should not be a difficult task. The system may generate questions as well as the solutions but students will not have permissions or access to the answers. This could lead to multi-user system in which users can login to, and by doing so would separate personal data and allow a calendar to plan revision or record statistics on current progress, as some examples. Due to time constraints these features will not be implemented but may be considered in the future based on how well received the system is by respective users. This shows that although the system is only a sample of what it could become, some students may still prefer the traditional pen and paper revising method.

Consideration of Alternative Solutions

The current system in place uses a set of word processed questions that are printed and given to students. Later the student returns their solutions to the teacher by a set deadline which then the results are marked and promptly returned to the student. If a student has any problems then they can see the teacher after lessons and discuss any problems they are having. However, problems with this system include extensive amounts of time required outside lessons, especially when many students are struggling and marking over thirty papers per class is a burden on teachers. As the questions and solutions are available on a shared device linked to all college computers, some students could take advantage of this and cheat on assignments, to prevent this occurrence the answers are not available to students. Although Rob Lamb can release permissions when an assignment is complete but this is very time consuming to do for every user and not all students use it for their revision to determine were they went wrong. Although if a group of students are having similar problems then these can be handled in a single session with Rob. But this removes tailored discussions relevant to specific students because they will have individual problems and a large after college session will be very general in terms of the problem at hand.

One solution could be to create a program that could generate questions, print and store the questions and solutions. Then students could answer the questions on paper and then hand the solutions in. This would be a more automated version of the current system. The next stage would be for the teacher to mark the paper and hand it back with the score, or the system could use optical character recognition to make to process easier. There could be several ways in which the questions could be presented; one would allow there to be a suitable space for students to show working in addition to space for the final answer. Or there could be a multiple choice system, in which students will be given maybe five options to choose from and after calculating the answer it will select the most appropriate one. These two methods have some advantages and disadvantages; the initial marking system proposal will be tedious for the teacher but will allow mistakes and misunderstandings of the student to be identified. This allows students to find out where they made a mistake from the teacher and will help clarify as well as explain the issue so that they will improve in the future as well as enable to use it to revise from later. The multiple choice questions would contrast with the previous marking system because of the time and effort required to use would be much less. The system could even be automated by the use of a scanner, which could collect images of the paper and an algorithm could detect the chosen answer. Then the marks would be worked out and then presented to candidates by printing the result on the answer sheet.

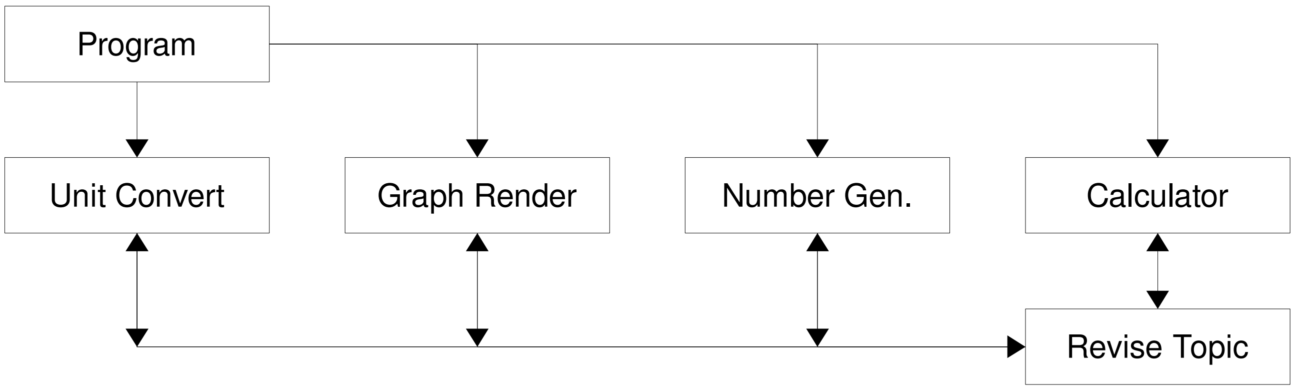
A third method to solve the problem will be to create a digital version of the previous alternative solution. This will include a web based service in which students will have a user-name and password in which to login to the system. This is needed because statistics may be recorded to give students a clear idea of which topics need extra work on and to determine others the user is good at. The login aspects will all separate permissions among users separating personal data and abstracting the complexities of the system to users. The service will allow users to learn new topics by the using a step by step walk-through of how to solve questions. Also, teachers will have access to student accounts and will offer help to students who need it as well as track their progress. Apart from learning new topics the service can generate new questions to help retest a student on any given topic to make sure they are confident and able at a particular topic.

Justification of Chosen Solution

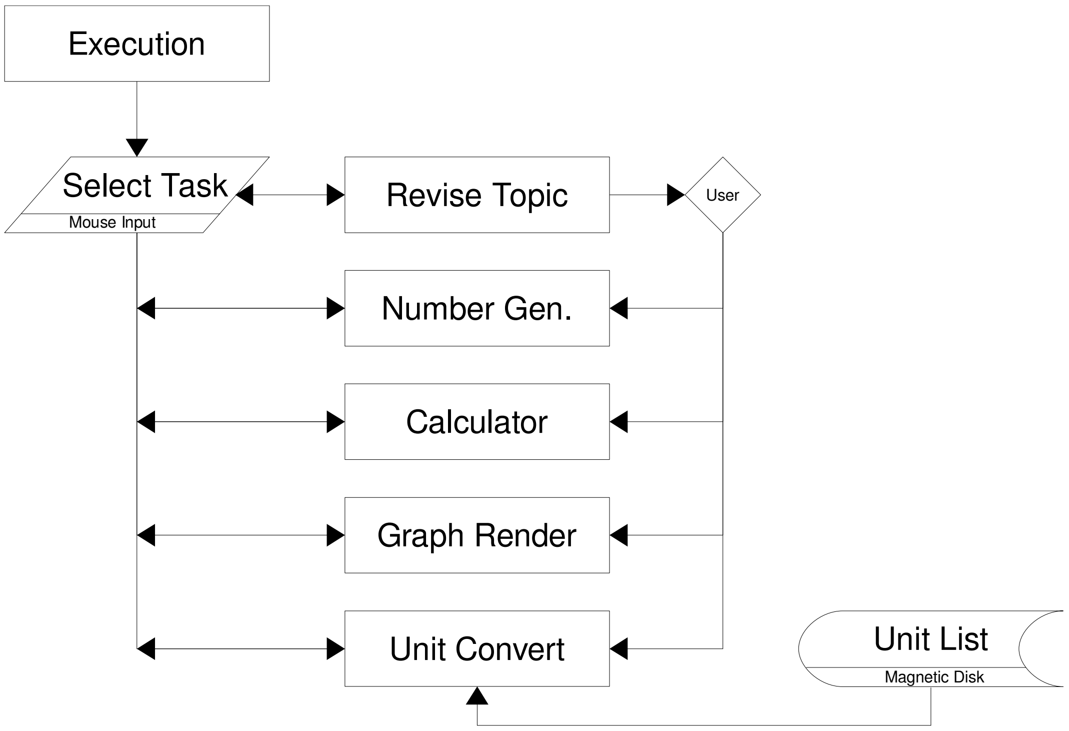
The best method is to use the best aspects from all considered solutions; this will ensure that the best and most appropriate solution is developed. This has lead to the decision to dismiss a web based service in favour of a portable executable; this is because a small number of students that are outside the questionnaire sample size may not have an internet question and is part of the system objectives. Unfortunately some students may not own a computer so they cannot use the system at home; therefore it is necessary to have the system installed on all machines at college so it can be used by any student any time. As the program will be portable by design it can be copied to a flash drive or any other storage medium meaning that it can be distributed to students by CD, e-mail or memory stick. This will allow students to use the software at home or public library in addition to the copy on college computers. Although the software may be executed from a read-only medium e.g. CD, this would hinder some functionality making the program less effective, but as portable devices are inexpensive this should not be an issue. To satisfy the needs of the system, the integrated development environment Lazarus will be used because it is free and open source software, which will cut costs. Plus, the IDE uses the Free Pascal compiler which handles object-oriented programming and is able to compile for many operating systems making the system accessible to a wide range of computer systems. A web based system will consist of a PHP server and will need to run continuously which will take up a considerable amount of time for maintenance, were time resources may be better used for other tasks. Heavy administration will be an issue because each student will need there own account and permissions, additional types of accounts will need to be developed with the most significant being the teacher and administrator. Meanwhile a portable executable will not have these problems because the system will act like an ordinary piece of software that will run locally. If the server goes down then students will not be able to use the revision system because it is entirely dependant on it which could be a massive issue.

Design

Outline System Design

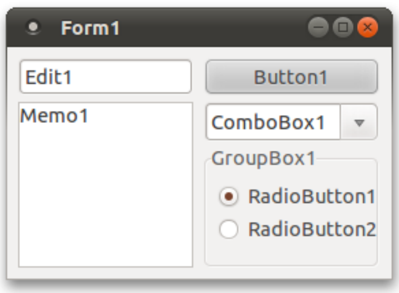


System Flow Chart

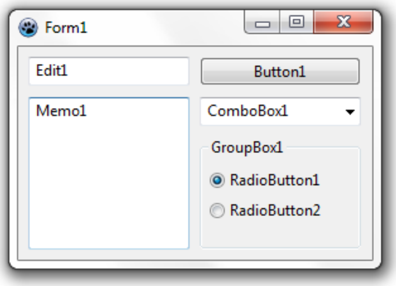


User Interface Designs

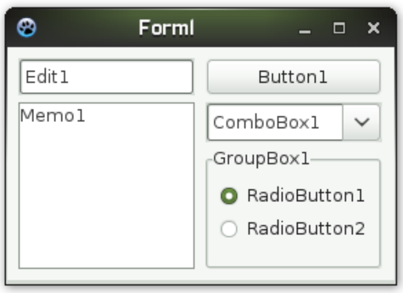
The graphical user interface will use the default operating system theme, this will help in making it cross-platform and will eliminate contrasting colour schemes with the operating system. It will also be advantageous that development time will be reduced as time will not be consumed on tasks such as changing colour. However, there maybe increased testing required to determine whether the GUI is renders correctly on all operating systems.



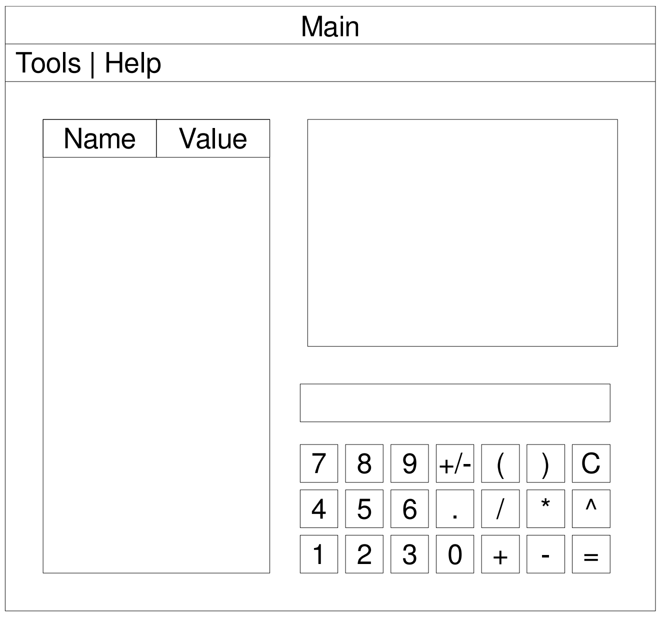
Ubuntu 10.10



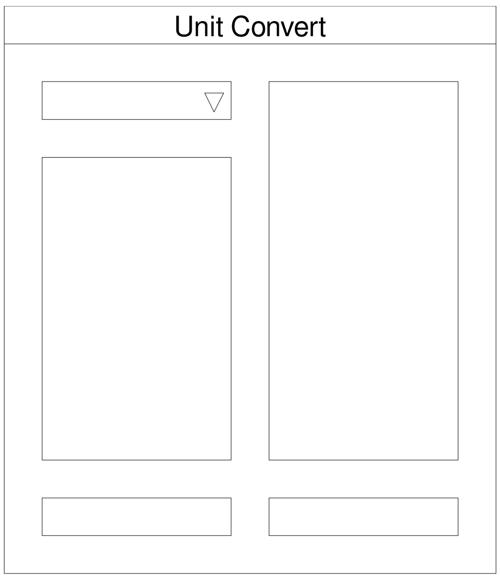
Windows 7



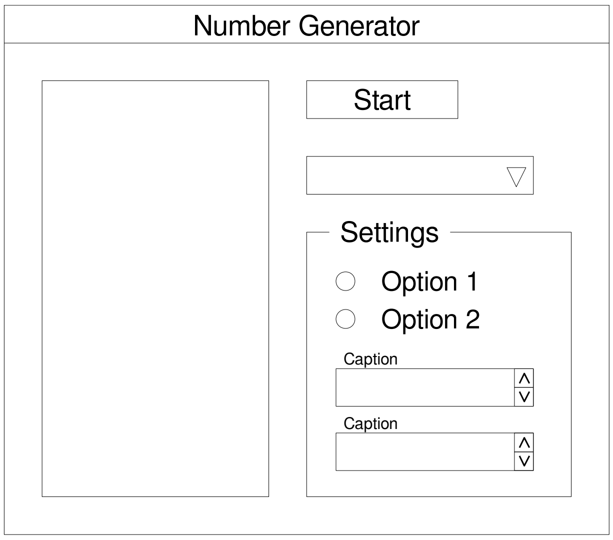
OpenSUSE 11.3



This is the main window, the main menu has the item tools where the user can access the other forms such as unit convert. The object on the left is a string grid where users can define there own variables. Moving to the top right box is a memo where the equation history is stored. Below is an edit box that accepts equations. Meanwhile below again is an array of buttons that allow users to enter data into the edit box. Keyboard input is still accepted in the edit box.

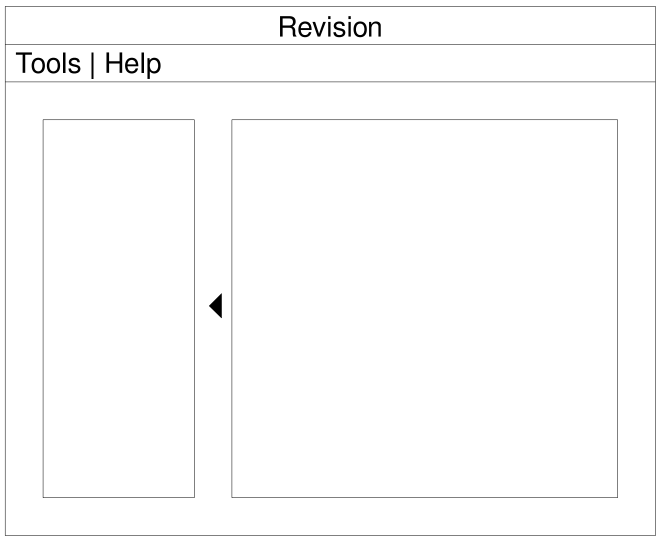


Here is the unit convert form, the top left object is a combo-box where the user has the ability to select which unit type they want to convert. Below is a list-box where based on the selection of the combo-box determines the contents. When the user selects a unit from this object all available units are displayed in the list-box to the right. At the bottom there are two edit boxes, this is where the user may enter a value that will be converted and displayed in the alternate box. The edit-boxes only accept numbers and one decimal point; this is to eliminate errors in conversion.



To the to far right is a list-box where generated values are displayed. Moving to the top right is a button that starts the number generation process. Below is a combo-box that allows users to choose which number pattern they want to be displayed. Below once more is a group-box that allows users to change the generation settings via two radio buttons and another two spin-edits with captions that explain what each of them represent.

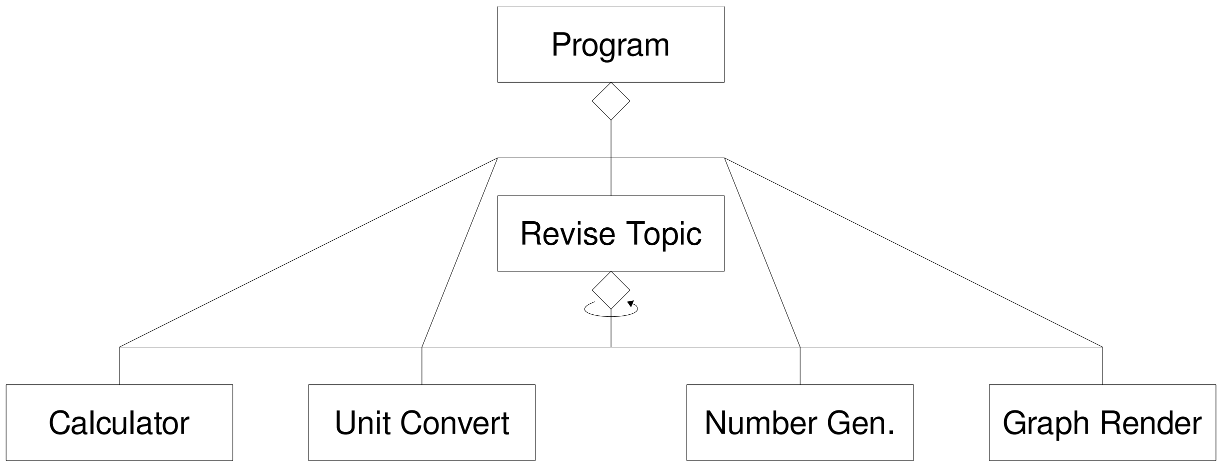
The large box on the form is a paintbox where the graphs will be rendered. To give a sense of scale axis will also be drawn. At the bottom in an edit-box, this is where user may enter an expression that is to be displayed.



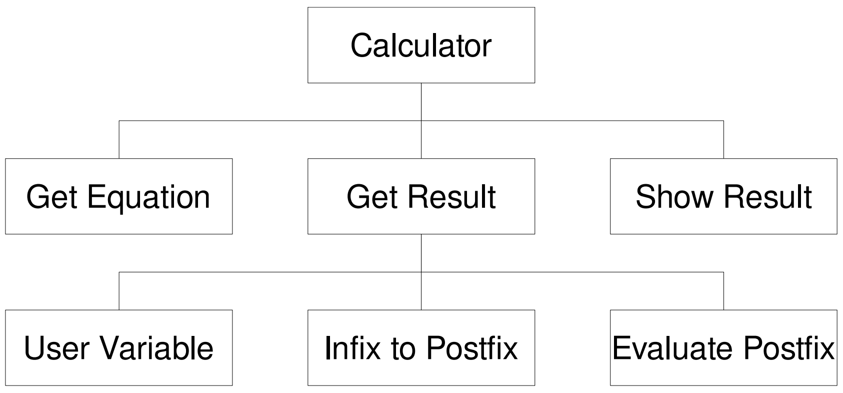
Like the main form the revision form has the same main menu, which will allow users to utilize the other features of the program. The left box houses a list of GCSE maths topics grouped into large sections and subdivided into specific topics in a tree view. The black arrow can show or hide the list to give more space for the right area and make full use of screen space. The right area provides users with a visual method of teaching and testing topics based on which one is chosen by the user.

Program Structure

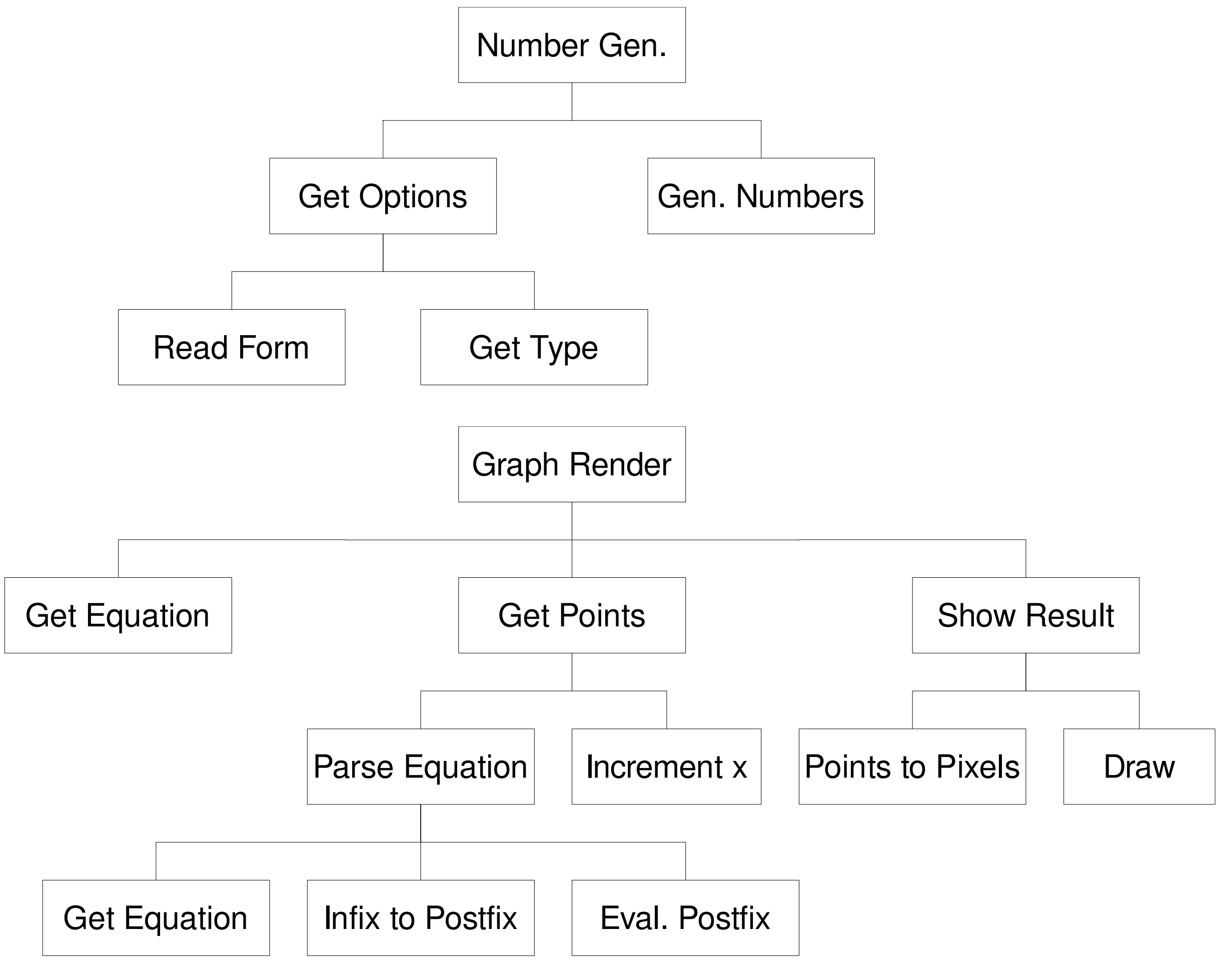
High Level Hierarchy

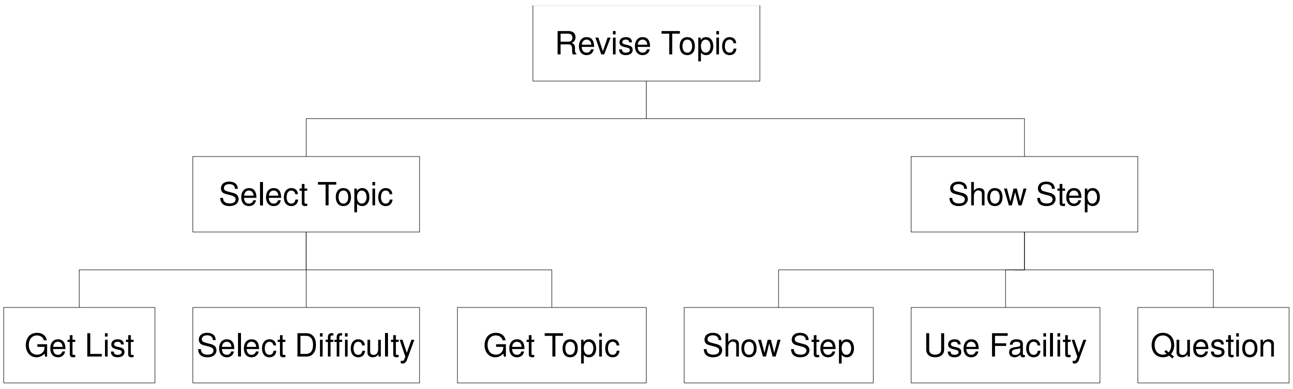


Low Level Hierarchy



In order to parse an expression it must first be converted to postfix notation (infix to postfix) and then the postfix should be evaluated (evaluate postfix), this is a modification of the shunting-yard algorithm working with the postfix algorithm.


Number Gen.
Get Options
Gen. Numbers
Read Form
Get Type
Graph Render
Get Equation
Get Points
Show Result
Parse Equation
Increment x
Points to Pixels
Draw
Get Equation
Infix to Postfix
Eval. Postfix



Design Data Dictionary

Main Form	
Name	Data Type
Input expression	String
Result	String (from float)
Variable name	String
Variable value	String (to float)

Unit Convert	
Name	Data Type
Input value	String (to float)
Result value	String (from float)
Unit type	Integer
Unit selection	Integer
Original unit	Integer

Number Generator	
Name	Data Type
Pattern	Integer
Option type	Boolean
Option 1	String (to integer)
Option 2	String (to integer)

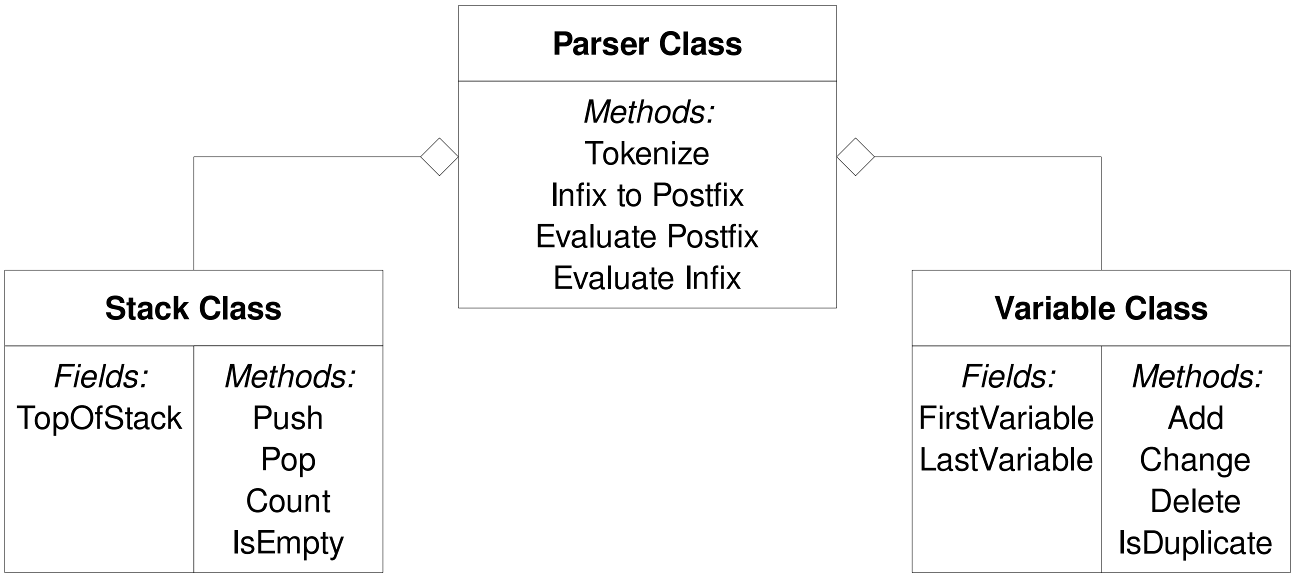
Graph Render	
Name	Data Type
Input expression	String
Graph	Bitmap

The revision form is not mentioned because it is subjective to what topic a user is studying, but will follow similar guidelines as other forms.

File Organisation

This section is not applicable because the new system will be a self contained executable that will not have any dependencies or require additional external data.

Object Class Diagrams and Definitions

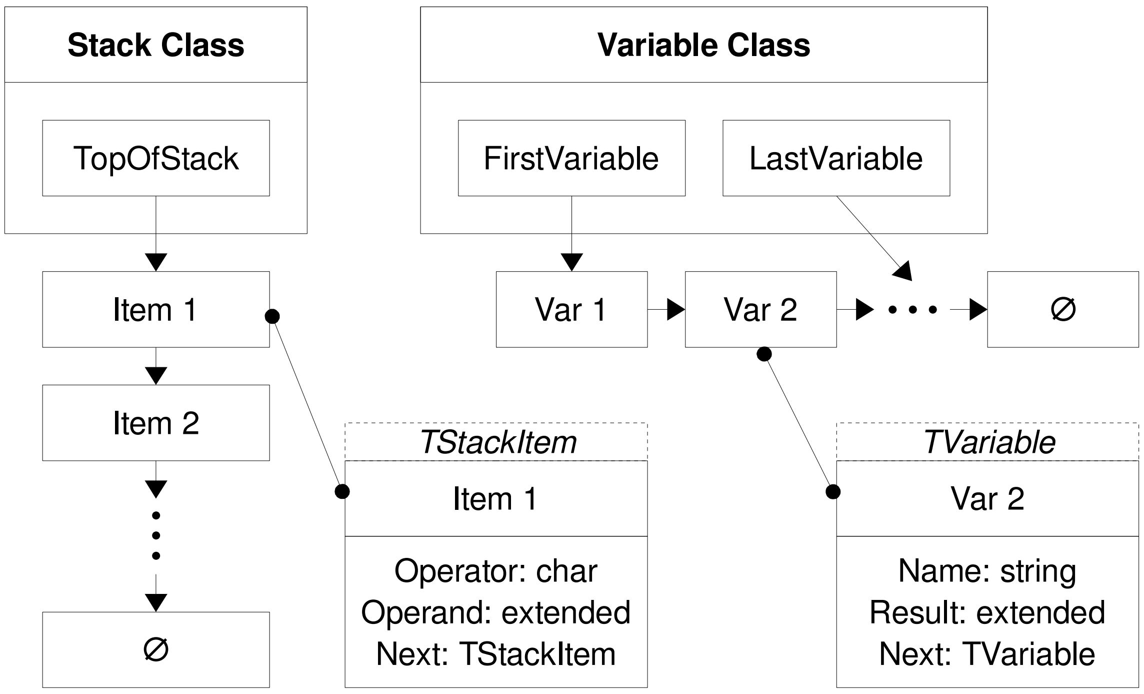


Parser Class

This class has two main aggregated components which are the stack and variable classes, these are needed for the functionality of the infix to postfix and evaluate postfix functions. The evaluate infix function simply calls the previous two that were mentioned so that some complexities of the class can be hidden away which will make it easier to use. The stack is needed by all functions of the parser, it is instantiated and freed from memory at the same time as the parser class itself, including the variable class as well. Both the infix to postfix and evaluate postfix algorithms are stack based hence one will need to be implemented in order for these functions to work. The variable class is needed to replace variable names with their corresponding value at the tokenize stage in preparation for infix to postfix conversion.

Stack and Variable Classes

Both of these classes are similar and it would be logical to create them by inheriting common methods from a single class. This is not possible because they are not similar enough to justify this design choice because the stack class is a dynamic stack implementation using linked list. When an item is pushed it is set to point to the top of the stack, then TopOfStack points to the new item. The variable class handles a linked list, when a variable is added, the last item points to the new item and so does LastVaraible. Meanwhile, when an item from the stack is removed only the top item is popped of, the variable class allows any item to be removed while still maintaining the data structure.



The stack class has a polymorphic push method because an operator or operand must be able to be pushed to the stack. This requires two overloaded declarations of the procedure, one with a char parameter and another with an extended data type.

Algorithms

Tokenize

```
Functional Tokenize(Expression): string
  Remove Spaces
  Count minuses
  Search for variable names, replace with value
  if there are multiple minuses, simplify to single operator
  if number is minus place brackets around it
EndFunction
```

This function takes the expression and replaces variable names with the value that it represents; this allows the user defined variables to be incorporated into an entered expression. It also removes spaces and double minuses to help simply the expression in preparation for the infix to postfix algorithm.

Infix to Postfix

```
Function InfixToPostfix(Expression): string
```

```
  Operators←^,/,*,+,-
  Operands←0..9,.
  i←0
  while i=Length of Expression
    i←i+1
    Case Expression Char i
      Operands: While (Char in Operands)
                  Add Char to Postfix
                EndWhile
      (: Push ( to Stack
      ): Pop Stack to Postfix until StackTop is (
      Operators: While StackProcedance<=OperatorProcedance
                   Pop Stack to Postfix
                 EndWhile
                 Add Char to Postfix
    EndCase
  EndWhile
  Repeat
    Pop Stack to Postfix
  Until StackIsEmpty
  Result←Postfix
EndFunction
```

First the operators and operands are defined to set-up the start of the algorithm. Each character starting from the leftmost is determined to be an operator or operand, if it is an operand then it is added to the result of the expression meanwhile an operator is pushed to the stack if it is empty. If there are other items on the stack then the precedence is compared to the expression precedence based on the order of the operators with the left most having the highest precedence, if the stack is less than or equal than the current evaluating character then it is added to the postfix result and popped of the stack. This is repeated until the stack is empty or the operator precedence is greater than the stack in which the character is pushed onto the stack. When the end of the expression is reached then all values are popped off the stack to ensure that all operators have been used and so that it can be used later by another function.

Evaluate Postfix

```
Function EvalPostfix(Postfix): float
  Operators←^,/,*,+,-
  Operands←0..9,.
  i←0
  While i<= Length of Expression
    i←i+1
    if Postfix Char i in Operands Push Postfix[i] to Stack
    else Case Postfix Char i
           +: Push (Pop Stack + Pop Stack) to Stack
           -: Push (Pop Stack - Pop Stack) to Stack
           *: Push (Pop Stack * Pop Stack) to Stack
           /: Push (Pop Stack / Pop Stack) to Stack
           ^: Push (Pop Stack ^ Pop Stack) to Stack
         EndCase
  EndWhile
  Pop Stack to EvalPostfix
  While not StackIsEmpty Pop Stack
  Result←EvalPostfix
EndFunction
```

Firstly the expression is scanned left to right to find and separate operators and operands. When an operator is found the two previous items on the stack are popped and are manipulated based on the operator, such as addition. The result is then pushed onto the stack. When the expression has been scanned the result will be the only item left on the stack unless there is an error in the postfix string. The final item is popped and is made the result of the function followed by the popping any remaining items so that it can be used later.

Push Item to Stack

```
Procedure PushToStack(Item)
  New Stack Item
  TopOfStack←NewItem
  NewItem^.Item←Item
EndProcedure
```

The pointer to the top of the stack is stored in the stack class, a new item is dynamically created and then the top of the stack then points to the newly created item.

Pop Stack

```
Procedure PopStack
  TempItem←TopOfStack
  TopOfStack←TopOfStack^NextItem
  Delete TempItem
EndProcedure
```

A pointer is created and is set to point to the second item on the stack, if there is only one item then it is nil. The item on top of the stack is freed from the heap and then the top of the stack becomes the temporary item.

Add Variable

```
Procedure AddVariable(Name,Value)
  if Name is duplicate then exit procedure
  New Variable Item
  NewVariable^Name←Name
  NewVariable^Value←Value
  NewVariable^NextItem←nil
  if FirstVariable=nil then
    FirstVariable=NewVariable
    LastVariable←NewVariable
  else
    LastVariable^NextItem←NewVariable
    LastVariable←LastVariable^NextItem
  endif
EndProcedure
```

Firstly there is a call to a function that checks to see is there another variable with the same name specified by the Name parameter. If it returns true then the procedure is exited, if not then a new variable is created and its properties are defined. Then it is added to the variable linked list.

Change Variable

```
Procedure ChangeVariable(Name,Value)
  CurrentItem←FirstItem
  while CurrentItem<>nil
    if CurrentItem^Name=Name then
      CurrentItem^Value←Value
      Exit procedure
    Endif
    CurrentItem←CurrentItem^NextItem
  EndWhile
EndProcedure
```

A temporary pointer is used to keep track which item the procedure is currently observing, the next item is reached by the current item becoming the item it points to. Each item is then checked it has the name that it is looking for; if true then the value of the current item is changes. If current item becomes false then the end current item is the last item and the procedure ends.

Delete Variable

```
Procedure DelVariable(Name)
  CurrentItem←FirstItem
  while CurrentItem<>nil
    if CurrentItem^Name=Name then
      PreviousItem^NextItem←CurrentItem^NextItem
      Delete CurrentItem
      Exit procedure
    Endif
    CurrentItem←CurrentItem^NextItem
  EndWhile
EndProcedure
```

A temporary pointer is used to traverse through the linked list, if the current item has the same name as the name passed as a parameter then the previous item pointer becomes the same as the current item pointer so that items are not lost to the heap and to preserve the linked list chain. The last stage involves removing the current variable from memory because it is no longer needed and is the item that was set to be removed. If current becomes nil then the end of the list is reached and the execution of the procedure ceases with no change because the value does not exist.

Security and Integrity of Data

In the current design security of low importance because it will not contain any personal information and none are written to disk in the early version. As the system will be constructed using the object-oriented paradigm this will provide two layers of abstraction. One high level layer as a GUI for the end user while low level strict data handling and processing by the use of classes from the prospective of the programmer.

Preliminary Test Plan

The plan it set out into two separate stages with the first one looking at the system from a high level, the users view point and the second delving deeper at low level from the point of the programmer. The first stage will utilise a top-down approach in order to make sure all variations of user input are handled correctly and produce the correct results. As bottom-up testing can be very tedious creating numerous situations that pass though all levels of code, the decision has been made that only such testing will occur when a test fails the initial stage will it be considered for stage two. The second stage will involve a trace table and dry run using the test data that failed in the previous stage to identify where the bug exists in the code. This will help fix the error and prevent it from occurring again and as a result will improve the reliability and stability of the program.

Detailed Test Plan

Test	*Purpose*	*Description*	*Test Data*	*Expected Value*
1.1	Basic addition test	Addition	9+1	10
1.2	Advanced addition test	Double addition	5+6+9	20
1.3	Complex addition test	Multi addition	2+8+5+6	21
1.4	Negation test	Addition of negative numbers	9+-5	4
1.5	Complex negation	Complex addition of negative numbers	9+---8+-6+--4	-1
1.6	Basic subtraction test	Subtraction	8-6	2

Test	*Purpose*	*Description*	*Test Data*	*Expected Value*
1.7	Advanced subtraction test	Double subtraction	1-5-7	-11
1.8	Complex subtraction test	Multi subtraction	5-9-8-4	-16
1.9	Negation test	Subtraction of negative numbers	-5--7	2
1.10	Complex negation	Complex subtraction of negative numbers	--2---5--7----4	8
1.11	Basic multiplication test	Multiplication	2*6	12
1.12	Advanced multiplication test	Double multiplication	2*7*3	42
1.13	Complex multiplication test	Multi multiplication	5*8*6*4	960
1.14	Negation test	Multiplication of negative numbers	-5*-7	35
1.15	Complex negation	Complex multiplication of negative numbers	-4*--5*---4*----2	160
1.16	Basic division test	Division	9/3	3
1.17	Advanced division test	Double division	8/4/2	1
1.18	Complex division test	Multi division	9/3/4/2	0.375
1.19	Negation test	Division of negative numbers	-4/-5	0.8
1.20	Complex negation	Complex division of negative numbers	--9/---2/----4/-6	0.1875
1.21	Basic power test	Power	6^4	1296

Test	*Purpose*	*Description*	*Test Data*	*Expected Value*
1.22	Advanced power test	Double power	3^4^2	43046721
1.23	Complex power test	Multi power	2^9^1^3	512
1.24	Negation test	Powers from negative numbers	-5^-2	0.04
1.25	Complex negation	Complex powers from negative numbers	--2^---1^-3^----2	0.5
1.26	Basic composite test	Addition and minus	5+-2	3
1.27	Advanced composite test	Double addition and minus	5-6+2-7	-6
1.28	Complex composite test	Multi Addition and minus	1+6-5-4+8-2	4
1.29	Basic composite test	Addition and multiplication	2+6*4	26
1.30	Advanced composite test	Double addition and multiplication	6*5+9*7	93
1.31	Complex composite test	Multi addition and multiplication	2+7*5+4*2*3	61
1.32	Basic composite test	Addition and division	2/4+1	1.5
1.33	Advanced composite test	Double addition and division	2/8+4/2	2.25
1.34	Complex composite test	Multi addition and division	5/2+6/4/2+5	8.25
1.35	Basic composite test	Addition and powers	4^6+3	4099
1.36	Advanced composite test	Double addition and powers	8+9^3+7	744

Test	*Purpose*	*Description*	*Test Data*	*Expected Value*
1.37	Complex composite test	Multi addition and powers	6^4+3^2^3+9	8.10 *10^24
1.38	Basic composite test	Minus and multiplication	5-9*3	-22
1.39	Advanced composite test	Double minus and multiplication	6-7*4-8	-30
1.40	Complex composite test	Multi minus and multiplication	7-5*4*8-9*2	-171
1.41	Basic composite test	Minus and division	8/4-3	-1
1.42	Advanced composite test	Double minus and division	9/4-1/2	1.75
1.43	Complex composite test	Multi minus and division	7-4/5-2/8-6	-0.05
1.44	Basic composite test	Minus and powers	3^5-8	235
1.45	Advanced composite test	Double minus and powers	4-6^2-3	-35
1.46	Complex composite test	Multi minus and powers	6-5^7-5^2-8	-78152
1.47	Basic composite test	Multiplication and division	7/3*6	14
1.48	Advanced composite test	Double multiplication and division	1/9*4/3	0.148
1.49	Complex composite test	Multi multiplication and division	4*5/8*2/9*4	2.22
1.50	Basic composite test	Multiplication and powers	4^4*5	1280
1.51	Advanced composite test	Double multiplication and powers	5*6^4*8	51840

Test	*Purpose*	*Description*	*Test Data*	*Expected Value*
1.52	Complex composite test	Multi multiplication and powers	3^2*6^3*7*2	27216
1.53	Basic composite test	Division and powers	7^3/8	42.875
1.54	Advanced composite test	Double division and powers	6^4/5^2	51.84
1.55	Complex composite test	Multi division and powers	3^9/4/3^2/1	546.75
1.56	Basic brackets test	Brackets	(2+6)*8	64
1.57	Advanced brackets test	Nested brackets	((2+6)*8)-5	59
1.58	Complex brackets test	Multi nested brackets	((2+6)*(8-5))+8	32
1.59	Simple bracket test	Two single brackets	(8+7)*(8/3)	40
1.60	Hard bracket test	Single and nested brackets	(8+7)*((8/3)/4)	10
1.61	Tough bracket test	Single and multi nested brackets	(8+7)*((8/3)*(8+1)-5)	285
1.62	Very complex bracket test	Nested and multi nested brackets	((8/3)*(8+1)-5)*((9/3)*(9+2)-7)	494
1.63	Normal data	Test of nil input	0	0
1.64	Normal data	Use of all operators	((7*4)+-5/2-9)^2	272.25
1.65	Normal data	Use of decimal	2.5*3	7
1.66	Normal data	Use of negative decimal	-5.9+6.2	0.3
1.67	Normal data	Standard test	0/1	0
1.68	Error catching	Exception	1/0	Error
1.69	Error catching	Missing operand	5*8/	Error

Test	*Purpose*	*Description*	*Test Data*	*Expected Value*
1.70	Error catching	Missing operand	4)*(6+4	Error
1.71	Error catching	Missing operator	5+*7	Error
1.72	Error catching	Missing operator	3*/9+4	Error
1.73	Error catching	Duplicate operand	2*5++3	Error
1.74	Error catching	Duplicate operand	4*9//4	Error
1.75	Exceptional data	Exception	Test data	Error
1.76	Erroneous data	Non-existent variable	25*temp	Error
1.77	Extreme data	Test extended data type range	-3.6*10^-4951	-3.6*E-4951
1.78	Extreme data	Test extended data type range	1.1*10^4932	1.1*10E4932
1.79	Boundary data	Data is above range	1.1*10^4933	Error
1.80	Boundary data	Data is below range	-3.6 * 10^-4952	Error

Testing

Test Strategies

Bottom-up testing looks at the lowest level of modules testing each routine to find and detect defects. Top-down testing is the process of finding bugs from the users point of view, also known as interface or stub testing. This method involves inputting data and then determining where the output is correct. Dry running, also known as a walk-through, tracing or desk checking involves reading through code and checked by hand. A trace table is completed to check values of variables at every stage of execution to reveal the causes of errors. White box tests are derived from the knowledge of the program code. Then test cases are devised that every possible route through the code is tested. Black box tests are derived from the program specification, cases are devised from inputs and checking where the output are correct. To that the program produces correct the results, it will be checked against a scientific calculator, which will be assumed to be a reliable.

Testing will begin with a black box, top-down approach to find any errors, if any occur then a dry run will commence with the input that provided the invalid output so that the problem can to found and eliminated. All evidence is available in the appendix.

Test Plan

The most important features will be tested first.

Test Series 1

- Test that all expression solutions are correct

- Test operators +, -, *, /, ^, (,)
- Make sure user variables are working correctly

Test Series 2

- If error is found then start trace table to find cause
- Make note of problem in system manual

Test Data

Test and Evidence	*Purpose*	*Description*	*Test Data*	*Returned Value*	*Correct Value*
1.1	Basic addition test	Addition	9+1	10	10
1.2	Advanced addition test	Double addition	5+6+9	20	20
1.3	Complex addition test	Multi addition	2+8+5+6	21	21
1.4	Negation test	Addition of negative numbers	9+-5	4	4
1.5	Complex negation	Complex addition of negative numbers	9+---8+-6+--4	-1	-1
1.6	Basic subtraction test	Subtraction	8-6	2	2
1.7	Advanced subtraction test	Double subtraction	1-5-7	-11	-11

Test and Evidence	*Purpose*	*Description*	*Test Data*	*Returned Value*	*Correct Value*
1.8	Complex subtraction test	Multi subtraction	5-9-8-4	-16	-16
1.9	Negation test	Subtraction of negative numbers	-5--7	2	2
1.10	Complex negation	Complex subtraction of negative numbers	--2---5--7----4	8	8
1.11	Basic multiplication test	Multiplication	2*6	12	12
1.12	Advanced multiplication test	Double multiplication	2*7*3	42	42
1.13	Complex multiplication test	Multi multiplication	5*8*6*4	960	960
1.14	Negation test	Multiplication of negative numbers	-5*-7	35	35
1.15	Complex negation	Complex multiplication of negative numbers	-4*--5*---4*----2	160	160
1.16	Basic division test	Division	9/3	3	3
1.17	Advanced division test	Double division	8/4/2	1	1
1.18	Complex division test	Multi division	9/3/4/2	0.375	0.375
1.19	Negation test	Division of negative numbers	-4/-5	0.8	0.8

Test and Evidence	*Purpose*	*Description*	*Test Data*	*Returned Value*	*Correct Value*
1.20	Complex negation	Complex division of negative numbers	--9/---2/----4/-6	0.1875	0.1875
1.21	Basic power test	Power	6^4	1296	1296
1.22	Advanced power test	Double power	3^4^2	6561	43046721
1.23	Complex power test	Multi power	2^9^1^3	512	512
1.24	Negation test	Powers from negative numbers	-5^-2	0.04	0.04
1.25	Complex negation	Complex powers from negative numbers	--2^---1^-3^----2	0.5	0.5
1.26	Basic composite test	Addition and minus	5+-2	3	3
1.27	Advanced composite test	Double addition and minus	5-6+2-7	-6	-6
1.28	Complex composite test	Multi Addition and minus	1+6-5-4+8-2	4	4
1.29	Basic composite test	Addition and multiplication	2+6*4	26	26
1.30	Advanced composite test	Double addition and multiplication	6*5+9*7	93	93
1.31	Complex composite test	Multi addition and multiplication	2+7*5+4*2*3	61	61
1.32	Basic composite test	Addition and division	2/4+1	1.5	1.5
1.33	Advanced composite test	Double addition and division	2/8+4/2	2.25	2.25

Test and Evidence	*Purpose*	*Description*	*Test Data*	*Returned Value*	*Correct Value*
1.34	Complex composite test	Multi addition and division	5/2+6/4/2+5	8.25	8.25
1.35	Basic composite test	Addition and powers	4^6+3	4099	4099
1.36	Advanced composite test	Double addition and powers	8+9^3+7	744	744
1.37	Complex composite test	Multi addition and powers	6^4+3^2^3+9	7866	8.10 *10^24
1.38	Basic composite test	Minus and multiplication	5-9*3	-22	-22
1.39	Advanced composite test	Double minus and multiplication	6-7*4-8	-30	-30
1.40	Complex composite test	Multi minus and multiplication	7-5*4*8-9*2	-171	-171
1.41	Basic composite test	Minus and division	8/4-3	-1	-1
1.42	Advanced composite test	Double minus and division	9/4-1/2	1.75	1.75
1.43	Complex composite test	Multi minus and division	7-4/5-2/8-6	-0.05	-0.05
1.44	Basic composite test	Minus and powers	3^5-8	235	235
1.45	Advanced composite test	Double minus and powers	4-6^2-3	-35	-35
1.46	Complex composite test	Multi minus and powers	6-5^7-5^2-8	-78152	-78152
1.47	Basic composite test	Multiplication and division	7/3*6	14	14
1.48	Advanced composite test	Double multiplication and division	1/9*4/3	0.148	0.148

Test and Evidence	*Purpose*	*Description*	*Test Data*	*Returned Value*	*Correct Value*
1.49	Complex composite test	Multi multiplication and division	4*5/8*2/9*4	2.22	2.22
1.50	Basic composite test	Multiplication and powers	4^4*5	1280	1280
1.51	Advanced composite test	Double multiplication and powers	5*6^4*8	51840	51840
1.52	Complex composite test	Multi multiplication and powers	3^2*6^3*7*2	27216	27216
1.53	Basic composite test	Division and powers	7^3/8	42.875	42.875
1.54	Advanced composite test	Double division and powers	6^4/5^2	51.84	51.84
1.55	Complex composite test	Multi division and powers	3^9/4/3^2/1	546.75	546.75
1.56	Basic brackets test	Brackets	(2+6)*8	64	64
1.57	Advanced brackets test	Nested brackets	((2+6)*8)-5	59	59
1.58	Complex brackets test	Multi nested brackets	((2+6)*(8-5))+8	32	32
1.59	Simple bracket test	Two single brackets	(8+7)*(8/3)	40	40
1.60	Hard bracket test	Single and nested brackets	(8+7)*((8/3)/4)	10	10
1.61	Tough bracket test	Single and multi nested brackets	(8+7)*((8/3)*(8+1)-5)	285	285
1.62	Very complex bracket test	Nested and multi nested brackets	((8/3)*(8+1)-5)*((9/3)*(9+2)-7)	494	494

Test and Evidence	*Purpose*	*Description*	*Test Data*	*Returned Value*	*Correct Value*
1.63	Normal data	Test of nil input	0	0	0
1.64	Normal data	Use of all operators	((7*4)+-5/2-9)^2	272.25	272.25
1.65	Normal data	Use of decimal	2.5*3	7	7
1.66	Normal data	Use of negative decimal	-5.9+6.2	0.3	0.3
1.67	Normal data	Standard test	0/1	0	0
1.68	Error catching	Exception	1/0	Error	Error
1.69	Error catching	Missing operand	5*8/	Error	Error
1.70	Error catching	Missing operand	4)*(6+4	Error	Error
1.71	Error catching	Missing operator	5+*7	Error	Error
1.72	Error catching	Missing operator	3*/9+4	Error	Error
1.73	Error catching	Duplicate operand	2*5++3	Error	Error
1.74	Error catching	Duplicate operand	4*9//4	Error	Error
1.75	Exceptional data	Exception	Test data	Error	Error
1.76	Erroneous data	Non-existent variable	25*temp	Error	Error
1.77	Extreme data	Test extended data type range	-3.6*10^-4951	-3.6*E-4951	-3.6*E-4951

Test and Evidence	*Purpose*	*Description*	*Test Data*	*Returned Value*	*Correct Value*
1.78	Extreme data	Test extended data type range	1.1*10^4932	1.1*10E4932	1.1*10E4932
1.79	Boundary data	Data is above range	1.1*10^4933	Error	Error
1.80	Boundary data	Data is below range	-3.6 * 10^-4952	Error	Error

Testing Evaluation

All bugs that occurred are related to powers of powers, this is because the algorithm is left-associative while powers are right-associative. An example is 5^{3^2}, inputted as 5^3^2, currently the parser reads this as ((5^3)^2) when it should be interpreted as (5^(3^2)). This will be easy to correct because this behaviour can be added to the infix to postfix algorithm. Only one trace table was completed because all of the test data had a recurring pattern of multiple powers, after the previous problem was solved the tests where run again which all returned correct answers.

Trace table for infix to postfix function using test data 1.37

Expression	*i*	*Stack Precedence*	*Expression Precedence*	*Postfix*	*Stack*
6^4+3^2^3+9	0				
	1			6	
	2				^
	3			6 4	^
	4	2	5	6 4 ^	+

	5			6 4 ^ 3	
	6	5	2	6 4 ^ 3 +	^
	7			6 4 ^ 3 + 2	
	8	2	2		^ ^
	9			6 4 ^ 3 + 2 3	
	10	2	5	6 4 ^ 3 + 2 3 ^ ^	+
	11			6 4 ^ 3 + 2 3 ^ ^ 9 +	

Result from evaluate postfix function is 8.10*10^24 – incorrect.

Trace table for corrected infix to postfix function using test data 1.37

Expression	*i*	*Stack Precedence*	*Expression Precedence*	*Postfix*	*Stack*
6^4+3^2^3+9	0				
	1			6	
	2				^
	3			6 4	
	4	2	5	6 4 ^	+
	5			6 4 ^ 3	
	6	5	2		+ ^

	7			6 4 ^ 3 2	
	8	2	2		+ ^ ^
	9			6 4 ^ 3 2 3	
	10	2	5	6 4 ^ 3 2 3 ^ ^ +	+
	11			6 4 ^ 3 2 3 ^ ^ + 9 +	

Result from evaluate postfix function 7866 – correct.

Test	Value	Previous Answer	New Answer	Correct Answer	Evidence
1.22	3^4^2	6561	43046721	43046721	Appendix 1.81

System Manual

Introduction

MathCalc is a tool that allows users to solve mathematical problems in the form of an infix expression. An additional feature included is a revision system and unit converter, which are arguably among the most important aspects of the program. It is built is a modular fashion to make it robust, reliable and easy to add and maintain features. The program is written in Lazarus which is based of Free Pascal, this makes it easy to compile for many operating systems because the compiler itself is multi-platform. All classes and implementation is available in the appendix.

Mathematical Parser

```
type
  TStringListPointer = ^TStringList;
```

This data type is used by the calculator frame to update the list of user created variables. It must be freed by the procedure that called it because it is not freed itself.

```
type
  ErrorMsg = class(Exception);
```

This is to detect errors and tell the user that there is a problem in the form of a pop-up box; users will have the option to continue or cancel the operation.

```
TVariableItem = ^Item;
Item = record
  VarName: string;
  VarResult: extended;
  Next: TVariableItem;
end;
```

A variable item is a dynamically created record that is part of a linked list to store all user created variables. It contains the name and value of the variable as well as a pointer to the next record. It is encapsulated via the variable class which provides a layer of abstraction in handling and manipulating variable item.

```
TStackItem = ^Node;
Node = record
  AOperator: char;
  AOperand: extended;
  Next: TStackItem;
end;
```

This is used data item that is used by the stack class shown later. It is essential for function of the infix to postfix and postfix evaluation algorithms. Each node of the stack has an operator, operand as well as a pointer to item below on the stack. This allows it to become dynamic which means that users will not be limited by the software, but by hardware.

```
TVariable = class
  private
    FirstVariable: TVariableItem;
    LastVariable: TVariableItem;
    function IsDuplicateVariable(const VarName: string): boolean;
  public
    constructor Create;
    destructor Free;
    procedure ChangeVariable(VarName: string; VarResult: extended);
    function GetVariable(VarName: string): extended;
    procedure AddVariable(VarName: string; VarResult: extended);
    procedure DelVariable(VarName: string);
    function GetAllVariables: TStringListPointer;
  end;
```

The variable class makes it easy to add, change and delete user created variables. A constructor has been added for initialization of the class and a destructor is added because when the class is not needed the variables are still in memory. The destructor uses the first variable pointer to go through the list freeing each item until the end it reached indicated by a null pointer.

```
procedure TVariable.ChangeVariable(VarName: string; VarResult: extended);
var CurrentItem: TVariableItem;
begin
  CurrentItem:=FirstVariable;
  while CurrentItem<>nil do
    begin
      if SameText(VarName,CurrentItem^.VarName) then
        begin
          CurrentItem^.VarResult:=VarResult;
          Break;
        end;
      CurrentItem:=CurrentItem^.Next;
    end;
end;
```

The change variable method starts from the top of the linked list and then moves down the list until it finds a variable with the same as defined in the parameter of the procedure. This is a linear search and a more efficient search such as the binary search cannot be used because it only works on an array structure. Also, there is no way of knowing the middle of the list. If the variable does not exist then this is detected by the end of the list via a null pointer and no changes are made. This is not strictly necessary as the user interface only allows deletion of variables returned from the string list pointer data type. The get variable function is used by the tokenize procedure of the parser class, the variable name is passed as a parameter and returns the value, the item is found again using a linear search similar to the change variable procedure.

```
procedure TVariable.AddVariable(VarName: string; VarResult: extended);
var NewVariable: TVariableItem;
begin
```

```
  if IsDuplicateVariable(VarName) then Exit;
  New(NewVariable);
  NewVariable^.VarName:=VarName;
  NewVariable^.VarResult:=VarResult;
  NewVariable^.Next:=nil;

  if FirstVariable=nil then
    begin
      FirstVariable:=NewVariable;
      LastVariable:=NewVariable;
    end
  else
    begin
      LastVariable^.Next:=NewVariable;
      LastVariable:=NewVariable;
    end;
end;
```

The add variable method uses the last variable pointer so that the entire list does not have to be traversed; this variable is only used by this method for convenience at the expense of a small amount of justifiable memory. A new record is added to the linked list but before this happens the is duplicate function is called to prevent two or more variables with the same name, which in turn is another linear search. When the new variable is added the last item the points to the new one, if there is one then the last and first variable changes if there is only one item. However, the last item pointer always becomes the new variable created.

```
procedure TVariable.DelVariable(VarName: string);
var CurrentItem, PrevItem: TVariableItem;
begin
  CurrentItem:=FirstVariable;
  PrevItem:=nil;
  while CurrentItem<>nil do
    begin
      if SameText(VarName,CurrentItem^.VarName) then
        begin
          if PrevItem=nil then FirstVariable:=CurrentItem^.Next
          else PrevItem^.Next:=CurrentItem^.Next;
          if CurrentItem=LastVariable then
            begin
              if PrevItem<>nil then LastVariable:=PrevItem;
              LastVariable^.Next:=nil;
            end;
          Dispose(CurrentItem);
          Exit;
        end;
      PrevItem:=CurrentItem;
      CurrentItem:=CurrentItem^.Next;
    end;
end;
```

Finally the delete variable searches for an item based on its name, it then frees it from memory, but the list is still maintained by keeping a record of the previous item visited and copying over pointers.

```
TStack = class
  private
    TopOfStack: TStackItem;
  public
```

```
    constructor Create;
    destructor Free;
    function IsEmpty: boolean;
    procedure Push(NewItem: char); overload;
    procedure Push(NewItem: extended); overload;
    procedure Pop;
    function GetOperand: extended;
    function GetOperator: char;
    function Count: integer;
  end;
```

This class is crucial to the function of the mathematical parser because it is used by the parser class for expression evaluation. As with the previous class there is a constructor and a destructor for initialization and preventing memory leaks respectively. After the class is used there may be some items left on the stack, the destructor then pops these items until they are gone. The is empty function is self explanatory; it works by testing if the top of the stack is nil.

```
procedure TStack.Push(NewItem: char); overload;
var NewNode: TStackItem;
begin
  New(NewNode);
  NewNode^.AOperator:=NewItem;
  NewNode^.Next:=TopOfStack;
  TopOfStack:=NewNode;
end;

procedure TStack.Push(NewItem: extended); overload;
var NewNode: TStackItem;
begin
  New(NewNode);
  NewNode^.AOperand:=NewItem;
  NewNode^.Next:=TopOfStack;
  TopOfStack:=NewNode;
end;
```

The push procedures have an overload attribute so that an operand (number) or an operator (+, ^ etc.) can be pushed to the stack using the same procedure. It dynamically creates another item which then points to the last item, then the top of stack points to the new item, it the new item is the only one then it has a null pointer to indicate the end of the stack.

```
procedure TStack.Pop;
var temp: TStackItem;
begin
  temp:=TopOfStack;
  TopOfStack:=TopOfStack^.Next;
  Dispose(temp);
end;
```

The pop procedure removes the top stack item; it does this by creating a temporary pointer which is a copy of the top of stack pointer. The top of stack becomes the pointer to the next item, and then the temporary item is the freed. The get operand and get operator return their corresponding values from the top of the stack, the pop method does not return a value because there are two data type possibilities. This lead to the design choice of from the programmer's point of view to first retrieve the data needed then popping the item of the stack. An alternative could be to add a parameter to the pop procedure which then dictates which data type is returned. This would require two pop functions which will need different parameters or maybe there could be pop-operand and a pop-operator. The latter method could cause problems because if the stack item holds an operator and the programmer calls pop-operand this would cause an error, hence the current method works well and is understandable. The count function goes through the stack and increments a temporary integer to determine how many items there are on the stack and returns the value, an alternative could be to store a permanent variable which increments on push and decrements on pop. The count function would simply return its value, but the first method was chosen because there is unlikely to be more than thirty items at a time.

```
TParser = class
  private
    Stack: TStack;
    function Tokenise(Expression: string): string;
  public
    Variable: TVariable;
    constructor Create;
    destructor Free;
    function InfixToPostfix(Expression: string): string;
    function EvalPostfix(Postfix: string): extended;
    function EvalInfix(Postfix: string): extended;
  end;
```

The parser class use an aggregation relationship of the stack and variable classes as discussed earlier. The variable class needs to be public to that its methods can be accessed by the developer. The parser constructor calls the constructor of both the variable and stack class so that they are instantiated and ready to use. The destructor of the parser also calls the destructor of the variable and stack it order to prevent memory leaks. The tokenize function is used by the infix to postfix function to separate operators from operands and to put brackets around negative numbers so that they are processed correctly in the next stage of evaluation. It also replaces the name of a user created variable with its corresponding value form the variable class; its main purpose is to process the expression into a readable format for the infix to postfix algorithm.

```
function TParser.InfixToPostfix(Expression: string): string;
const Bidmas: string = '~^/*+-';
var
  postfix: string = '';
  i, j, StackPrecedence, ExpressionPrecedence, BracketCount: integer;
begin
  Expression:=Tokenise(Expression);
  i:=0;
  BracketCount:=0;
```

```
    while i<= Length(Expression) do
      begin
        Inc(i);
        case Expression[i] of
          '0'..'9','.': begin
          if Expression[i+1]in['0'..'9','.'] then
            begin
              while Expression[i] in['0'..'9','.'] do
                begin
                  postfix:=postfix+Expression[i];
                  Inc(i);
                end;
              Dec(i);
            end
           else postfix:=postfix+Expression[i];
              postfix:=postfix+' ';
          end;
          '(': begin Stack.Push(Expression[i]); inc(BracketCount); end;
          ')':begin
                Dec(BracketCount);
                while Stack.GetOperator<>'(' do
                  begin
                    postfix:=postfix+Stack.GetOperator+' ';
                    Stack.Pop;
                  end;
                Stack.Pop;
                if BracketCount=0 then
                  while not stack.IsEmpty do
                    begin
                      postfix:=postfix+Stack.GetOperator+' ';
                      stack.Pop;
                    end;
              end;
          '+','-','*','/','^','~': begin
            if Stack.IsEmpty then Stack.Push(Expression[i])
            else
              begin
                for j:=1 to 6 do
                  if Expression[i]=Bidmas[j] then ExpressionPrecedence:=j;
                for j:=1 to 6 do
                  if Stack.GetOperator=Bidmas[j] then StackPrecedence:=j;
                if (Expression[i]='^') and (Stack.GetOperator='^') then
                  begin
                    while (StackPrecedence>ExpressionPrecedence) and (not
  Stack.IsEmpty) do
                      begin
                        if Stack.GetOperator='(' then Break;
                        postfix:=postfix+Stack.GetOperator+' ';
                        Stack.Pop;
                        if (Stack.IsEmpty) or (Stack.GetOperator='(') then Break
                        else
                        for j:=1 to 6 do
                          if Stack.GetOperator=Bidmas[j] then StackPrecedence:=j;
                      end;
                  end
                else
                  begin
                    while (StackPrecedence<=ExpressionPrecedence) and (not
  Stack.IsEmpty) do
                      begin
                        if Stack.GetOperator='(' then Break;
                        postfix:=postfix+Stack.GetOperator+' ';
                        Stack.Pop;
                        if (Stack.IsEmpty) or (Stack.GetOperator='(') then Break
                        else
                        for j:=1 to 6 do
```

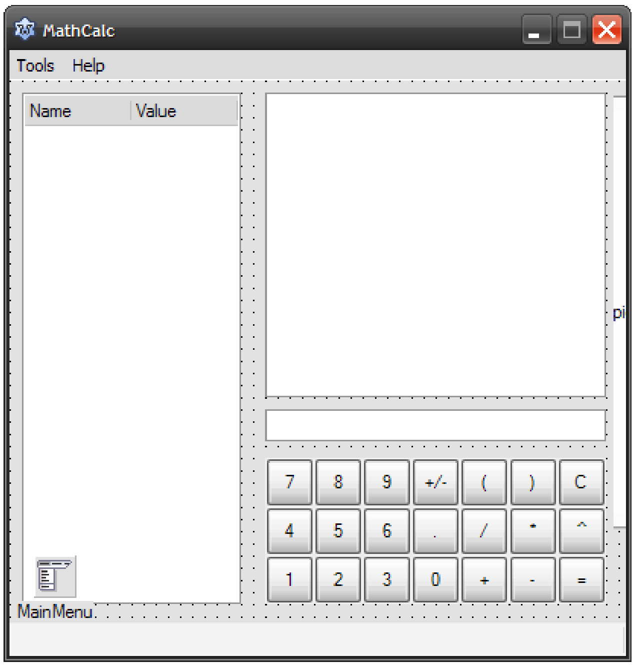
```
                          if Stack.GetOperator=Bidmas[j] then StackPrecedence:=j;
                      end;
                  end;
                  Stack.Push(Expression[i]);
              end;
          end;
      end;
    end;

    while not Stack.IsEmpty do
      begin
        postfix:=postfix+Stack.GetOperator+' ';
        Stack.Pop;
      end;
    Result:=Postfix;
  end;
```

Taking an expression, obeying the BIDMAS rules and finding the solution is difficult to do in one step. Instead the expression must be first converted to postfix also know as reverse polish notation so that it can easily be evaluated. The infix to postfix algorithm uses a stack based method to make the conversion, hence why the stack class it needed. The algorithm incorporates parentheses, negative and decimal values as well as multiplication, division and powers to make it a general purpose maths tool. To make the conversion the input string is traversed and a tree is developed and presented into reverse polish notation following the BIDMAS rules and returned as a string as well. This is done by scanning the string and differentiating between an operator and operand, then each one is pushed onto the stack. If an operator is encountered then it's precedence it compared with the previous operator on the stack if the precedence is higher then it is added to the output string until there are no items or an operator with lower precedence is encountered. If the stack is empty then the current operator is pushed onto the stack regardless. There were some problems related to powers which were discovered from testing, this is because powers are right associative while other operators are left associative, the design of the algorithm does not take this into account as stated in the design section and treats powers with left associative behaviour. This is why manipulating multiple powers when converting infix to postfix returns the incorrect answer because operations are not being carried out in the correct order. To correct this when a power is encountered the precedence comparison is reversed in order to become right associative which corrects the problem in the algorithm. When an opening parenthesis is detected then it is pushed to the stack and the bracket count is incremented. Then a closing bracket forces all items to be popped of the stack and onto the output until the opening bracket is encountered. Parenthesis is never added to output because postfix already defines the order of execution. The end is reached when the input string parameter end it reached. Then all items on the stack are then popped onto the output, finishing the algorithm and allowing the stack to be used later because it will be empty.

```
function TParser.EvalPostfix(Postfix: string): extended;
var
  i: integer;
  temp1, temp2: extended;
  tempstr: string;
begin
  i:=0;
  while i<Length(Postfix) do
    begin
      Inc(i);
      if Postfix[i] in['0'..'9'] then
        begin
          tempstr:='';
          while Postfix[i]<>' ' do
            begin
              tempstr:=tempstr+Postfix[i];
              Inc(i);
            end;
          Stack.Push(StrToFloat(tempstr));
        end
      else if Postfix[i] in['+','-','*','/','^','~'] then
        begin
          temp1:=Stack.GetOperand;
          Stack.Pop;
          if (Stack.Count>=1) and (Postfix[i]<>'~') then
            begin
              temp2:=Stack.GetOperand;
              Stack.Pop;
            end
          else if Postfix[i]<>'~' then raise ErrorMsg.Create('Error with
expression');
          case Postfix[i] of
            '+': Stack.Push(temp2+temp1);
            '-': Stack.Push(temp2-temp1);
            '*': Stack.Push(temp2*temp1);
            '/': Stack.Push(temp2/temp1);
            '^': Stack.Push(Power(temp2,temp1));
            '~': Stack.Push(temp1*-1);
          end;
        end;
    end;
  Result:=Stack.GetOperand;
  while not Stack.IsEmpty do Stack.Pop;
end;
```

The evaluate postfix function uses the stack to push items from the string, this is when the overloaded push procedure is used because in the infix to postfix procedure only operators are added to the stack, now only operands are used. When an operator is encountered the last two items are popped and manipulated by the operator, the result is then pushed back onto the stack. When the end of the string is reached only the final and ultimate answer should remain which is then returned by the function. There is some error detection such as when an operator is encountered and there are not at least two operands then this will cause a run time error but this event is caught and handled. When finished with the stack, all items are popped so that it can be used later. The evaluate infix function calls the infix to postfix function and then the evaluate postfix function to return the solution with one call and abstracts some complexities of the class, it simply wraps the two functions into one for convenience.

Main Window

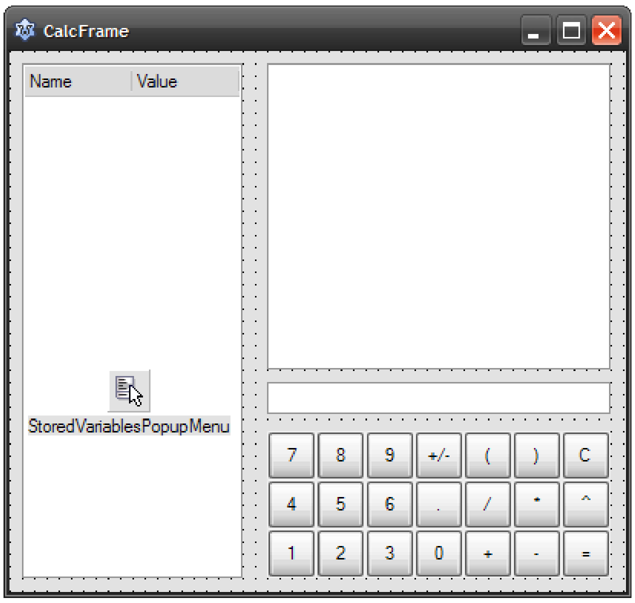


This is the design view of the main form in Lazarus. It has a main menu which can be seen at the top left and a status bar at the bottom. At the centre there is a frame which contains the calculator frame, behind it you can see a little of the revision frame behind it, to the right. The visibility of the revision frame is set to false by default and only one frame will be shown at any time.

```
type
  TMainForm = class(TForm)
    CalcFrame: TCalcFrame;
    MainMenu: TMainMenu;
    MathTopicFrame: TMathTopicFrame;
    MenuGraph: TMenuItem;
    MenuRevise: TMenuItem;
    MenuNumGen: TMenuItem;
    MenuAbout: TMenuItem;
    MenuHelp: TMenuItem;
    MenuTools: TMenuItem;
    MenuUnitConv: TMenuItem;
    StatusBar: TStatusBar;
    procedure MenuAboutClick(Sender: TObject);
    procedure MenuGraphClick(Sender: TObject);
    procedure MenuNumGenClick(Sender: TObject);
    procedure MenuReviseClick(Sender: TObject);
    procedure MenuUnitConvClick(Sender: TObject);
  end;
```

All of these procedures are for the main menu which allows users to move to other area of the program. The main form is frame based to minimise complexity.

Calculator

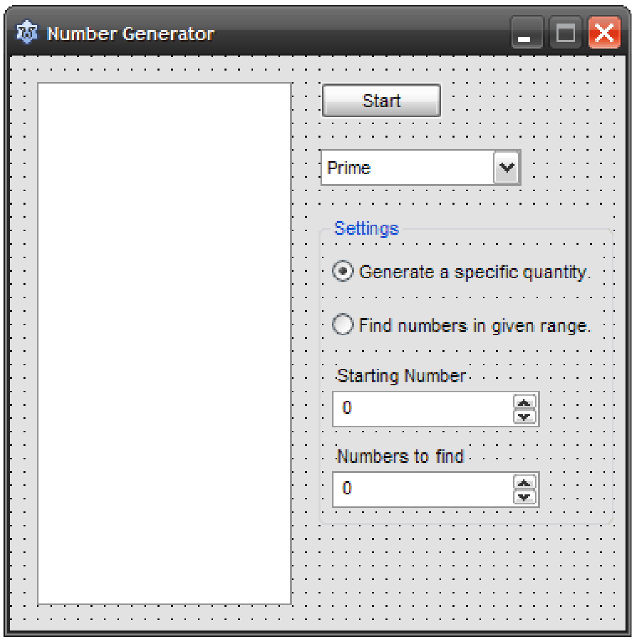


The calculator does not have its own window, but uses the frame component which will be integrated into the main form. There is a pop-up menu used for the string grid so that users can create, modify and delete variables.

```
type
  TCalcFrame = class(TFrame)
    InputButton1: TButton;
    InputButton0: TButton;
    InputButtonPower: TButton;
    InputButtonOpenBracket: TButton;
    InputButtonDivide: TButton;
    InputButtonCloseBracket: TButton;
    InputButtonEquals: TButton;
    InputButtonClear: TButton;
    InputButtonTimes: TButton;
    InputButtonMinus: TButton;
    InputButtonAdd: TButton;
    InputButtonPlusMinus: TButton;
    InputButtonDecimal: TButton;
    InputButton2: TButton;
    InputButton3: TButton;
    InputButton4: TButton;
    InputButton5: TButton;
    InputButton6: TButton;
    InputButton7: TButton;
    InputButton8: TButton;
    InputButton9: TButton;
    ExpressionInput: TEdit;
    HistoryMemo: TMemo;
    MenuAddVar: TMenuItem;
    MenuDelVar: TMenuItem;
    MenuChangeVar: TMenuItem;
    StoredVariablesPopupMenu: TPopupMenu;
    StoredVariables: TStringGrid;
    procedure ExpressionInputKeyPress(Sender: TObject; var Key: char);
    procedure InputButtonClick(Sender: TObject);
    procedure InputButtonClearClick(Sender: Tobject);
    procedure InputButtonEqualsClick(Sender: Tobject);
    procedure PlusMinusClick(Sender: TObject);
    procedure MenuAddVarClick(Sender: TObject);
    procedure MenuChangeVarClick(Sender: TObject);
    procedure MenuDelVarClick(Sender: TObject);
    procedure StoredVariablesContextPopup(Sender: TObject);
  private
    Parser: TParser;
  public
    procedure UpdateVarList;
  end;
```

This frame has a large number of visual components, a memo is used to display the expression history, a string grid to display user created variables, buttons for data input and an edit-box where expressions are entered into. Expression input key press procedure is for the edit-box, the enter key as ASCII value of 13, if this is found then input button equals click procedure is called. The input button click procedure takes the caption of the sender and appends it to the edit-box, this applies to most buttons like the number six or plus sign. The input button clear click procedure clears all text from edit-box, only applies to the C button. Input button equals uses the evaluate infix function from the parser class and converts it to a string. Plus minus click add a minus to the front of the contents of the edit-box or removes it if the box already has one. Then next three procedures are for the pop-up menu, they are self explanatory and provide add, modify and delete options by using the variable class method from the parser class. The next procedure shows the pop-up menu if the mouse is over the string grid. The very last procedure uses the get all variable function from the variable class, which returns pointer to a string list which is used to fill the string grid.

Number Generator



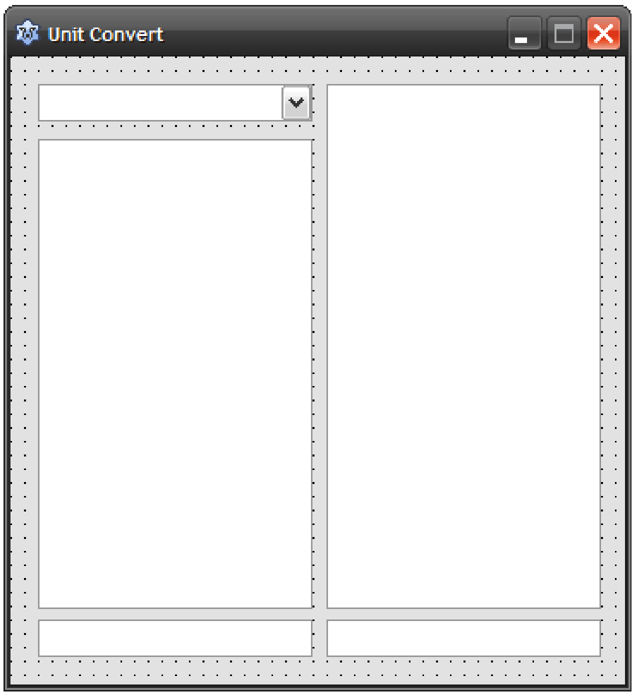
Only one radio button in the group box may be active at one time, but the default setting is that the top one is active and the second inactive. Based on the active radio button determines the caption of the spin edits. The spin edits only accept a range of 0 – 1x10^9, if the value is negative it is becomes zero, above the limit the inputted value is changed to the maximum allowed digit. If there are numbers in the list-box then they are cleared before starting a new sequence.

```
type
```

```
  TNumberGenForm = class(TForm)
    GenSelect: TComboBox;
    LabelStart: TLabel;
    LabelEnd: TLabel;
    MethodGroup: TGroupBox;
    Method1Radio: TRadioButton;
    Method2Radio: TRadioButton;
    MethodEnd: TSpinEdit;
    MethodStart: TSpinEdit;
    OutputList: TListBox;
    Start: TButton;
    procedure Method1RadioClick(Sender: TObject);
    procedure Method2RadioClick(Sender: TObject);
    procedure StartClick(Sender: TObject);
  public
    procedure GenPrime;
    procedure GenSquare;
    procedure GenTriangle;
    procedure GenComposite;
    function IsPrime(const Test: integer): boolean;
    procedure GetMethod(out Param1, Param2: integer; out Method: boolean);
  end;
```

This form does not need any public variables because they are displayed via the list-box visual component. The method 1 radio click procedure is used by top radio button, it changes the caption of the spin-edits to explain there function to the user and so does the next procedure. When the start button is clicked the stack click procedure is executed, based on what the user has entered determines what pattern will be made and the quantity. The next four generator procedures calculate the sequence based on their name and output the results to the list-box. The is prime function is used by the generate prime and composite procedures to avoid repeating code. The get method procedure returns the data the user have entered into the form to determine what they specified.

Unit Convert

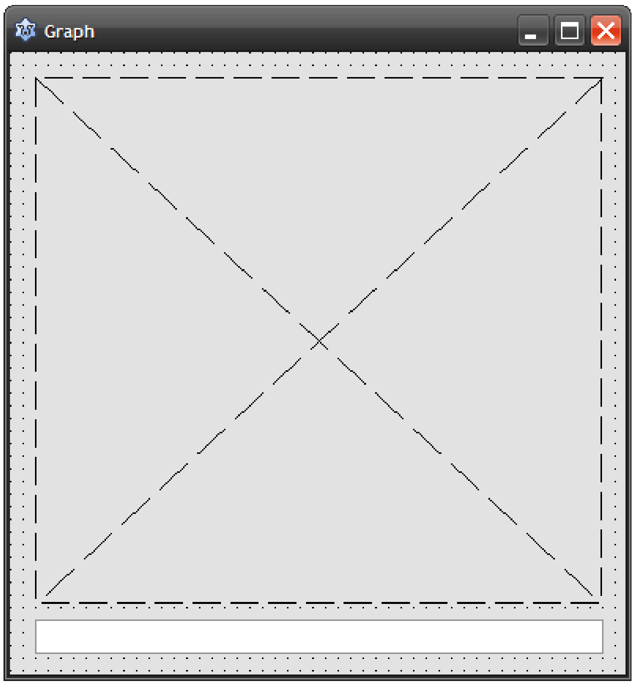


This is a simple interface in which errors are kept at a minimum because all erroneous data is rejected before conversions are made. This keeps users required knowledge of the system to be at a minimal level by abstracting the complexities. Also units are separated into categories, where one category cannot be converted with another, e.g. time and mass. This will reduce runtime errors making the system easier to operate.

```
type
  TUnitConvertForm = class(TForm)
    ConvertValue: TEdit;
    ResultValue: TEdit;
    Filter: TComboBox;
    ResultList: TListBox;
    ConvertList: TListBox;
    procedure ConvertListSelectionChange(Sender: TObject; User: boolean);
    procedure ConvertValueChange(Sender: TObject);
    procedure ConvertKeyPress(Sender: TObject; var Key: char);
    procedure FilterChangeBounds(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure ResultListSelectionChange(Sender: TObject; User: boolean);
    procedure ResultValueChange(Sender: TObject);
  public
    procedure ReturnConvTypes(var ConvertTo, ConvertFrom: TConvType);
  end;
```

The form create procedure provides initialization of the form and gets the list of available unit types to be provided in the combo box. It then selects the default area unit type and displays the corresponding measurements on both List-box. When the index of the combo box changes the filter change bound procedure is called, this causes the left list box is filled with a new list of items from the selected unit type from the combo box. This in turn then fills the second list box with available conversion units. Convert list selection change is executed when there is a selection change in the left combo box which then triggers the contents of the right box to be recalculated. Therefore giving a list of valid selections to the user, removing to possibility computing related error. Convert key press is executed when the user's input into any of the edit boxes is monitored at real-time, the contents can only contain the characters one to nine and one decimal point. If character is otherwise then it is not added to the edit box. This eliminates string to float errors. If the user's input character passes convert key press from the convert value change procedure then the alternative edit box contents it calculated from the contents of the edit box the user has entered data in. This works for both edit boxes. If the edit is blank there will be an error, but an exception statement has been added so when this situation occurs no conversion will be made. The result values change procedure is the same as the previous procedure mentioned but when data in inputted into the right edit-box the left is recalculated from the data entered.

Graph Render

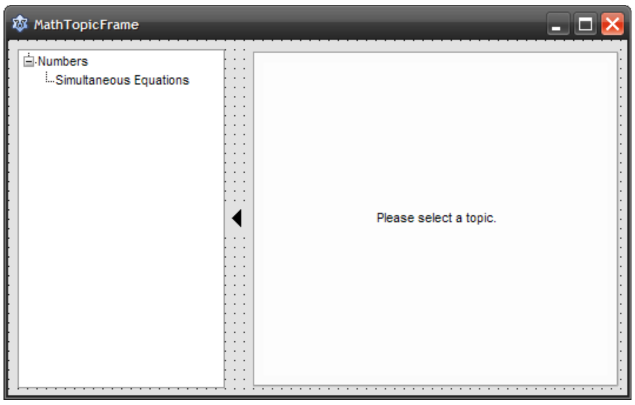


The paint-box component is useful because it makes it convenient to draw to and modify a defined area. The down side it that if the window is minimized or another window overlaps it, the contents are not automatically redrawn. This problem can be fixed by having a redraw procedure that will execute when the graph window repainted. There is no button to accept input but when the edit-box is in focus and a return is detected then the function draw procedure is executed.

```
type
  TGraphForm = class(TForm)
    FunctionInput: TEdit;
    PaintArea: TPaintBox;
    procedure FunctionInputKeyPress(Sender: TObject; var Key: char);
    procedure FormCreate(Sender: TObject);
    procedure PaintAreaPaint(Sender: TObject);
  private
    Parser: TParser;
    XRange: integer;
    YRange: integer;
    XRangePixels: integer;
    YRangePixels: integer;
    CorrectX, CorrectY: integer;
    Origin: TPoint;
  public
    x: extended;
    procedure DrawAxis;
    procedure DrawGraph;
  end;
```

The functionality of the graph form is loosely based on the parser class to provide the result of a function at predefined points of a certain range. The form create procedure creates an instance of the parser so that is can be initialized; it also creates a variable with the name of x to be used by the draw graph procedure. To make programming easier the functionality of the parser is used, such is the variable x that is set to the x coordinate of the input function. This variable is incremented and the input expression is re-evaluated and added to a temporary array which when finished is used to paint the graph to screen. The paint procedure renders the axis on the paint box while the following procedure detects a return in the edit-box to determine if a user has finalized their input. The draw axis procedure is simple as it draws two perpendicular lines based on the size of the paint-box. The draw graph procedure uses the parser to change the value of x based on which point is been calculated, then it translates the points to pixels and joins up the pixels to create a polygonal line. There are a superfluous number of points to add detail and accuracy to the graph which will enable curves and other shapes to appear natural.

Revision



The main objects on the revision frame are the tree view and notebook. A notebook is the perfect component in this situation because it can have multiple pages and the tabs can be hidden from the user. So pages can be set to change based on the selection of the tree view component.

```
type
  TPageData = record
    Stage: shortint;
    Difficulty: byte;
    X: integer;
    Y: integer;
    XCo1: integer;
    YCo1: integer;
    XCo2: integer;
    YCo2: integer;
    Res1: integer;
    Res2: integer;
  end;

type
  TMathTopicFrame = class(TFrame)
    Blank: TPage;
    BlankLabel: TLabel;
    SelDif: TComboBox;
    SimEquLabelExp1: TLabel;
    SimEquLine: TShape;
    SimEquLabel3: TLabel;
    SimEquLabel4: TLabel;
    SimEquAdd: TLabel;
    SimEquNext: TButton;
    SimEquCombo: TComboBox;
    ListBox1: TListBox;
    SimEquLabel1: TLabel;
    SimEquLabel2: TLabel;
    Item1: TPage;
    TriandImp: TPage;
    ShowHideTree: TArrow;
    Button3: TButton;
    Notebook: TNotebook;
    SimEqu: TPage;
    TopicTree: TTreeView;
    procedure ShowHideTreeClick(Sender: TObject);
    procedure SimEquNextClick(Sender: TObject);
    procedure TopicTreeSelectionChanged(Sender: TObject);
  private
    SimEquData: TPageData;
  public
    procedure GenVariables(const PageName: string);
    procedure GenSimEqu(const GenStage: byte);
  end;
```

The page data record exists is so that each page of the notebook can have its set of variables which can be used as and when needed. The arrow shows and hides the tree view so that there are allowing more space for the main objects on-screen and to provide a minimalist interface. This is achieved by the show hide tree click procedure which hides the tree view and moves the notebook to the edge of the frame, or shows the tree view and moves the notebook back to its original position. The next procedure is called when the user selects the simultaneous equation revision item from the tree view; it initializes the page data for that page so that it can be used. Topic tree selection changed procedure swaps the page of the notebook based on what is selected is in the tree view. Generate variables procedure calls the correct generation procedure base on what has been selected in the tree view. Meanwhile the procedure called generate simultaneous equation advances the current stage defined by the parameter, 0 for the first stage. If a stage is called that does not exist then nothing will happen.

User Manual

Contents

Introduction

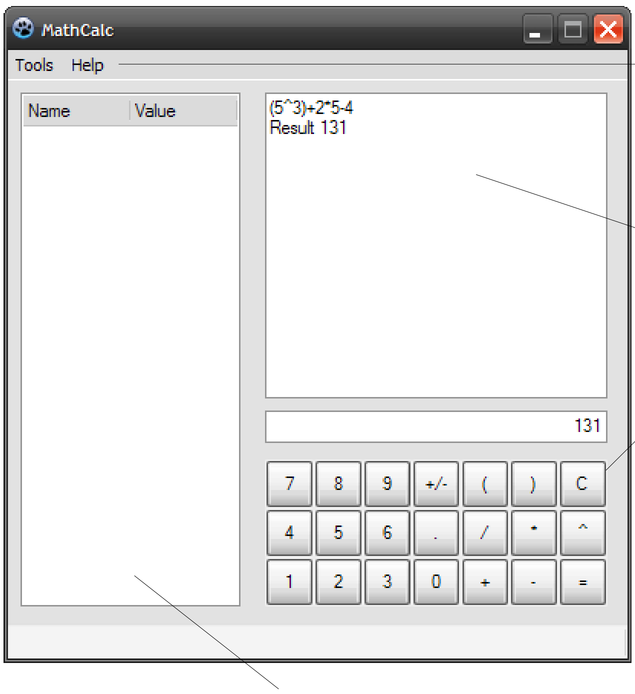
This user manual is designed to give you all the information you will need to operate and fully utilize the features of MathCalc. The software can evaluate equations and return the result when given as infix. The main features include unit conversion, number generation and graph rendering as well as the ability to show the user how to solve equations.

Installation

The software is portable and can be executed on any computer with a user that has the necessary permissions. It does not make registry entries and leaves no temporary data on disk. To install the software, copy the folder containing the executable to your Documents or Programs area, for quick access you may want to create a short cut of the program and place it on the desktop.

Calculator

Here is the main window of the program which also houses the built-in calculator.



```
(5^3)+2*5-4
Result 131
```

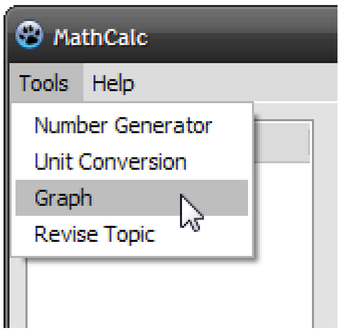
```
131
```

Looking at the top left you will notice the main menu; here you will have access to other areas of the program.

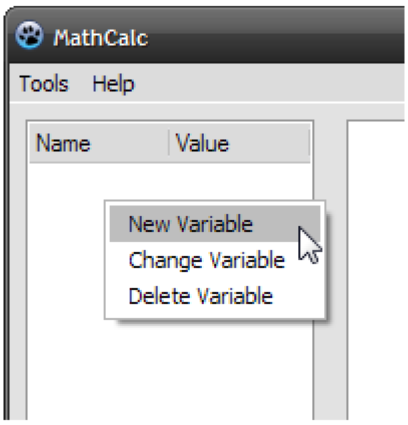
This box records the history of all equations. Useful if you need to backtrack or check you working.

Here is where you will enter the expressions to be processed. You can use the buttons below or type in what you want in the text area. The program accepts brackets, plus, minus, times, divide and powers as well as negative and decimal numbers.

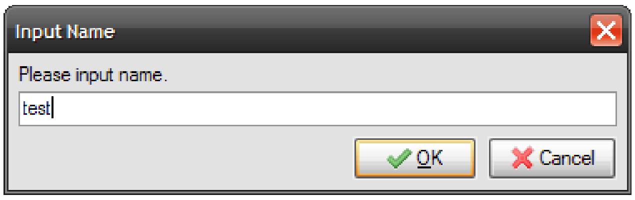
To the far left is an area where you can define your own variables. It will be explained later.



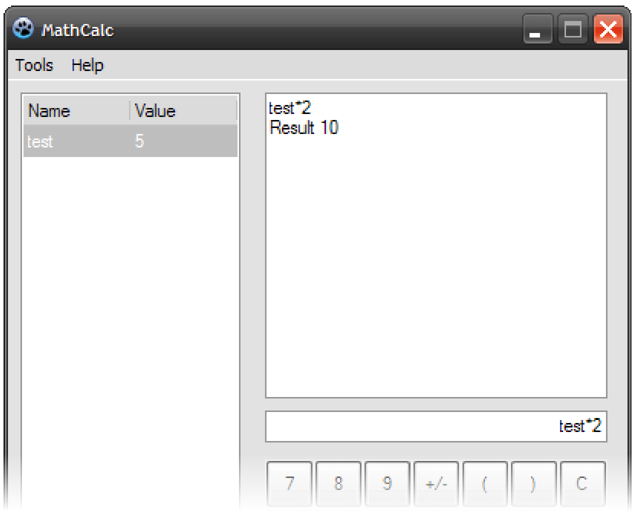
From the tools menu you can switch between different areas of the program easily. Also data you had in one area will not be lost when you switch to another. At the moment there the help menu does not provide assistance but only contains the about box, which contains information about MathCalc.



To add variables just right click anywhere in the box as shown, then a box will appear (see below) that will ask you for the name of the variable. Then another will appear asking for the value. To change an item just select it right click and select change variable, as before a box will appear allowing you to enter the new value. Deletion is similar to modifying but this time you select the delete variable option from the right click menu.

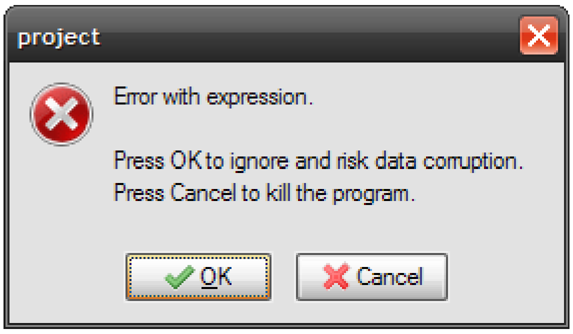


Here is the input box as mentioned earlier. In this instance a new variable is being added with the name test.



```
test*2
Result 10
```

Here the variable named test has been created with a value of 5. To use it simply treat as an operand, meaning type the name when you want to use the variable. Here test has been multiplied by two to get the result of ten, which is correct. As the alphabet is not shown on screen keyboard input is the only way to use variables unless you use a third party on screen keyboard.

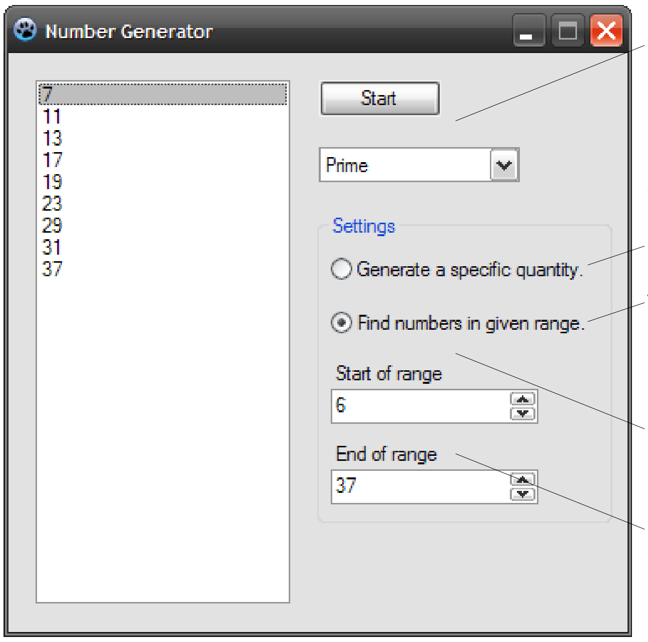


Error with expression.

Press OK to ignore and risk data corruption.
Press Cancel to kill the program.

If you make a mistake when input an expression such as missing operator or a misspelling variable, don't worry. You will be given the option to continue or to cancel. If you cancel you will have to restart the program because it will have ended because of preventing potential data corruption.

Number Generator



7
11
13
17
19
23
29
31
37

Prime

Generate a specific quantity.
Find numbers in given range. ●

Start of range
6

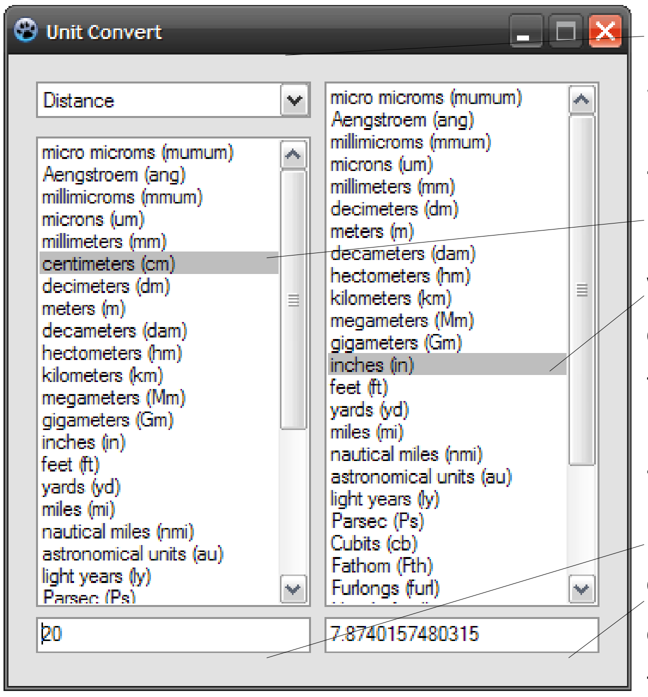
End of range
37

This drop down menu holds a list of available number patterns

There are two methods to search for numbers, the first is to start from a given value and then find a specified number of numbers. Alternatively, all values can be searched between two values.

Based on which radio button is checked the caption of these number boxes will change making their purpose self explanatory.

Unit Convert



Distance

micro microms (mumum)
Aengstroem (ang)
millimicroms (mmum)
microns (um)
millimeters (mm)
centimeters (cm)
decimeters (dm)
meters (m)
decameters (dam)
hectometers (hm)
kilometers (km)
megameters (Mm)
gigameters (Gm)
inches (in)
feet (ft)
yards (yd)
miles (mi)
nautical miles (nmi)
astronomical units (au)
light years (ly)
Parsec (Ps)

micro microms (mumum)
Aengstroem (ang)
millimicroms (mmum)
microns (um)
millimeters (mm)
decimeters (dm)
meters (m)
decameters (dam)
hectometers (hm)
kilometers (km)
megameters (Mm)
gigameters (Gm)
inches (in)
feet (ft)
yards (yd)
miles (mi)
nautical miles (nmi)
astronomical units (au)
light years (ly)
Parsec (Ps)
Cubits (cb)
Fathom (Fth)
Furlongs (furl)

20

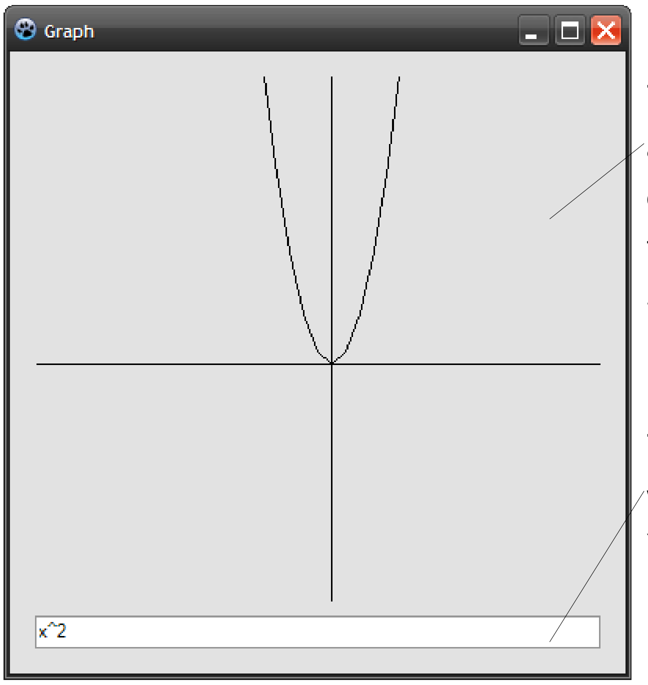
7.8740157480315

Here is the unit type selector, units are separated to eliminate incompatible types.

The left box contains all available measurements for the selected unit type, while the right box lists the available conversion units related to the selection from the left selection.

The left input area represents the selected measurement, as numbers are entered the conversion is calculated in real-time. You can also edit the right input area to reverse the conversion.

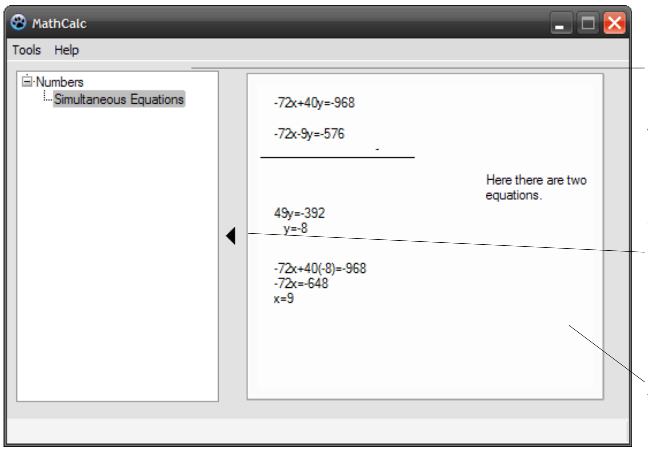
Graph Render



The main body of the window houses the area where the graph is drawn. Usage is easy because you only need to enter a function and it is not necessary to change the scale of the graph.

To enter a function just simply click on the white rectangle at the bottom of the window. You do not need to put the notation f(x) as it is not needed.

Revision



```
-72x+40y=-968

-72x-9y=-576
                 -
______________________

                              Here there are two
                              equations.
49y=-392
  y=-8

-72x+40(-8)=-968
-72x=-648
x=9
```

Currently, only simultaneous equation revision is available, at the left you can select which topic you want to revise.

This arrow can show/hide the topic selection box.

At right box you will be shown a step by step solution with on screen instructions.

Evaluation

Meeting Objectives

Objective 1.1 Relaxed calm environment for users

The user interface of the solution is designed to be as simple as possible, fonts and colour schemes are inherited from the operating system so that users are in a familiar environment. Also error handling is presented by a simple error message that explains the problem so that the issue can be resolved quickly.

Objective 1.2 Clean graphical user interface

As mentioned earlier the program uses the same visual theme as the operating system. All text in the program has been written concisely and presented utilizing available screen space but designed to avoid cluttering.

Objective 1.3 Quick and responsive program

As long as the system requirements are met then the program will operate in a timely manor. These requirements are very low and most if not all computers should be able to execute the program quickly.

Objective 1.4 Variety of maths topics accessible to the user

Currently the program only houses the ability to teach users how to solve simultaneous equations. If given more time, more topics from the GCSE would be added to the program, this can be easily achieved due to the modular nature of program.

Objective 1.5 Users can set difficultly of a topic

Simultaneous equations revision currently has a three point scale of difficultly ranging from easy to hard. If given more time than this system would also be implemented into new topics from the GCSE specification.

Objective 1.6 Guided through topic using concise presentation

The current working revision topic has been simplified into five steps, without omitting key and valuable information. Only relevant text is shown at any single time to keep text on screen at a minimum.

Objective 1.7 Questions generated by computer

As the only topic currently available is simultaneous equation revision, the equations are randomly generated and this is the key aspects to add variety. However, this is limited to the topics available and if given time these would be a high implementation priority.

Objective 1.8 Users are able to be shown the solution

Revision is pointless without checking if your method procedures the correct result. All topics that are offered to users will have this feature because without it would make revising a more arduous task.

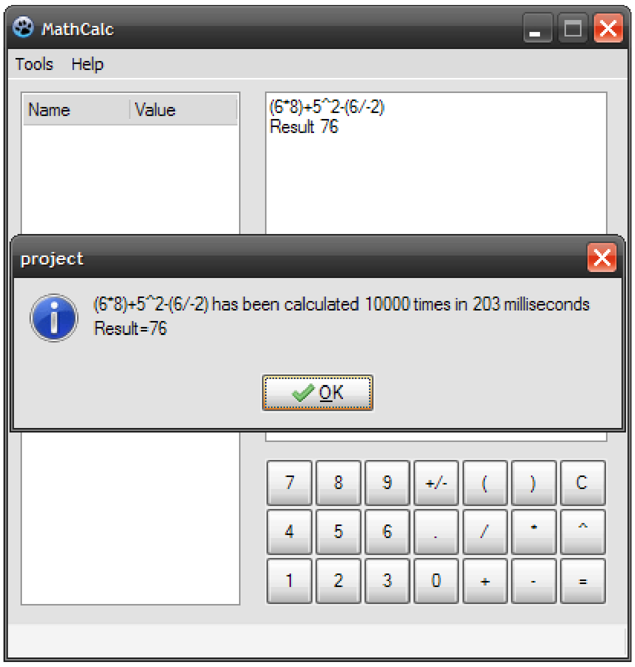
Objective 2.1.1 Complies with BIDMAS

The parser fully recognizes the rules of brackets, indecencies, division, multiplication, addition and subtraction and their precedence value. Evidence is clearly depicted in the results of testing.

Objective 2.1.2 Supports parentheses, +, -, *, /, ^ operators

As well as finding the correct order that calculation need to be processed in it can also perform these actions as well as return the result to the user. Testing has shown that the parser is reliable and all known bugs have been corrected.

Objective 2.1.3 Computes quickly



```
(6*8)+5^2-(6/-2)
Result 76
```

```
(6*8)+5^2-(6/-2) has been calculated 10000 times in 203 milliseconds
Result=76
```

A few modifications have been to the program to demonstrate the computational time of calculating the result of an expression. The screen-shot shows that it took an AMD Athlon 64 X2 6400+ less than a quarter of a second to calculate the result 10000 times. Users will only need to perform one calculation at a time showing that efficiency of the algorithm is sufficient for user needs.

Objective 2.1.4 Past calculations are stored in history

The calculator component of the program stores all past calculations above the input area of the main window. It shows the expression and its result so that if needed it can be used later.

Objective 2.1.5 Dynamic user defined variables

At the left area of the main window there is the user variable interface that allows creation, modification and deletion of variables.

Objective 2.2 Users can draw graphs

Users have the ability to draw graphs with the use of the graph tool that has been implemented into the program. It handles basic graphs that are relevant to the GSCE syllabus.

Objective 2.3 Unit converter to help understand units

The unit converter allows users to perform conversions between popular measurements easily by using the built-in tool. Conversions are displayed using a considerable number of decimal places for accuracy and to minimise rounding errors that may occur.

Objective 2.4 Number generator to clarify patterns

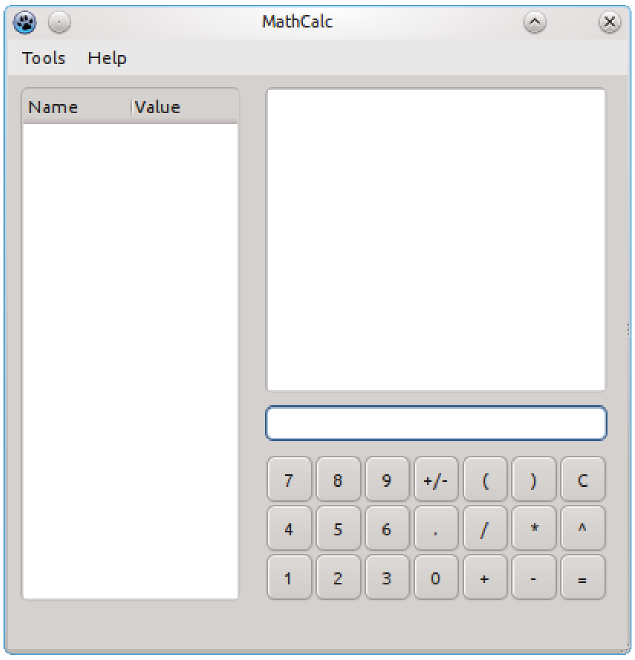
The number generator allows users to display common number patterns and therefore allow these values to be used when needed. It calculates values quickly and is organized in an ordered list.

Objective 2.5 Portable program

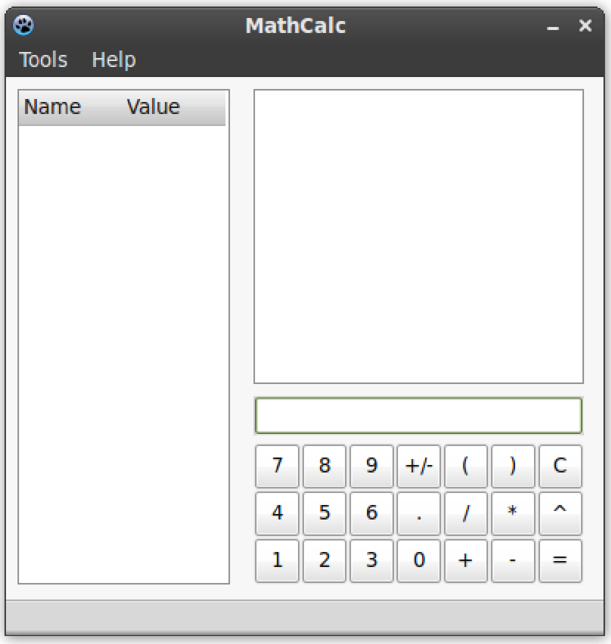
The program does not need to be installed making it portable, no registry entries are made and it has no dependencies. These help it to be portable and it can be used on any storage medium with sufficient space.

Objective 2.6 Platform-independent

Here are some screen-shots of the program in two different operating systems.



Kubuntu 10.10



Linux Mint 9

Due to the nature of Free Pascal the program can be compiled for many operating systems such as Linux, MacOS and Windows as including architectures such as ARM, X86 and PowerPC. This shows that the program can be used in a diverse range of hardware and software combination.

User Feedback

After sending a copy of the program to Rob Lamb he sent a letter back stating his opinion of the project along with prospective users. (See Appendix – Letter from Rob Lamb).

From this information the overall progress of the project is developing an acceptable rate but there is a significant issue. If given more time, then there would be more topics pulled from the GSCE syllabus and arranged into a revisable format. Other extensions may include the ability to store information about user progress and performance.

Appendix

Analysis

Questionnaire

Question 1

1.1 Please state your confidence in maths?

Very Low	Low	Medium	High	Very High
☐	☐	☐	☐	☐

1.2 How many hours do you spend revising each week?

0 – 2 Hours	> 2 – 4 Hours	> 4 – 6 Hours	> 6 – 8 Hours	> 8 Hours
☐	☐	☐	☐	☐

1.3 Please select your preferred revision methods?

☐ Associative (patterns)	☐ Audio (audio book)	☐ Dialogic (talking)
☐ E-learning (computer)	☐ Flash cards	☐ Multimedia
☐ Observation	☐ Reading	☐ Visual

Other ___________________

1.4 Please select what learner types you are?

☐ Activity based	☐ Auditory leaner	☐ Collaborative
☐ Cramming	☐ Experiential (practice)	☐ Habit leaner (repetition)
☐ Kinaesthetic (physical)	☐ Reciprocal (talking)	☐ Rote (repetition recall)
☐ Trial and error	☐ Visual learner	Other ___________________

1.5 Do you believe that you would benefit from extra revision material?

☐ Yes ☐ No

Reason

Question 2

2.1 Do you own a computer?

☐ Yes ☐ No

2.2 If applicable, do you have an internet connection?

☐ Yes ☐ No

2.3 Are you comfortable using computers?

Very Low	Low	Medium	High	Very High
☐	☐	☐	☐	☐

2.4 Do you believe that you would use a computer based revision system?

☐ Yes ☐ No

Reason

2.5 Please select which revision methods you would like to be implemented into the system?

☐ Associative (patterns)	☐ Audio (audio book)	☐ Dialogic (talking)
☐ E-learning (computer)	☐ Flash cards	☐ Multimedia
☐ Observation	☐ Reading	☐ Visual

Other ___________________

2.6 Please select what additional functionality you would need the revision to be able to perform?

☐ Dynamic Questions	☐ Dynamic Variables	☐ Graph rendering
☐ Performance statistics	☐ Variable difficultly	Other ___________________

Question 3

3.1 Describe your opinion of graphical/scientific calculators?

Graph drawing	☐ Slow	☐ Adequate	☐ Quick
Number of variables	☐ Low	☐ Adequate	☐ Many
Screen size	☐ Small	☐ Adequate	☐ Large
Ease of use	☐ Difficult	☐ Adequate	☐ Easy

Questionnaire Results

Question 1

1.1 Please state your confidence in maths?

Very Low	Low	Medium	High	Very High
2	**8**	**13**	**6**	**1**

1.2 How many hours do you spend revising each week?

0 – 2 Hours	> 2 – 4 Hours	> 4 – 6 Hours	> 6 – 8 Hours	> 8 Hours
5	**11**	**8**	**4**	**2**

1.3 Please select your preferred revision methods?

3 Associative (patterns)	**0** Audio (audio book)	**2** Dialogic (talking)
16 E-learning (computer)	**7** Flash cards	**1** Multimedia
2 Observation	**6** Reading	**20** Visual

Other (none)

1.4 Please select what learner types you are?

3 Activity based	**0** Auditory leaner	**4** Collaborative
15 Cramming	**12** Experiential (practice)	**6** Habit leaner (repetition)
0 Kinaesthetic (physical)	**5** Reciprocal (talking)	**4** Rote (repetition recall)
6 Trial and error	**23** Visual learner	Other (none)

1.5 Do you believe that you would benefit from extra revision material?

23 Yes **7** No

Reason

Question 2

2.1 Do you own a computer?

30 Yes **0** No

2.2 If applicable, do you have an internet connection?

30 Yes **0** No

2.3 Are you comfortable using computers?

Very Low	Low	Medium	High	Very High
0	**2**	**5**	**18**	**5**

2.4 Do you believe that you would use a computer based revision system?

24 Yes **6** No

Reason

2.5 Please select which revision methods you would like to be implemented into the system?

4 Associative (patterns)	**0** Audio (audio book)	**0** Dialogic (talking)
16 E-learning (computer)	**4** Flash cards	**7** Multimedia
0 Observation	**7** Reading	**21** Visual
Other (**4** interactive)		

2.6 Please select what additional functionality you would need the revision to be able to perform?

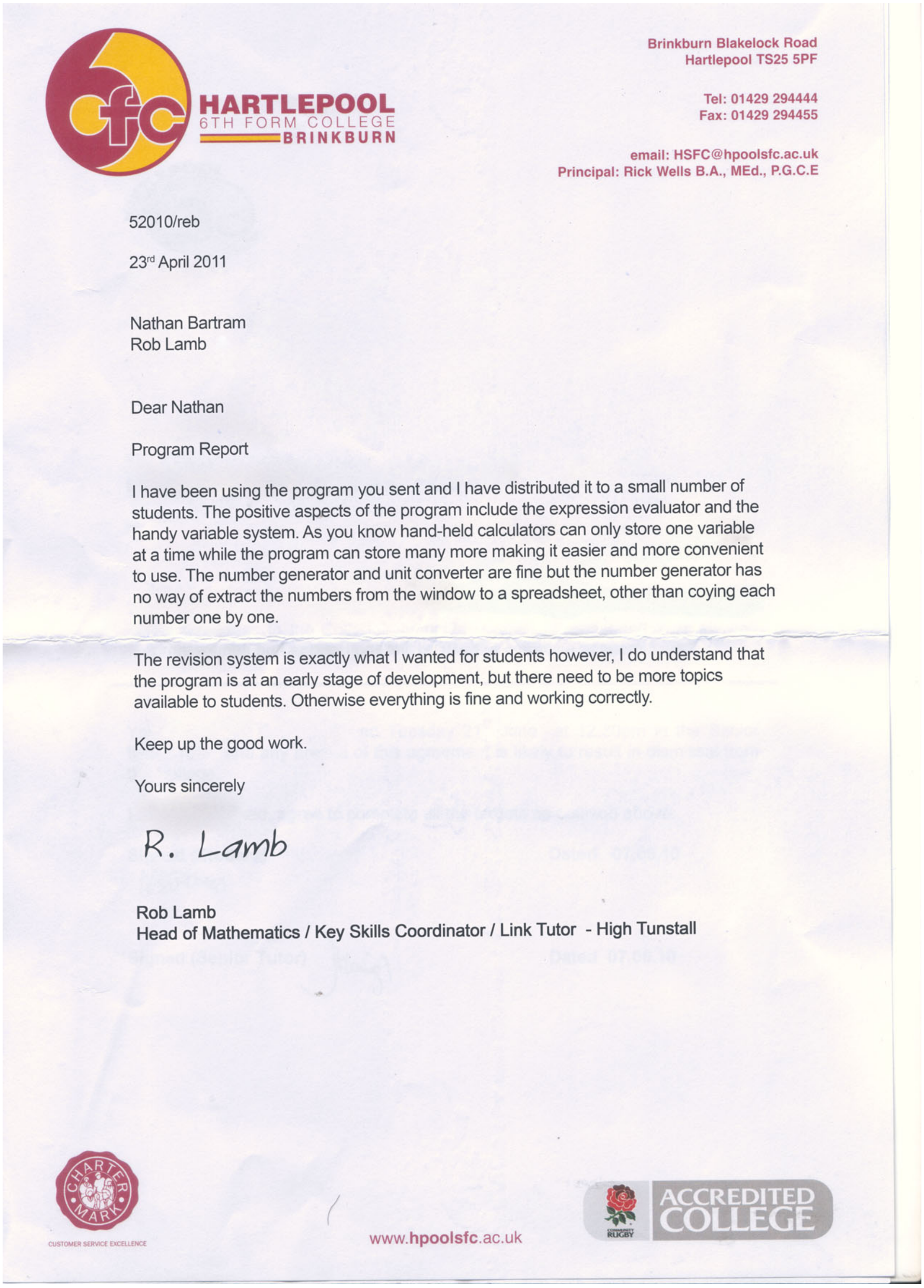
8 Dynamic Questions	**15** Dynamic Variables	**19** Graph rendering
5 Performance statistics	**9** Variable difficultly	Other (**17** unit converter, **18** number generator)

Question 3

3.1 Describe your opinion of graphical/scientific calculators?

Graph drawing	**22** Slow	**8** Adequate	**0** Quick
Number of variables	**16** Low	**14** Adequate	**0** Many
Screen size	**14** Small	**16** Adequate	**0** Large
Ease of use	**17** Difficult	**10** Adequate	**3** Easy

Letter from Rob Lamb



HARTLEPOOL
6TH FORM COLLEGE
BRINKBURN

Brinkburn Blakelock Road
Hartlepool TS25 5PF

Tel: 01429 294444
Fax: 01429 294455

email: HSFC@hpoolsfc.ac.uk
Principal: Rick Wells B.A., MEd., P.G.C.E

52010/reb

23rd April 2011

Nathan Bartram
Rob Lamb

Dear Nathan

Program Report

I have been using the program you sent and I have distributed it to a small number of students. The positive aspects of the program include the expression evaluator and the handy variable system. As you know hand-held calculators can only store one variable at a time while the program can store many more making it easier and more convenient to use. The number generator and unit converter are fine but the number generator has no way of extract the numbers from the window to a spreadsheet, other than coying each number one by one.

The revision system is exactly what I wanted for students however, I do understand that the program is at an early stage of development, but there need to be more topics available to students. Otherwise everything is fine and working correctly.

Keep up the good work.

Yours sincerely

R. Lamb

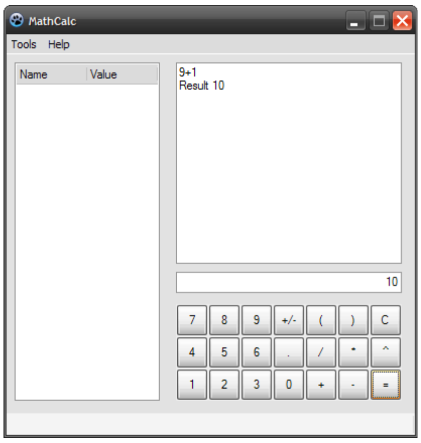
Rob Lamb
Head of Mathematics / Key Skills Coordinator / Link Tutor - High Tunstall

CHARTER MARK
CUSTOMER SERVICE EXCELLENCE

www.hpoolsfc.ac.uk

COMMUNITY RUGBY
ACCREDITED COLLEGE

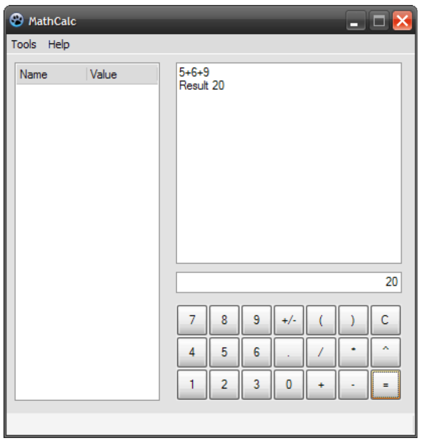
Testing



```
9+1
Result 10
```

10

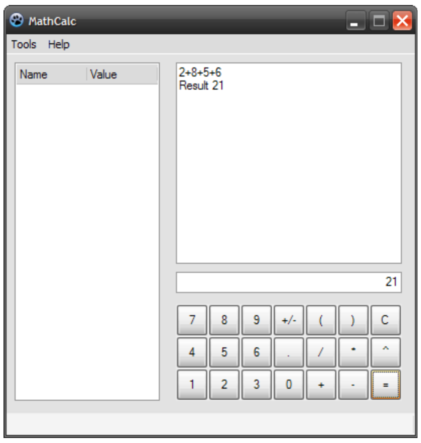
1.1 Basic addition. Shows correct answer in edit-box.



```
5+6+9
Result 20
```

20

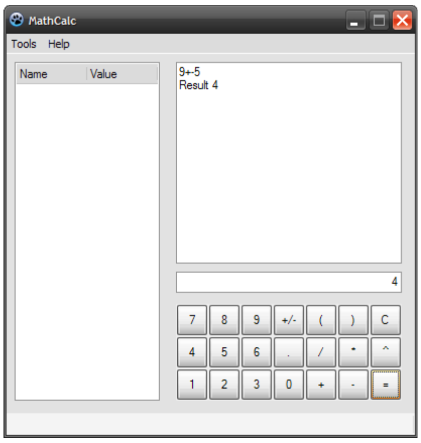
1.2 Advanced addition. Returned sum of numbers.



```
2+8+5+6
Result 21
```

21

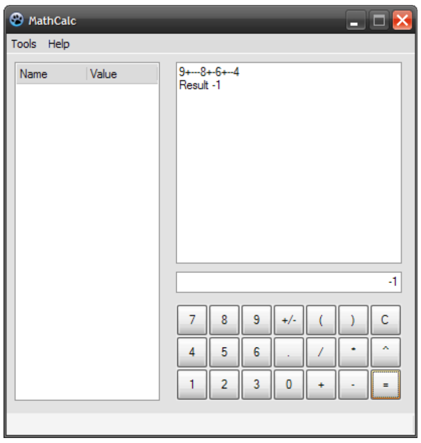
1.3 Complex addition. Sum gives right answer.



```
9+-5
Result 4
```

4

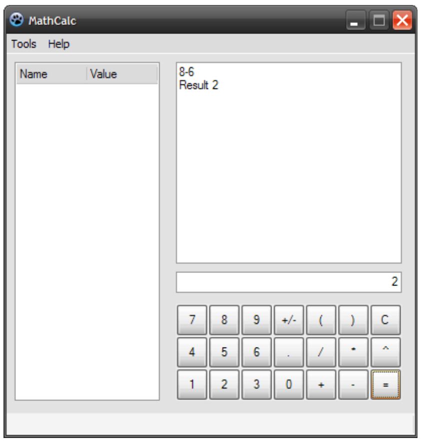
1.4 Negation. Testing reveals simple subtraction works, a fundamental task.



```
9+---8+-6+--4
Result -1
```

-1

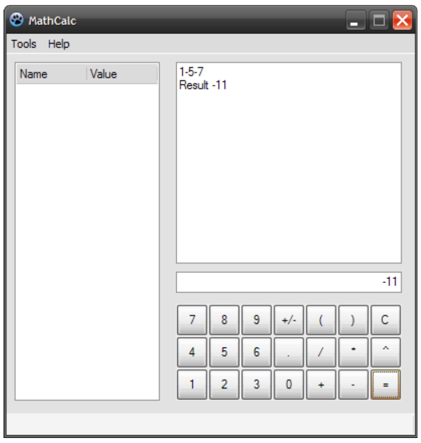
1.5 Complex negation. Minus, minus is correct, testing common occurrence.



```
8-6
Result 2
```

2

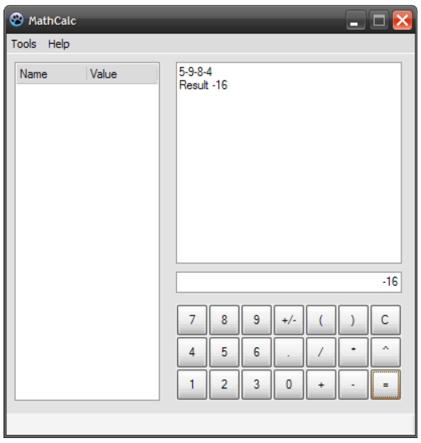
1.6 Basic subtraction. Similar to 1.4 but necessary to see if all possibilities are fine.



```
1-5-7
Result -11
```

-11

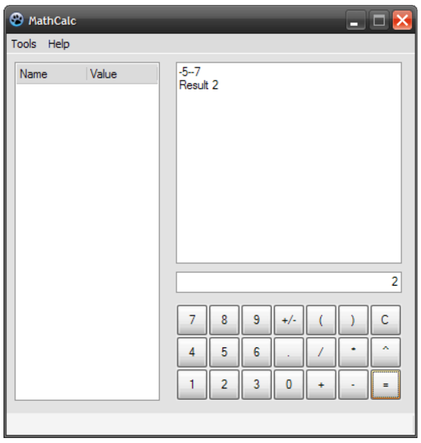
1.7 Advanced subtraction.



```
5-9-8-4
Result -16
```

-16

1.8 Complex subtraction test.



```
-5--7
Result 2
```

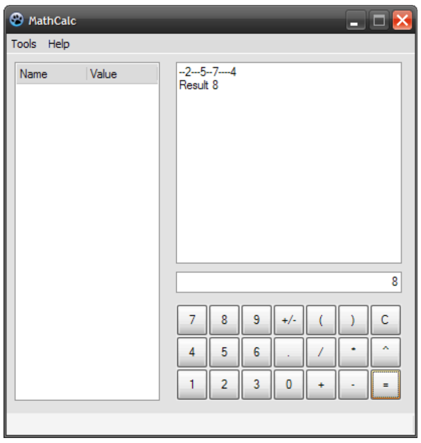
2

1.9 Negation. Another similar

First multiple negation to determine test precedence.

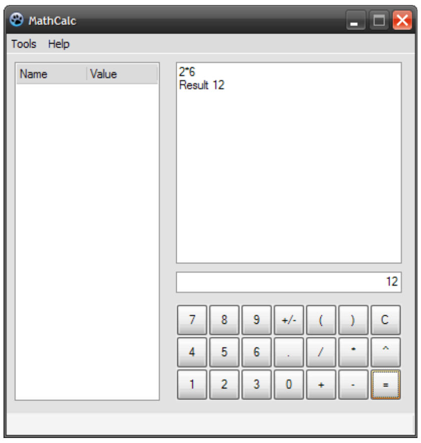
Several minuses for worst case test proves OK.

to 1.4 test if tokenize functions correctly. It does.



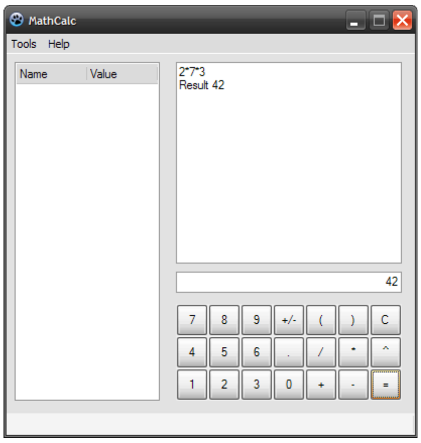
```
--2---5--7----4
Result 8
```

1.10 Complex negation. A calculator gets the same answer, must be correct.



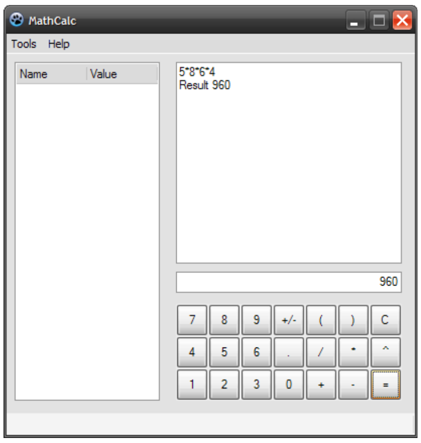
```
2*6
Result 12
```

1.11 Basic multiplication. Determining if multiplication is working.



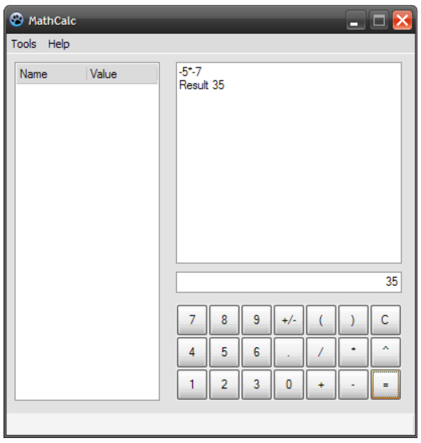
```
2*7*3
Result 42
```

1.12 Advanced multiplication. Another but essential bug finding mission.



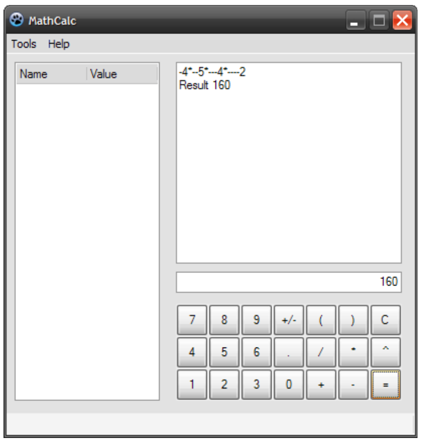
```
5*8*6*4
Result 960
```

1.13 Complex multiplication. A severe case of products that copes fine.



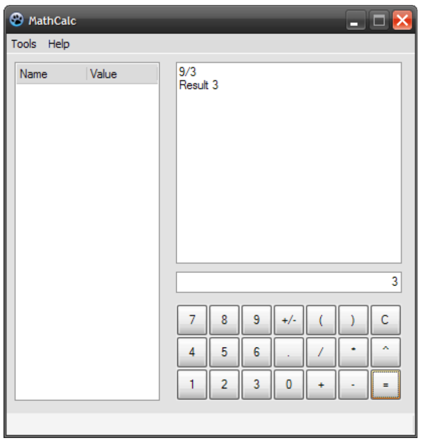
```
-5*-7
Result 35
```

1.14 Negation. Like other tests before it, a systematic method is best.



```
-4*--5*---4*----2
Result 160
```

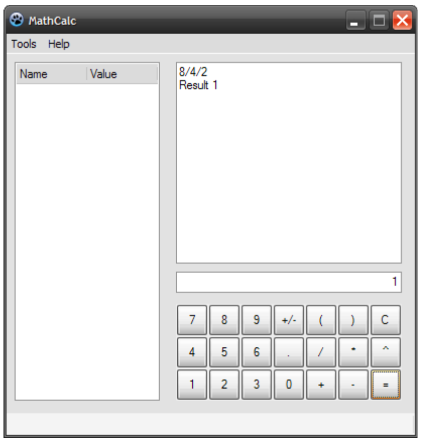
1.15 Complex negation. This screen shot shows multiplication to the max.



```
9/3
Result 3
```

```
3
```

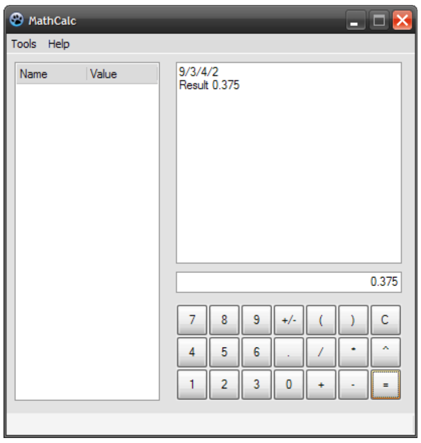
1.16 Basic division. A simple calculation to see if it gets the right answer.



```
8/4/2
Result 1
```

```
1
```

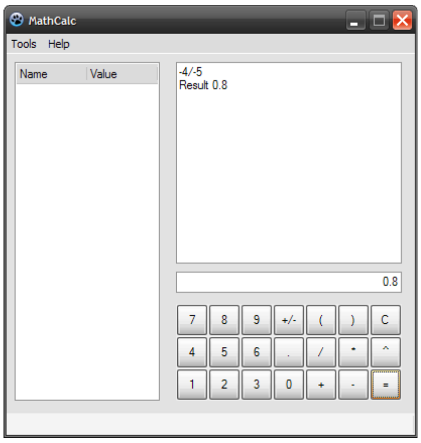
1.17 Advanced division. Sometimes one division is just not enough.



```
9/3/4/2
Result 0.375
```

```
0.375
```

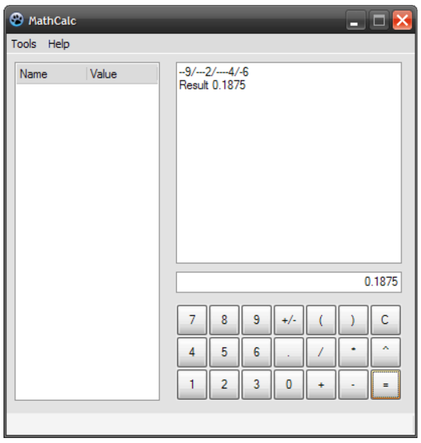
1.18 Complex division. Here is the first floating point result demonstration in action.



```
-4/-5
Result 0.8
```

```
0.8
```

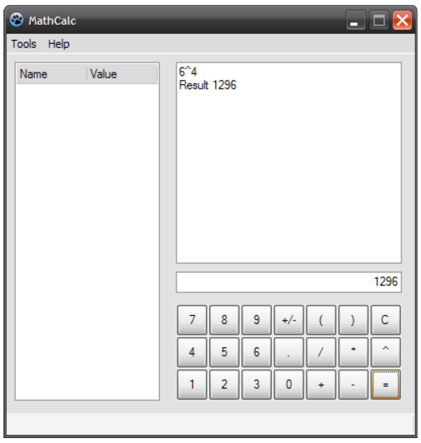
1.19 Negation. Two minuses divide, become positive, maths at work.



```
--9/---2/----4/-6
Result 0.1875
```

```
0.1875
```

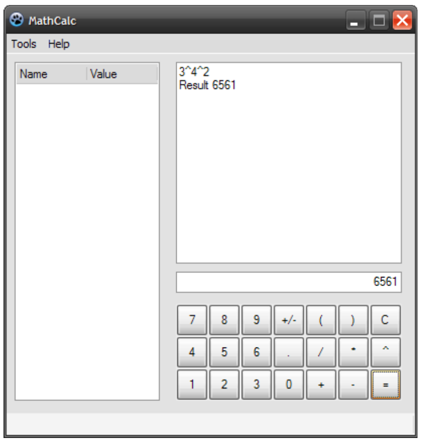
1.20 Complex negation. A thorough test on dividing minuses. Pass!



```
6^4
Result 1296
```

```
1296
```

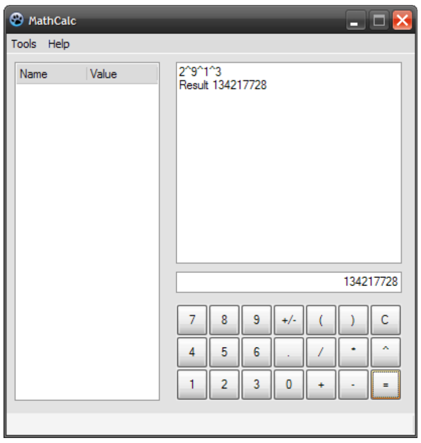
1.21 Basic power. One of the more basic tests show that the answer is right. For now.



```
3^4^2
Result 6561
```

```
6561
```

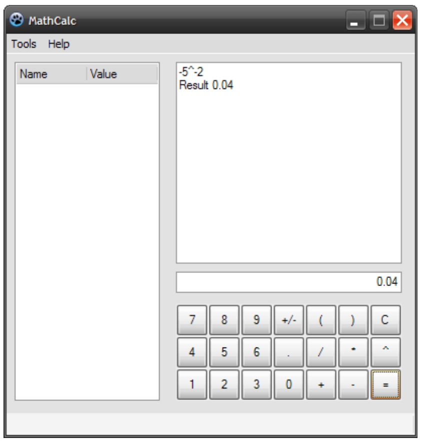
1.22 Advanced power. Oops,



```
2^9^1^3
Result 134217728
```

```
134217728
```

1.23 Complex power. This is



```
-5^-2
Result 0.04
```

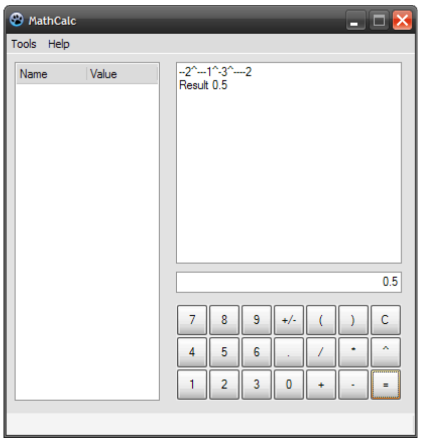
```
0.04
```

1.24 Negation. Negative

the first mistake. Should be 43046721, never mind.

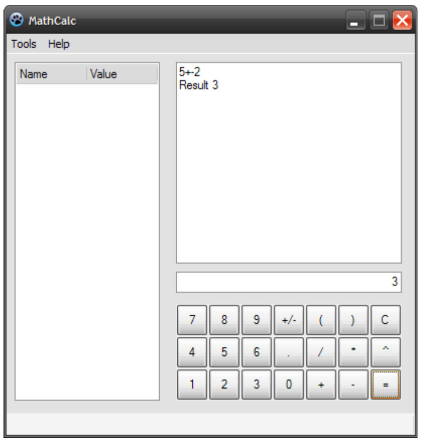
correct, the previous must have been a bad case.

powers seem to be functioning as expected.



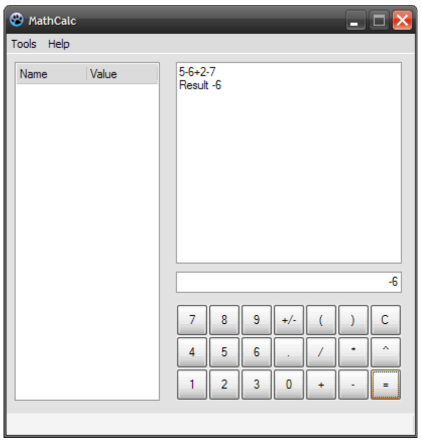
```
--2^---1^-3^----2
Result 0.5
```

1.25 Complex negation. A worse scenario than the previous test.



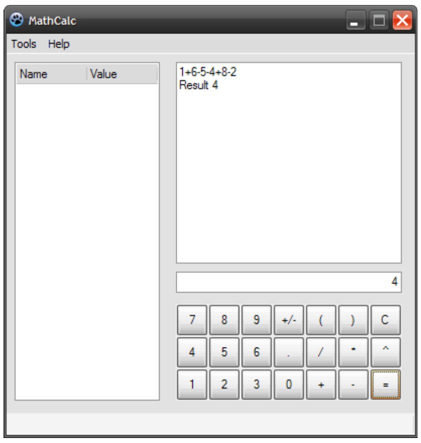
```
5+-2
Result 3
```

1.26 Basic composite. This shows no bugs are in the department of plus or minus.



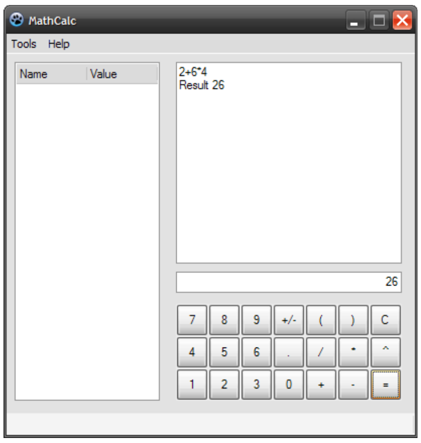
```
5-6+2-7
Result -6
```

1.27 Advanced composite. Some users are demanding, but all cases are covered.



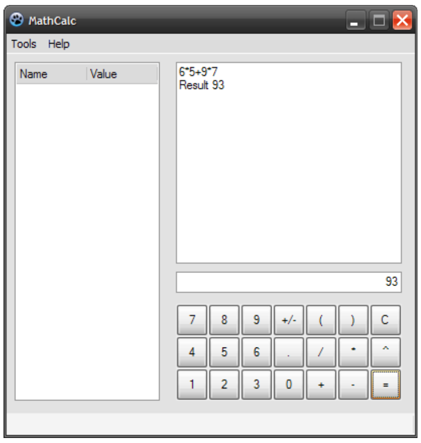
```
1+6-5-4+8-2
Result 4
```

1.28 Complex composite. All is fine here, will the next result be so lucky?



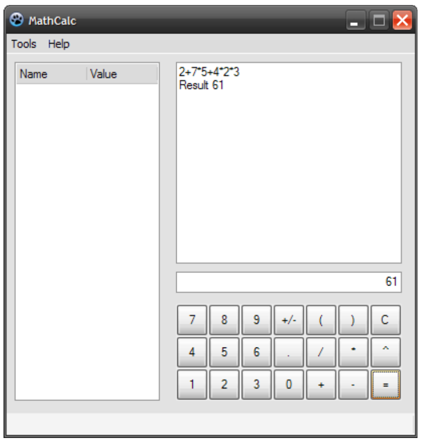
```
2+6*4
Result 26
```

1.29 Basic composite. The union of add and times proves to be a good match.



```
6*5+9*7
Result 93
```

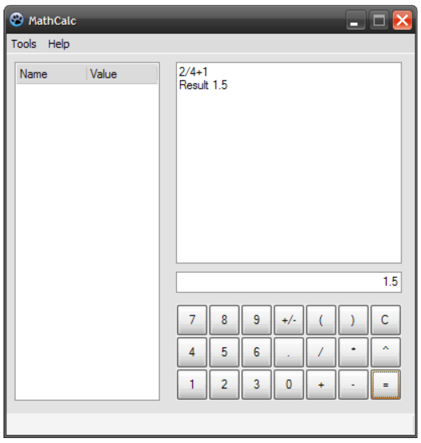
1.30 Advanced composite. Using two operators simultaneously.



```
2+7*5+4*2*3
Result 61
```

```
61
```

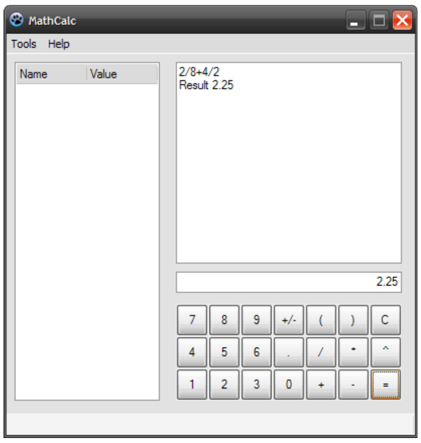
1.31 Complex composite. A bad case at its worst, but the answer disagrees.



```
2/4+1
Result 1.5
```

```
1.5
```

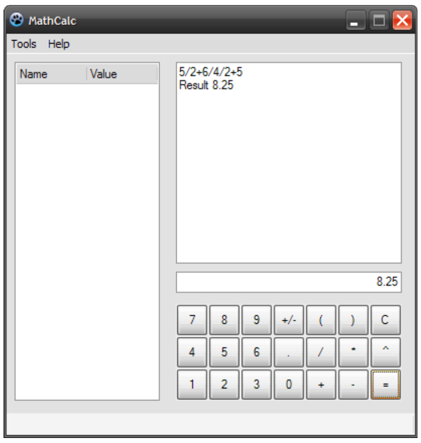
1.32 Basic composite. This example shows that the BIDMAS rule is followed.



```
2/8+4/2
Result 2.25
```

```
2.25
```

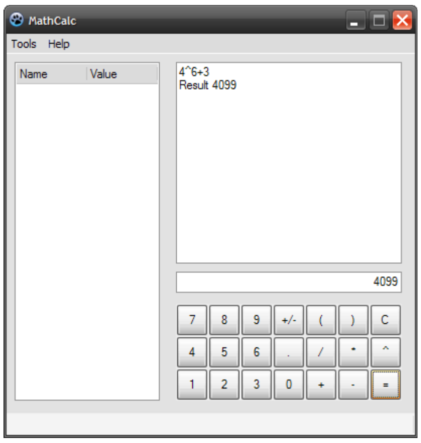
1.33 Advanced composite. Exhausting important operator combinations.



```
5/2+6/4/2+5
Result 8.25
```

```
8.25
```

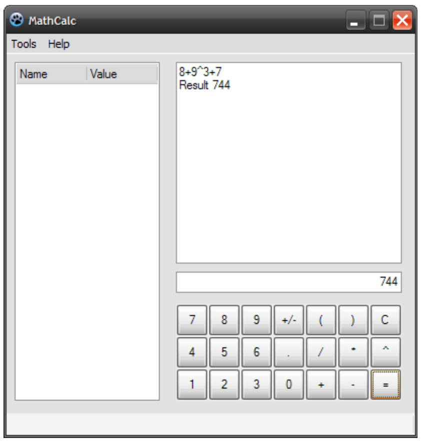
1.34 Complex composite. When the small succeed try a larger string.



```
4^6+3
Result 4099
```

```
4099
```

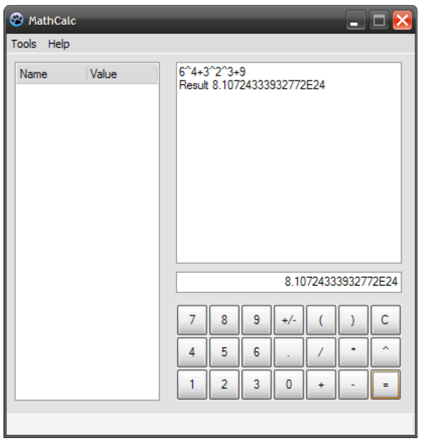
1.35 Basic composite. Back to another combination to see if all is well.



```
8+9^3+7
Result 744
```

```
744
```

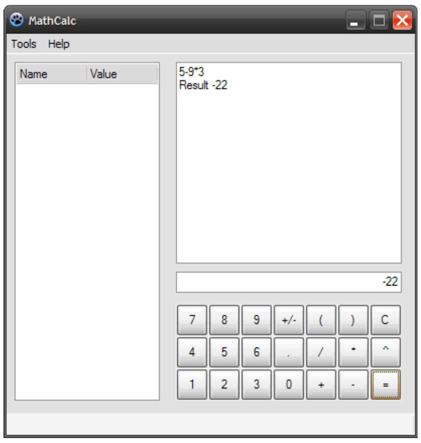
1.36 Advanced composite. A harder strain on the evaluate infix algorithm.



```
6^4+3^2^3+9
Result 8.10724333932772E24
```

```
8.10724333932772E24
```

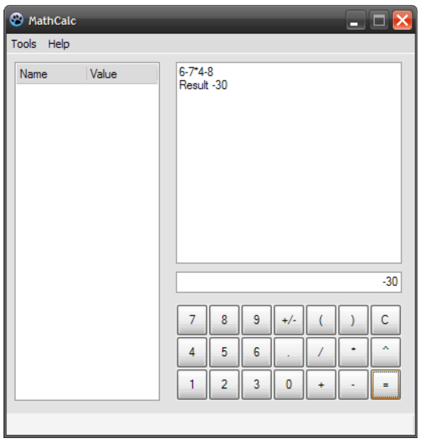
1.37 Complex composite.



```
5-9*3
Result -22
```

```
-22
```

1.38 Basic composite. This



```
6-7*4-8
Result -30
```

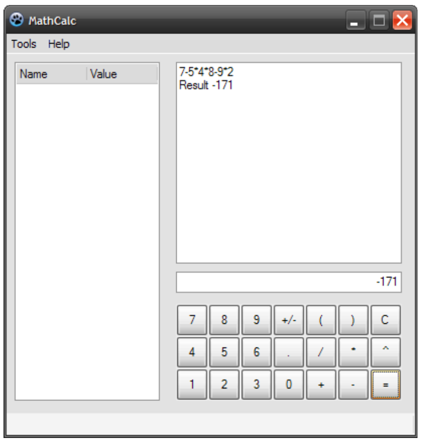
```
-30
```

1.39 Advanced composite.

Another problem revealed that is related to powers.

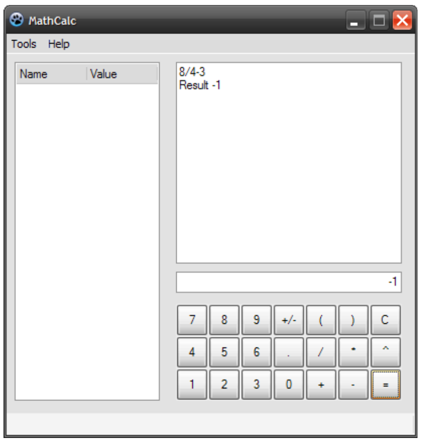
expression has a negative result, but it is correct.

These test seem identical, they examine other aspects.



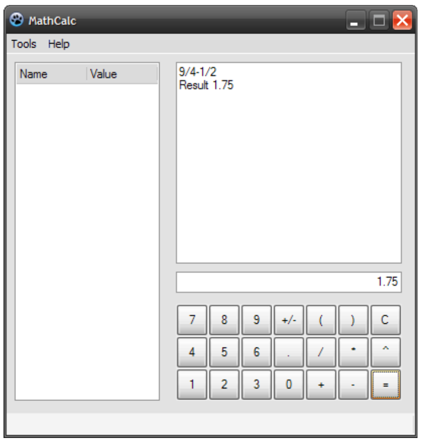
```
7-5*4*8-9*2
Result -171
```

-171



```
8/4-3
Result -1
```

-1



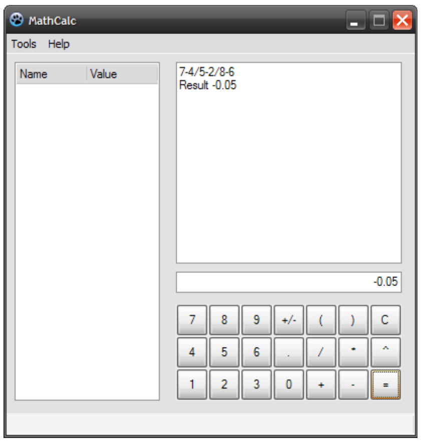
```
9/4-1/2
Result 1.75
```

1.75

1.40 Complex composite. Another answer that cannot be faulted with.

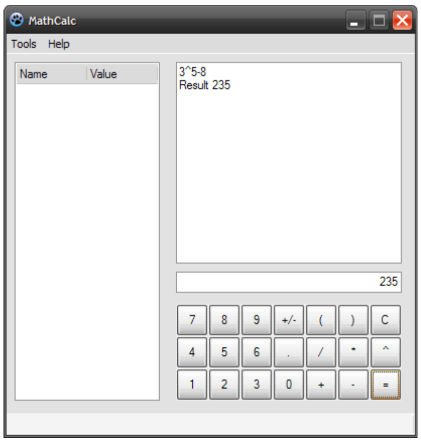
1.41 Basic composite. You would think all combinations have been checked.

1.42 Advanced composite. A more difficult version of the previous test.



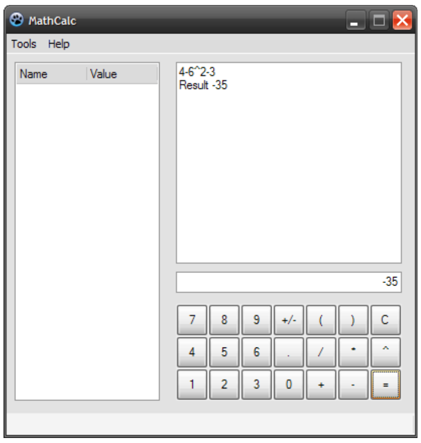
```
7-4/5-2/8-6
Result -0.05
```

-0.05



```
3^5-8
Result 235
```

235



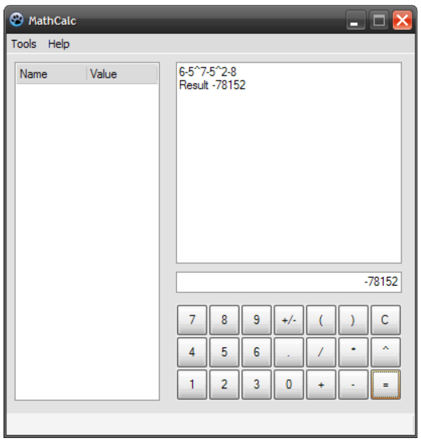
```
4-6^2-3
Result -35
```

-35

1.43 Complex composite. An even more difficult example of mixing and matching.

1.44 Basic composite. Back to simple expressions with a new sequence.

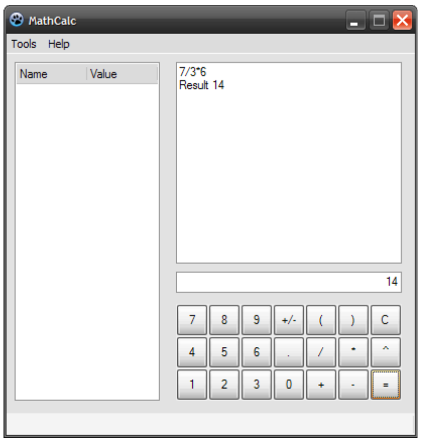
1.45 Advanced composite. These appear to be a breeze for the new system.



```
6-5^7-5^2-8
Result -78152
```

```
-78152
```

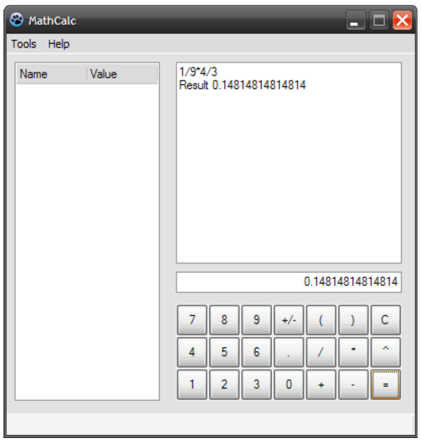
1.46 Complex composite. This examination has produced good results.



```
7/3*6
Result 14
```

```
14
```

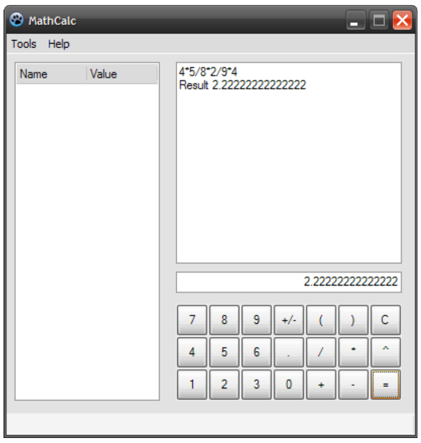
1.47 Basic composite. The start of testing simple times and divide operations.



```
1/9*4/3
Result 0.148148148148148
```

```
0.148148148148148
```

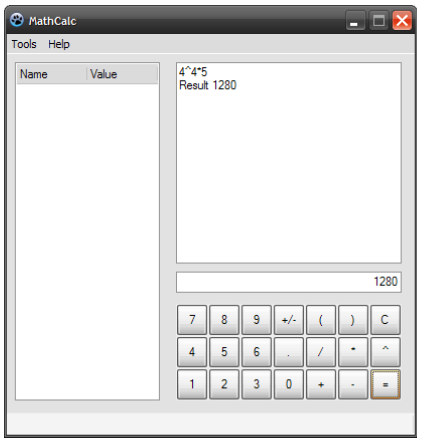
1.48 Advanced composite. In this example there is a recurring decimal answer.



```
4*5/8*2/9*4
Result 2.22222222222222
```

```
2.22222222222222
```

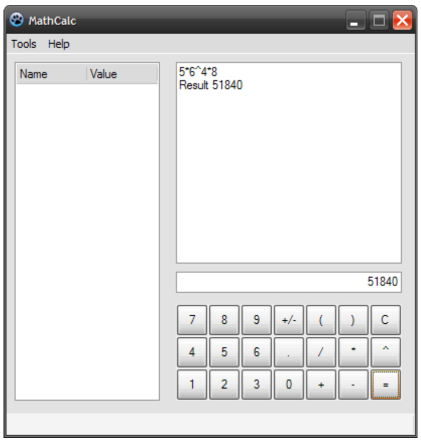
1.49 Complex composite. Another recurring decimal, at least it is correct.



```
4^4*5
Result 1280
```

```
1280
```

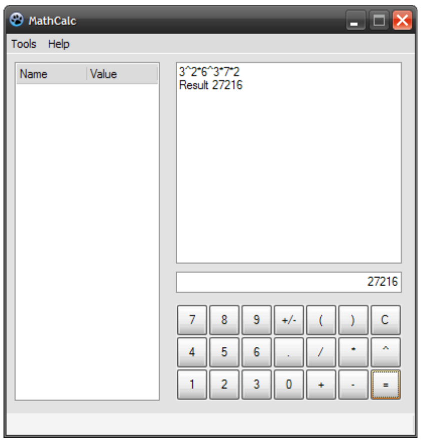
1.50 Basic composite. Powers and multiplication are similar concepts.



```
5*6^4*8
Result 51840
```

```
51840
```

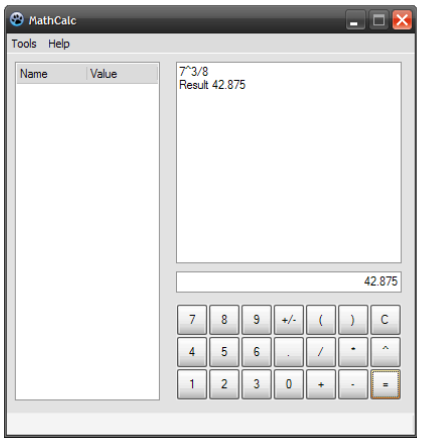
1.51 Advanced composite. Comprehensive testing is needed for all situations.



```
3^2*6^3*7*2
Result 27216
```

```
27216
```

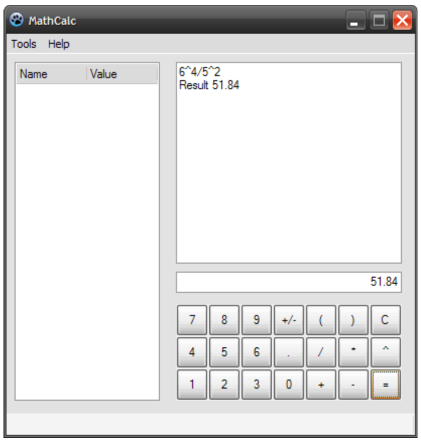
1.52 Complex composite.



```
7^3/8
Result 42.875
```

```
42.875
```

1.53 Basic composite. The



```
6^4/5^2
Result 51.84
```

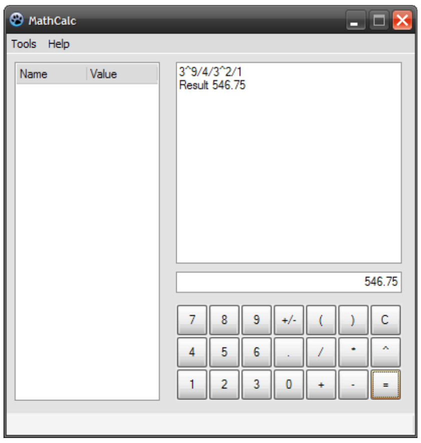
```
51.84
```

1.54 Advanced composite.

The worst circumstances need to be evaluated.

status of the result: answer is accurate, 100%.

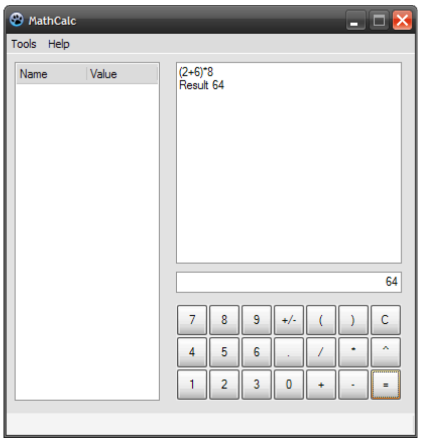
Will a similar expression occur? Safe to be sure.



3^9/4/3^2/1
Result 546.75

546.75

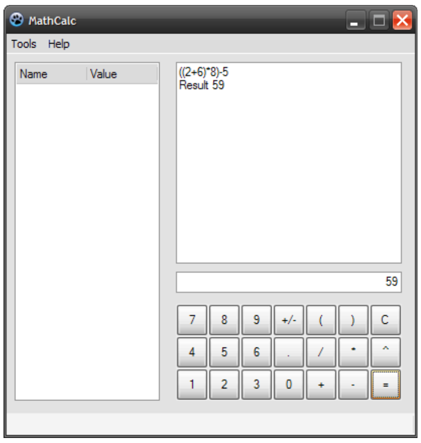
1.55 Complex composite. It is good to make sure the extreme cases are covered.



(2+6)*8
Result 64

64

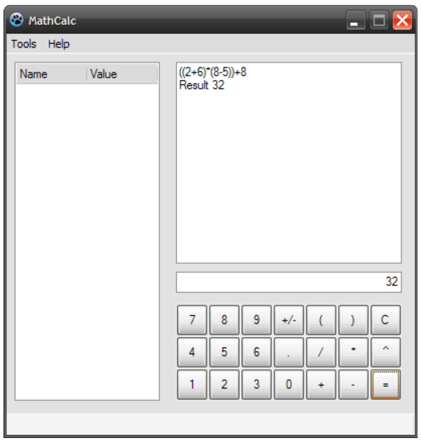
1.56 Basic brackets. Parenthesis is a useful tool that needs to work for users.



((2+6)*8)-5
Result 59

59

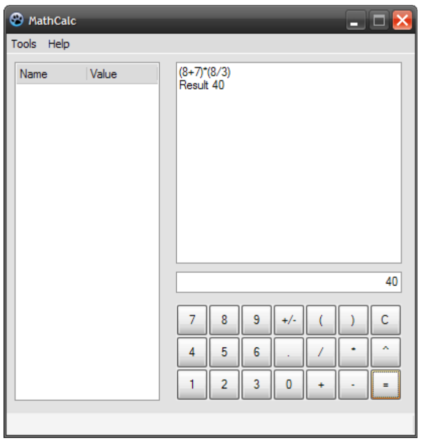
1.57 Advanced brackets. Embedded brackets for the demanding calculations.



((2+6)*(8-5))+8
Result 32

32

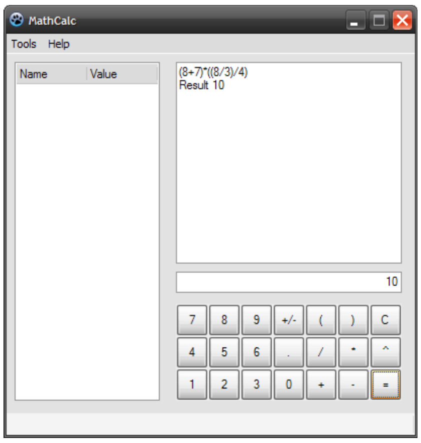
1.58 Complex brackets. Just making sure people get the right answer, they did.



(8+7)*(8/3)
Result 40

40

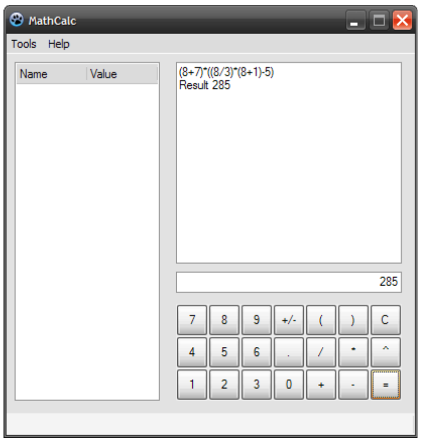
1.59 Simple bracket. Adding an extra operator adds more testing complexity.



(8+7)*((8/3)/4)
Result 10

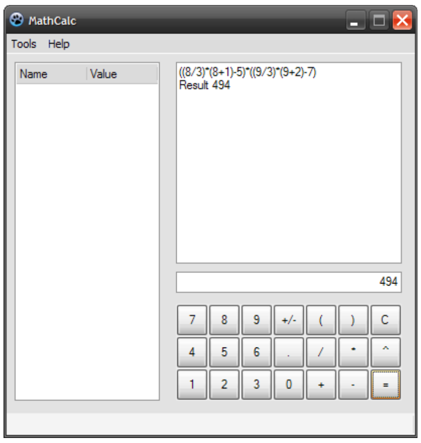
10

1.60 Hard bracket. Some of the larger expressions produce small results.



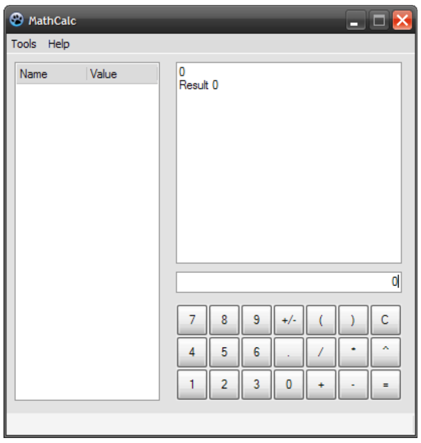
```
(8+7)*((8/3)*(8+1)-5)
Result 285
```

1.61 Tough bracket. This trial run is perhaps the biggest expression of all.



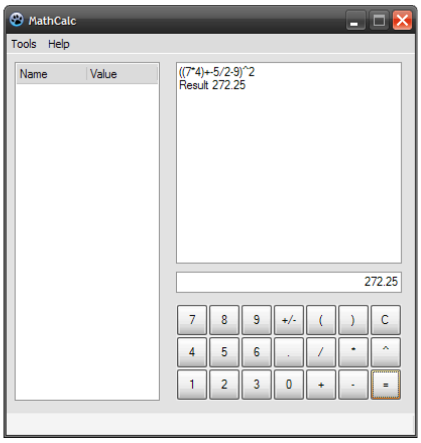
```
((8/3)*(8+1)-5)*((9/3)*(9+2)-7)
Result 494
```

1.62 Very complex bracket. Well, this one is gigantic, unlike the next test.



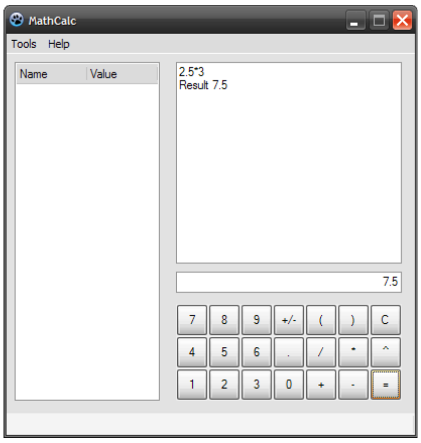
```
0
Result 0
```

1.63 Normal data. Having a positive outcome illustrates that the system can cope.



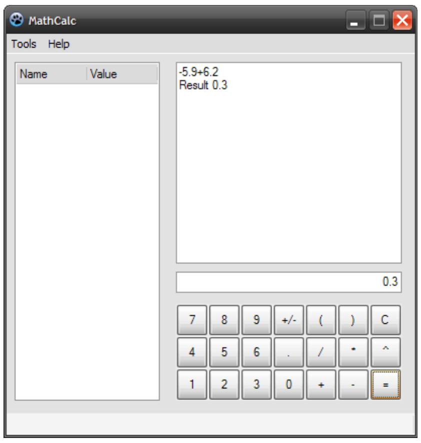
```
((7*4)+-5/2-9)^2
Result 272.25
```

1.64 Normal data. A regular test just to make sure that a normal output is produced.



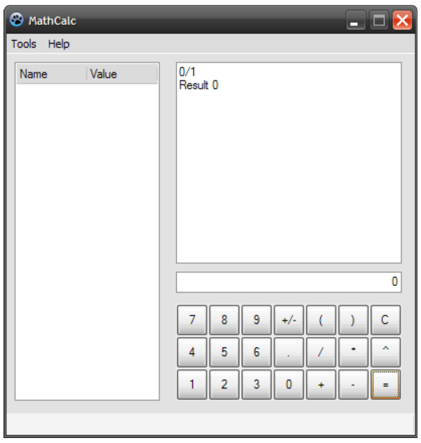
```
2.5*3
Result 7.5
```

1.65 Normal data. Repeating similar tests just to see if it works as expected.

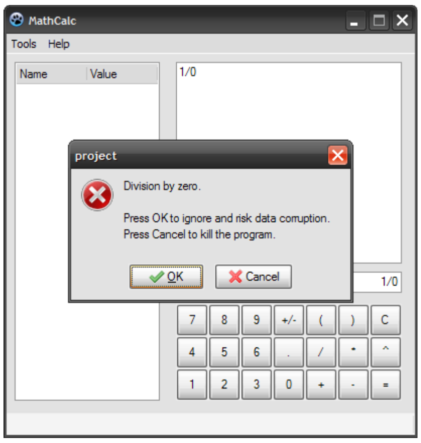


```
-5.9+6.2
Result 0.3
```

1.66 Normal data. More typical data, fortunately the number of bugs is low.



```
0/1
Result 0
```

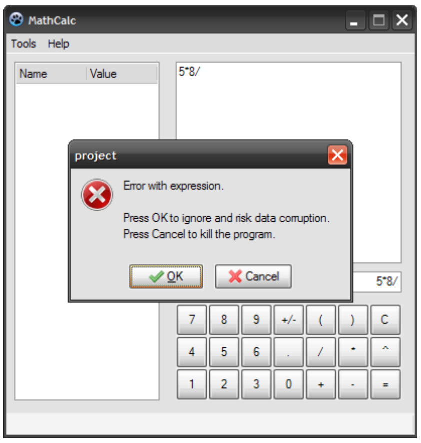


```
1/0
```

project

Division by zero.

Press OK to ignore and risk data corruption.
Press Cancel to kill the program.



```
5*8/
```

project

Error with expression.

Press OK to ignore and risk data corruption.
Press Cancel to kill the program.

1.67 Normal data. Trying to

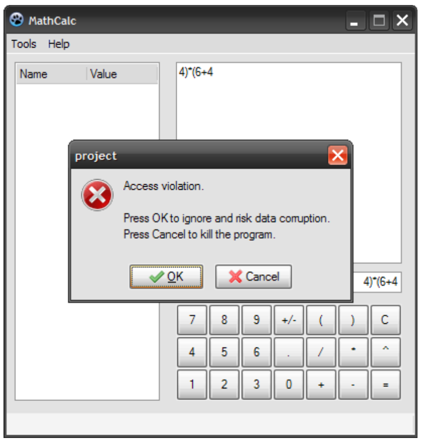
1.68 Error catching. This may

1.69 Error catching. A

see if the logic of the algorithm is stable.

seem that the test has failed, truth is: a success.

Missing operand, error catching working fine here.

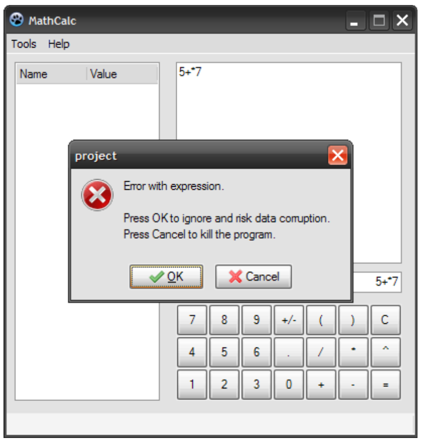


4)*(6+4

Access violation.

Press OK to ignore and risk data corruption.
Press Cancel to kill the program.

1.70 Error catching. Mismatching parenthesis flags up an error.

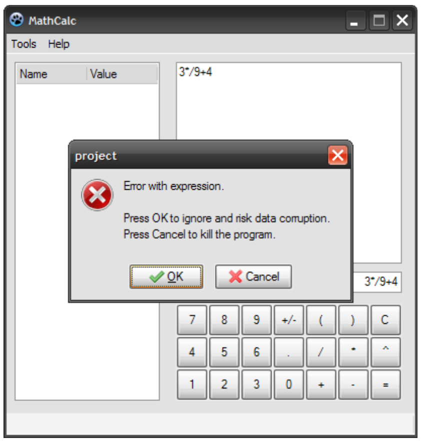


5+*7

Error with expression.

Press OK to ignore and risk data corruption.
Press Cancel to kill the program.

1.71 Error catching. More error detection helps users and developers.

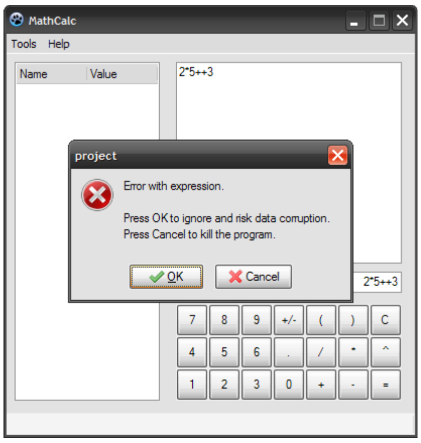


3*/9+4

Error with expression.

Press OK to ignore and risk data corruption.
Press Cancel to kill the program.

1.72 Error catching. Trying all significant operator related mistakes.

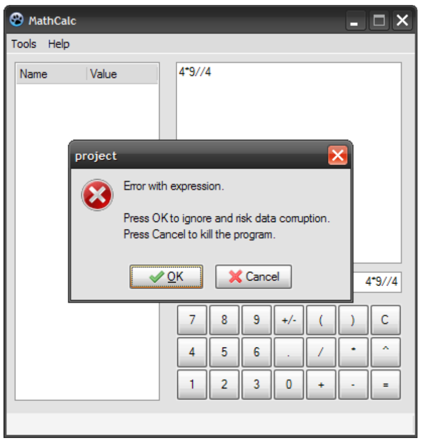


2*5++3

Error with expression.

Press OK to ignore and risk data corruption.
Press Cancel to kill the program.

1.73 Error catching. Plus related error, but this is what is supposed to happen.

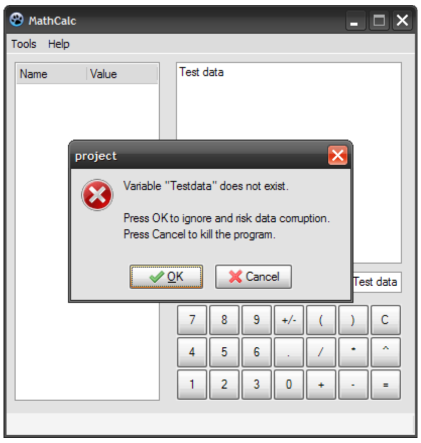


4*9//4

Error with expression.

Press OK to ignore and risk data corruption.
Press Cancel to kill the program.

1.74 Error catching. The story of the missing operand has a good ending.

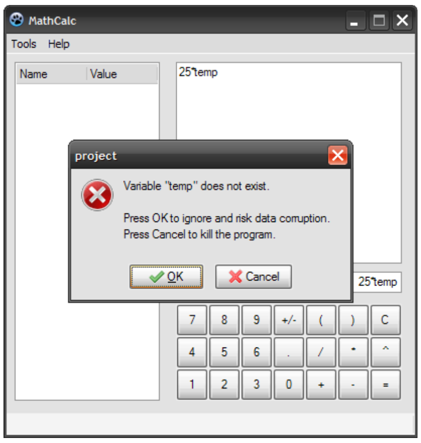


Test data

Variable "Testdata" does not exist.

Press OK to ignore and risk data corruption.
Press Cancel to kill the program.

1.75 Exceptional data. The other acute cases when no expression is entered.

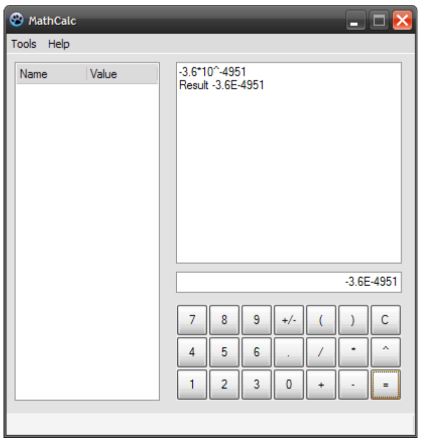


25*temp

Variable "temp" does not exist.

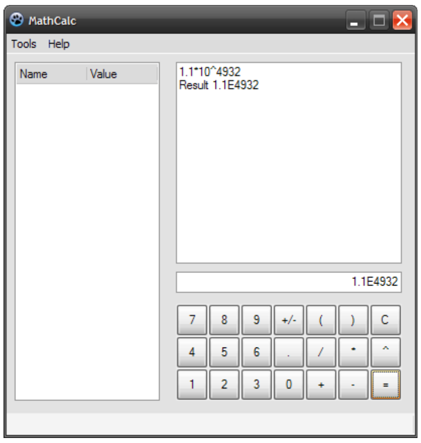
Press OK to ignore and risk data corruption.
Press Cancel to kill the program.

25*temp



-3.6*10^-4951
Result -3.6E-4951

-3.6E-4951



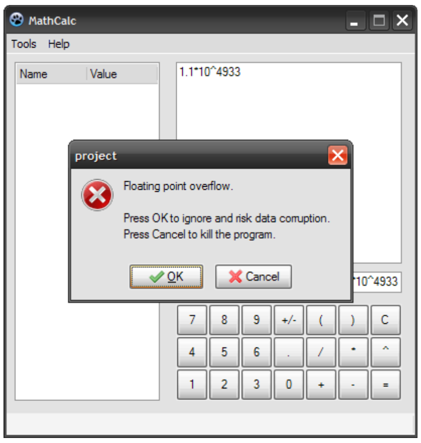
1.1*10^4932
Result 1.1E4932

1.1E4932

1.76 Erroneous data. Determining if the variable class is doing its job.

1.77 Extreme data. Some numbers are just not small enough to be useful.

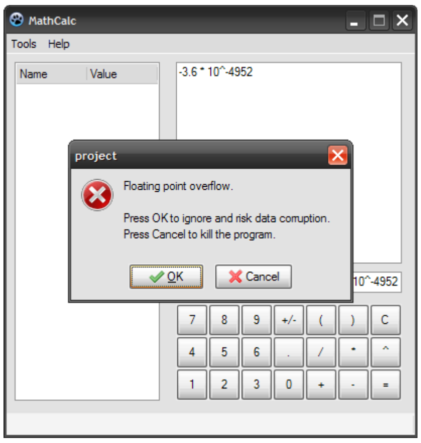
1.78 Extreme data. While others are unpronounceable, but are necessary.



1.1*10^4933

Floating point overflow.

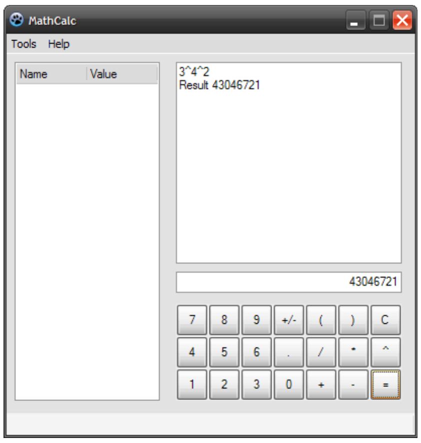
Press OK to ignore and risk data corruption.
Press Cancel to kill the program.



-3.6 * 10^-4952

Floating point overflow.

Press OK to ignore and risk data corruption.
Press Cancel to kill the program.



3^4^2
Result 43046721

43046721

1.79 Boundary data. Overflows are rare but must be handled with care.

1.80 Boundary data. An example of an overflow, does not affect most users.

1.81 Algorithm correction. Here is the correction made from test 1.37.

Project Code

Main Program

```
program project;

{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Interfaces, // this includes the LCL widgetset
  Forms, Main, LResources, UnitConvert, UnitEdit,
  About, NumberGen, Graph, Calc, MathTopic;

{$IFDEF WINDOWS}{$R project.rc}{$ENDIF}
```

```
begin
  {$I project.lrs}
  Application.Initialize;
  Application.CreateForm(TMainForm, MainForm);
  Application.CreateForm(TUnitConvertForm, UnitConvertForm);
  Application.CreateForm(TNumberGenForm, NumberGenForm);
  Application.CreateForm(TAboutForm, AboutForm);
  Application.CreateForm(TGraphForm, GraphForm);
  Application.Run;
end.
```

Parser Code

```
unit Parser;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, Math;

type
  TStringListPointer = ^TStringList;

type
  ErrorMsg = class(Exception);

  TVariableItem = ^Item;
  Item = record
    VarName: string;
    VarResult: extended;
    Next: TVariableItem;
  end;

  TStackItem = ^Node;
  Node = record
    AOperator: char;
    AOperand: extended;
    Next: TStackItem;
  end;

  TVariable = class
    private
      FirstVariable: TVariableItem;
      LastVariable: TVariableItem;
      function IsDuplicateVariable(const VarName: string): boolean;
    public
      constructor Create;
      destructor Free;
      procedure ChangeVariable(VarName: string; VarResult: extended);
      function GetVariable(VarName: string): extended;
      procedure AddVariable(VarName: string; VarResult: extended);
      procedure DelVariable(VarName: string);
      function GetAllVariables: TStringListPointer;
    end;

  TStack = class
    private
      TopOfStack: TStackItem;
    public
      constructor Create;
      destructor Free;
      function IsEmpty: boolean;
      procedure Push(NewItem: char); overload;
      procedure Push(NewItem: extended); overload;
      procedure Pop;
      function GetOperand: extended;
      function GetOperator: char;
      function Count: integer;
    end;

  TParser = class
    private
      Stack: TStack;
      function Tokenise(Expression: string): string;
    public
      Variable: TVariable;
      constructor Create;
      destructor Free;
      function InfixToPostfix(Expression: string): string;
      function EvalPostfix(Postfix: string): extended;
      function EvalInfix(Postfix: string): extended;
    end;

implementation

{ TStack }
{--------------------------------------------------------------------------------}
constructor TStack.Create;
begin
  TopOfStack:=nil;
end;
{--------------------------------------------------------------------------------}
destructor TStack.Free;
begin
  while not(IsEmpty) do Pop;
end;
{--------------------------------------------------------------------------------}
function TStack.IsEmpty: boolean;
begin
  Result:=TopOfStack=nil;
end;
{--------------------------------------------------------------------------------}
procedure TStack.Push(NewItem: char); overload;
var NewNode: TStackItem;
begin
  New(NewNode);
  NewNode^.AOperator:=NewItem;
  NewNode^.Next:=TopOfStack;
  TopOfStack:=NewNode;
end;
{--------------------------------------------------------------------------------}
procedure TStack.Push(NewItem: extended); overload;
var NewNode: TStackItem;
begin
  New(NewNode);
  NewNode^.AOperand:=NewItem;
  NewNode^.Next:=TopOfStack;
  TopOfStack:=NewNode;
end;
{--------------------------------------------------------------------------------}
procedure TStack.Pop;
var temp: TStackItem;
begin
  temp:=TopOfStack;
  TopOfStack:=TopOfStack^.Next;
  Dispose(temp);
end;
{--------------------------------------------------------------------------------}
function TStack.GetOperand: extended;
```

```
begin
  Result:=TopOfStack^.AOperand;
end;
{-------------------------------------------------------------------------------}
function TStack.GetOperator: char;
begin
  Result:=TopOfStack^.AOperator;
end;
{-------------------------------------------------------------------------------}
function TStack.Count: integer;
var temp: TStackItem;
begin
  temp:=TopOfStack;
  Count:=0;
  while not(temp=nil) do
    begin
      temp:=temp^.Next;
      Inc(Count);
    end;
end;
{-------------------------------------------------------------------------------}

{ TVariable }
{-------------------------------------------------------------------------------}
constructor TVariable.Create;
begin
  FirstVariable:=nil;
  LastVariable:=nil;
end;
{-------------------------------------------------------------------------------}
destructor TVariable.Free;
var CurrentItem: TVariableItem;
begin
  CurrentItem:=FirstVariable;
  while CurrentItem<>nil do
    begin
      Dispose(CurrentItem);
      CurrentItem:=CurrentItem^.Next;
    end;
end;
{-------------------------------------------------------------------------------}
procedure TVariable.ChangeVariable(VarName: string; VarResult: extended);
var CurrentItem: TVariableItem;
begin
  CurrentItem:=FirstVariable;
  while CurrentItem<>nil do
    begin
      if SameText(VarName,CurrentItem^.VarName) then
        begin
          CurrentItem^.VarResult:=VarResult;
          Break;
        end;
      CurrentItem:=CurrentItem^.Next;
    end;
end;
{-------------------------------------------------------------------------------}
function TVariable.GetVariable(VarName: string): extended;
var CurrentItem: TVariableItem;
begin
  CurrentItem:=FirstVariable;
  while CurrentItem<>nil do
    begin
      if SameText(VarName,CurrentItem^.VarName) then
        begin
          Result:=CurrentItem^.VarResult;
          Exit;
        end;
      CurrentItem:=CurrentItem^.Next;
    end;
  raise ErrorMsg.Create('Variable "'+VarName+'" does not exist');
end;
{------------------------------------------------------------------------------}
procedure TVariable.AddVariable(VarName: string; VarResult: extended);
var NewVariable: TVariableItem;
begin
  if IsDuplicateVariable(VarName) then Exit;
  New(NewVariable);
  NewVariable^.VarName:=VarName;
  NewVariable^.VarResult:=VarResult;
  NewVariable^.Next:=nil;

  if FirstVariable=nil then
    begin
      FirstVariable:=NewVariable;
      LastVariable:=NewVariable;
    end
  else
    begin
      LastVariable^.Next:=NewVariable;
      LastVariable:=NewVariable;
    end;
end;
{------------------------------------------------------------------------------}
procedure TVariable.DelVariable(VarName: string);
var CurrentItem, PrevItem: TVariableItem;
begin
  CurrentItem:=FirstVariable;
  PrevItem:=nil;
  while CurrentItem<>nil do
    begin
      if SameText(VarName,CurrentItem^.VarName) then
        begin
          if PrevItem=nil then FirstVariable:=CurrentItem^.Next
          else PrevItem^.Next:=CurrentItem^.Next;
          if CurrentItem=LastVariable then
            begin
              if PrevItem<>nil then LastVariable:=PrevItem;
              LastVariable^.Next:=nil;
            end;
          Dispose(CurrentItem);
          Exit;
        end;
      PrevItem:=CurrentItem;
      CurrentItem:=CurrentItem^.Next;
    end;
end;
{------------------------------------------------------------------------------}
function TVariable.IsDuplicateVariable(const VarName: string): boolean;
var CurrentItem: TVariableItem;
begin
  CurrentItem:=FirstVariable;
  while CurrentItem<>nil do
    begin
      if SameText(VarName,CurrentItem^.VarName) then
        begin
          Result:=true;
          Exit;
        end;
      CurrentItem:=CurrentItem^.Next;
    end;
  Result:=false;
end;
```

```
{------------------------------------------------------------------------------}
function TVariable.GetAllVariables: TStringListPointer;
var
  CurrentItem: TVariableItem;
  List: TStringList;
begin
  List:=TStringList.Create;
  CurrentItem:=FirstVariable;
  while CurrentItem<>nil do
    begin
      List.Add(CurrentItem^.VarName);
      List.Add(FloatToStr(CurrentItem^.VarResult));
      CurrentItem:=CurrentItem^.Next;
    end;
  Result:=@List;
end;
{------------------------------------------------------------------------------}

{ TParser }
{------------------------------------------------------------------------------}
constructor TParser.Create;
begin
  Variable:=TVariable.Create;
  Stack:=TStack.Create;
end;
{------------------------------------------------------------------------------}
destructor TParser.Free;
begin
  Variable.Free;
  Stack.Free;
end;
{------------------------------------------------------------------------------}
function TParser.Tokenise(Expression: string): string;
var
  i, k, max, Minus: integer;
  temp: string;
  LastOperator: char;
begin
  temp:='';
  for i:=1 to Length(Expression) do
    if Expression[i]<>' ' then temp:=temp+Expression[i];
  Result:='';
  expression:=temp;
  i:=0;
  Minus:=0;
  LastOperator:=Chr(0);
  max:=Length(Expression);
  if Expression[1]='-' then
    begin
      Inc(i);
      Inc(Minus);
      Result:='(~';
    end;
  while i<max do
    begin
      Inc(i);
      if Expression[i] in['+','-','*','/','(',')','^'] then
        begin
          if Expression[i]='-' then
            begin
              if Expression[i+1]='-' then
                begin
                  k:=2;
                  Inc(i,2);
                  while Expression[i]='-' do
                    begin
                      Inc(i);
                      Inc(k);
                    end;
                  Dec(i);
                  if (k mod 2 <> 0) then
                    begin
                      if LastOperator =' ' then Result:=Result+'+ (~'
                      else
                        begin
                          Result:=Result+'(~';
                          LastOperator:='-';
                        end;
                      Inc(Minus);
                    end;
                end
              else if Expression[i-1] in['+','-','*','/','^'] then
                begin
                  Result:=Result+'(~';
                  Inc(Minus);
                end
              else if (Expression[i]='-') and (Expression[i-1] in['0'..'9','.'])
then
                begin
                  Result:=Result+'+(~';
                  Inc(Minus);
                end
              else Result:=Result+Expression[i]
            end
          else
            begin
              Result:=Result+Expression[i];
              LastOperator:=Expression[i];
            end;
        end
      else if Expression[i] in['0'..'9','.'] then
        begin
          if not(Expression[i+1] in['0'..'9','.']) and (Minus>0) then
            begin
              Result:=Result+Expression[i]+')';
              Dec(Minus);
            end
          else
            begin
              while Expression[i] in['0'..'9','.'] do
                begin
                  Result:=Result+Expression[i];
                  Inc(i);
                end;
              Dec(i);
            end;
          LastOperator:=' ';
        end
      else
        begin
          temp:='';
          while not(Expression[i] in['+','-','*','/','(',')','^','.']) do
            begin
              if i>max then Break;
              temp:=temp+Expression[i];
              Inc(i);
            end;
            Dec(i);
          temp:=FloatToStr(Variable.GetVariable(temp));
          if temp[1]='-' then
            begin
              temp[1]:='~';
```

```
              Result:=Result+'('+temp+')';
            end
          else Result:=Result+temp;
        end;
      end;

    while Minus>0 do
      begin
        Result:=Result+')';
        Dec(Minus);
      end;
end;
{-------------------------------------------------------------------------------}
function TParser.InfixToPostfix(Expression: string): string;
const Bidmas: string = '~^/*+-';
var
  postfix: string = '';
  i, j, StackPrecedence, ExpressionPrecedence, BracketCount: integer;
begin
  Expression:=Tokenise(Expression);
  i:=0;
  BracketCount:=0;
  while i<= Length(Expression) do
    begin
      Inc(i);
      case Expression[i] of
        '0'..'9','.': begin
        if Expression[i+1]in['0'..'9','.'] then
          begin
            while Expression[i] in['0'..'9','.'] do
              begin
                postfix:=postfix+Expression[i];
                Inc(i);
              end;
            Dec(i);
          end
         else postfix:=postfix+Expression[i];
            postfix:=postfix+' ';
        end;
        '(': begin Stack.Push(Expression[i]); inc(BracketCount); end;
        ')':begin
              Dec(BracketCount);
              while Stack.GetOperator<>'(' do
                begin
                  postfix:=postfix+Stack.GetOperator+' ';
                  Stack.Pop;
                end;
              Stack.Pop;
              if BracketCount=0 then
                while not stack.IsEmpty do
                  begin
                    postfix:=postfix+Stack.GetOperator+' ';
                    stack.Pop;
                  end;
            end;
        '+','-','*','/','^','~': begin
          if Stack.IsEmpty then Stack.Push(Expression[i])
          else
            begin
              for j:=1 to 6 do
                if Expression[i]=Bidmas[j] then ExpressionPrecedence:=j;
              for j:=1 to 6 do
                if Stack.GetOperator=Bidmas[j] then StackPrecedence:=j;
              if (Expression[i]='^') and (Stack.GetOperator='^') then
                begin
                  while (StackPrecedence>ExpressionPrecedence) and (not
Stack.IsEmpty) do
                    begin
                      if Stack.GetOperator='(' then Break;
                      postfix:=postfix+Stack.GetOperator+' ';
                      Stack.Pop;
                      if (Stack.IsEmpty) or (Stack.GetOperator='(') then Break
                      else
                      for j:=1 to 6 do
                        if Stack.GetOperator=Bidmas[j] then StackPrecedence:=j;
                    end;
                end
              else
                begin
                  while (StackPrecedence<=ExpressionPrecedence) and (not
Stack.IsEmpty) do
                    begin
                      if Stack.GetOperator='(' then Break;
                      postfix:=postfix+Stack.GetOperator+' ';
                      Stack.Pop;
                      if (Stack.IsEmpty) or (Stack.GetOperator='(') then Break
                      else
                      for j:=1 to 6 do
                        if Stack.GetOperator=Bidmas[j] then StackPrecedence:=j;
                    end;
                end;
                Stack.Push(Expression[i]);
            end;
        end;
    end;
  end;

  while not Stack.IsEmpty do
    begin
      postfix:=postfix+Stack.GetOperator+' ';
      Stack.Pop;
    end;
  Result:=Postfix;
end;
{------------------------------------------------------------------------------}
function TParser.EvalPostfix(Postfix: string): extended;
var
  i: integer;
  temp1, temp2: extended;
  tempstr: string;
begin
  i:=0;
  while i<Length(Postfix) do
    begin
      Inc(i);
      if Postfix[i] in['0'..'9'] then
        begin
          tempstr:='';
          while Postfix[i]<>' ' do
            begin
              tempstr:=tempstr+Postfix[i];
              Inc(i);
            end;
          Stack.Push(StrToFloat(tempstr));
        end
      else if Postfix[i] in['+','-','*','/','^','~'] then
        begin
          temp1:=Stack.GetOperand;
          Stack.Pop;
          if (Stack.Count>=1) and (Postfix[i]<>'~') then
            begin
              temp2:=Stack.GetOperand;
              Stack.Pop;
            end
          else if Postfix[i]<>'~' then raise ErrorMsg.Create('Error with
expression');
          case Postfix[i] of
            '+': Stack.Push(temp2+temp1);
            '-': Stack.Push(temp2-temp1);
            '*': Stack.Push(temp2*temp1);
            '/': Stack.Push(temp2/temp1);
            '^': Stack.Push(Power(temp2,temp1));
            '~': Stack.Push(temp1*-1);
          end;
        end;
    end;
  Result:=Stack.GetOperand;
  while not Stack.IsEmpty do Stack.Pop;
end;
{------------------------------------------------------------------------------}
function TParser.EvalInfix(Postfix: string): extended;
begin
  Postfix:=InfixToPostfix(Postfix);
  Result:=EvalPostfix(Postfix);
end;
{------------------------------------------------------------------------------}
end.
```

Main Form Code

```
unit Main;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, LResources, Forms, Controls, Dialogs, Menus, ComCtrls,
  ExtCtrls, Calc, About, NumberGen, UnitConvert, MathTopic, Graph;

type

  { TMainForm }

  TMainForm = class(TForm)
    CalcFrame: TCalcFrame;
    MainMenu: TMainMenu;
    MathTopicFrame: TMathTopicFrame;
    MenuGraph: TMenuItem;
    MenuRevise: TMenuItem;
    MenuNumGen: TMenuItem;
    MenuAbout: TMenuItem;
    MenuHelp: TMenuItem;
    MenuTools: TMenuItem;
    MenuUnitConv: TMenuItem;
    StatusBar: TStatusBar;
    procedure MenuAboutClick(Sender: TObject);
    procedure MenuGraphClick(Sender: TObject);
    procedure MenuNumGenClick(Sender: TObject);
    procedure MenuReviseClick(Sender: TObject);
    procedure MenuUnitConvClick(Sender: TObject);
  end;

var
  MainForm: TMainForm;

implementation
```

```
{ TMainForm }
{------------------------------------------------------------------------------}
procedure TMainForm.MenuUnitConvClick(Sender: TObject);
begin
  UnitConvertForm.Show;
end;
{------------------------------------------------------------------------------}
procedure TMainForm.MenuAboutClick(Sender: TObject);
begin
  AboutForm.ShowModal;
end;
{------------------------------------------------------------------------------}
procedure TMainForm.MenuGraphClick(Sender: TObject);
begin
  GraphForm.Show;
end;
{------------------------------------------------------------------------------}
procedure TMainForm.MenuNumGenClick(Sender: TObject);
begin
  NumberGenForm.Show;
end;
{------------------------------------------------------------------------------}
procedure TMainForm.MenuReviseClick(Sender: TObject);
begin
  if CalcFrame.Visible then
    begin
      MainForm.Width:=MathTopicFrame.Width+8;
      MainForm.Height:=MathTopicFrame.Height+StatusBar.Height+24;
      MenuRevise.Caption:='Calculator';
    end
  else
    begin
      MainForm.Width:=CalcFrame.Width+8;
      MainForm.Height:=CalcFrame.Height+StatusBar.Height+32;
      MenuRevise.Caption:='Revise Topic';
    end;
  CalcFrame.Visible:=not CalcFrame.Visible;
  MathTopicFrame.Visible:=not MathTopicFrame.Visible;
end;
{------------------------------------------------------------------------------}

initialization
  {$I main.lrs}

end.
```

Calculator Frame Code

```
unit Calc;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, LResources, Forms, StdCtrls, ExtCtrls, StrUtils, Controls,
  Dialogs, Grids, Menus, Parser;

type

  { TCalcFrame }

  TCalcFrame = class(TFrame)
    InputButton1: TButton;
    InputButton0: TButton;
    InputButtonPower: TButton;
    InputButtonOpenBracket: TButton;
    InputButtonDivide: TButton;
    InputButtonCloseBracket: TButton;
    InputButtonEquals: TButton;
    InputButtonClear: TButton;
    InputButtonTimes: TButton;
    InputButtonMinus: TButton;
    InputButtonAdd: TButton;
    InputButtonPlusMinus: TButton;
    InputButtonDecimal: TButton;
    InputButton2: TButton;
    InputButton3: TButton;
    InputButton4: TButton;
    InputButton5: TButton;
    InputButton6: TButton;
    InputButton7: TButton;
    InputButton8: TButton;
    InputButton9: TButton;
    ExpressionInput: TEdit;
    HistoryMemo: TMemo;
    MenuAddVar: TMenuItem;
    MenuDelVar: TMenuItem;
    MenuChangeVar: TMenuItem;
    StoredVariablesPopupMenu: TPopupMenu;
    StoredVariables: TStringGrid;
    procedure ExpressionInputKeyPress(Sender: TObject; var Key: char);
    procedure InputButtonClick(Sender: TObject);
    procedure InputButtonClearClick(Sender: Tobject);
    procedure InputButtonEqualsClick(Sender: Tobject);
    procedure PlusMinusClick(Sender: TObject);
    procedure MenuAddVarClick(Sender: TObject);
    procedure MenuChangeVarClick(Sender: TObject);
    procedure MenuDelVarClick(Sender: TObject);
    procedure StoredVariablesContextPopup(Sender: TObject);
  private
    Parser: TParser;
  public
    procedure UpdateVarList;
  end;

implementation

{ TCalcFrame }
{------------------------------------------------------------------------------}
procedure TCalcFrame.UpdateVarList;
var
  i, j: integer;
  List: TStringList;
begin
  try
    List:=Parser.Variable.GetAllVariables^;
    StoredVariables.Clean;
    StoredVariables.RowCount:=(List.Count div 2)+1;
    StoredVariables.Cells[0,0]:='Name';
    StoredVariables.Cells[1,0]:='Value';
    i:=0;
    j:=-1;
      while j<List.Count-1 do
        begin
          Inc(i);
          Inc(j);
          StoredVariables.Cells[0,i]:=List.Strings[j];
          Inc(j);
          StoredVariables.Cells[1,i]:=List.Strings[j];
        end;
  finally
```

```
    List.Free;
  end;
end;
{-------------------------------------------------------------------------------}
procedure TCalcFrame.InputButtonEqualsClick(Sender: TObject);
begin
  if Parser=nil then Parser:=TParser.Create;
  HistoryMemo.Append(ExpressionInput.Text);
  ExpressionInput.Text:=FloatToStr(Parser.EvalInfix(ExpressionInput.Text));
  HistoryMemo.Append('Result '+ExpressionInput.Text);
end;
{-------------------------------------------------------------------------------}
procedure TCalcFrame.StoredVariablesContextPopup(Sender: TObject);
begin
  if StoredVariables.Selection.Top<=0 then
    begin
      MenuChangeVar.Enabled:=false;
      MenuDelVar.Enabled:=false;
    end
  else
    begin
      MenuChangeVar.Enabled:=true;
      MenuDelVar.Enabled:=true;
    end;
  StoredVariablesPopupMenu.PopUp();
end;
{-------------------------------------------------------------------------------}
procedure TCalcFrame.InputButtonClick(Sender: TObject);
begin
  ExpressionInput.Text:=ExpressionInput.Text+TButton(Sender).Caption;
end;
{-------------------------------------------------------------------------------}
procedure TCalcFrame.PlusMinusClick(Sender: TObject);
var temp: string;
begin
  temp:=ExpressionInput.Text;
  if temp='' then Exit
  else if temp[1]='-' then temp:=MidStr(temp,2,length(temp))
  else
    begin
      temp:='-'+temp;
    end;
  ExpressionInput.Text:=temp;
end;
{-------------------------------------------------------------------------------}
procedure TCalcFrame.InputButtonClearClick(Sender: TObject);
begin
  ExpressionInput.Clear;
end;
{-------------------------------------------------------------------------------}
procedure TCalcFrame.MenuAddVarClick(Sender: TObject);
var
  VarName: string;
  VarResult: extended;
begin
  if Parser=nil then Parser:=TParser.Create;
  VarName:=InputBox('Input Name','Please input name.','');
  try
    if VarName='' then Exit
    else VarResult:=StrToFloat(InputBox('Input Value','Please input
Value.',''));
  except
    Exit;
  end;
  Parser.Variable.AddVariable(VarName,VarResult);
  UpdateVarList;
```

```
end;
{-----------------------------------------------------------------------------}
procedure TCalcFrame.MenuChangeVarClick(Sender: TObject);
var
  VarName: string;
  VarResult: extended;
begin
  VarName:=StoredVariables.Cells[0,StoredVariables.Selection.Top];
  try
    VarResult:=StrToFloat(InputBox('New
Value',VarName,StoredVariables.Cells[1,StoredVariables.Selection.Top]));
  except
    Exit;
  end;
  Parser.Variable.ChangeVariable(VarName,VarResult);
  UpdateVarList;
end;
{-----------------------------------------------------------------------------}
procedure TCalcFrame.MenuDelVarClick(Sender: TObject);
var VarName: string;
begin
  VarName:=StoredVariables.Cells[0,StoredVariables.Selection.Top];
  Parser.Variable.DelVariable(VarName);
  UpdateVarList;
end;
{-----------------------------------------------------------------------------}
procedure TCalcFrame.ExpressionInputKeyPress(Sender: TObject; var Key: char);
begin
  if Ord(Key)=13 then InputButtonEqualsClick(Sender);
end;
{-----------------------------------------------------------------------------}

initialization
  {$I calc.lrs}

end.
```

Number Generator Form Code

```
unit NumberGen;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, LResources, Forms, Controls, Graphics, StdCtrls, Grids,
  ExtCtrls, ComCtrls, Spin;

type

  { TNumberGenForm }

  TNumberGenForm = class(TForm)
    GenSelect: TComboBox;
    LabelStart: TLabel;
    LabelEnd: TLabel;
    MethodGroup: TGroupBox;
    Method1Radio: TRadioButton;
    Method2Radio: TRadioButton;
    MethodEnd: TSpinEdit;
    MethodStart: TSpinEdit;
    OutputList: TListBox;
    Start: TButton;
    procedure Method1RadioClick(Sender: TObject);
    procedure Method2RadioClick(Sender: TObject);
    procedure StartClick(Sender: TObject);
  public
    procedure GenPrime;
    procedure GenSquare;
    procedure GenTriangle;
    procedure GenComposite;

    function IsPrime(const Test: integer): boolean;
    procedure GetMethod(out Param1, Param2: integer; out Method: boolean);
  end;

var
  NumberGenForm: TNumberGenForm;

implementation

{ TNumberGenForm }
{-----------------------------------------------------------------------------}
procedure TNumberGenForm.StartClick(Sender: TObject);
begin
  OutputList.Clear;
  case GenSelect.ItemIndex of
    0: GenPrime;
    1: GenSquare;
    2: GenTriangle;
    3: GenComposite;
  end;
end;
{-----------------------------------------------------------------------------}
procedure TNumberGenForm.Method1RadioClick(Sender: TObject);
begin
  LabelStart.Caption:='Starting number';
  labelEnd.Caption:='Numbers to find';
end;
{-----------------------------------------------------------------------------}
procedure TNumberGenForm.Method2RadioClick(Sender: TObject);
begin
  LabelStart.Caption:='Start of range';
  labelEnd.Caption:='End of range';
end;
{-----------------------------------------------------------------------------}
procedure TNumberGenForm.GetMethod(out Param1, Param2: integer; out Method:
boolean);
begin
  Method:=Method1Radio.Checked;
  Param1:=StrToInt(MethodStart.Text);
  Param2:=StrToInt(MethodEnd.Text);
end;
{-----------------------------------------------------------------------------}
function TNumberGenForm.IsPrime(const Test: integer): boolean;
var i, max: integer;
begin
  if (Test<=1) or ((Test mod 2)=0) and (Test<>2) then
    begin
      Result:=false;
      Exit;
    end;
  max:=Trunc(Sqrt(Test));
  i:=1;
  while i<max do
    begin
      Inc(i,2);
      if (Test mod i)=0 then
        begin
          Result:=false;
          Exit;
        end;
    end;
  Result:=true;
end;
{-----------------------------------------------------------------------------}
procedure TNumberGenForm.GenPrime;
var
  StartValue, EndValue, count: integer;
  method: boolean;
begin
  GetMethod(StartValue,EndValue,Method);
  if Method then
    begin
      count:=0;
      while count<EndValue do
        begin
          if IsPrime(StartValue) then
            begin
              Inc(count);
              OutputList.Items.Add(IntToStr(StartValue));
            end;
          Inc(StartValue);
        end;
    end
  else
    while StartValue<=EndValue do
      begin
        if IsPrime(StartValue) then OutputList.Items.Add(IntToStr(StartValue));
        Inc(StartValue);
      end;
end;
{-----------------------------------------------------------------------------}
procedure TNumberGenForm.GenSquare;
var
  StartValue, EndValue: integer;
  Method: boolean;
begin
  GetMethod(StartValue,EndValue,Method);
  if StartValue=0 then StartValue:=1;
  for StartValue:=StartValue to EndValue do
    case Method of
      true: OutputList.Items.Add(IntToStr(StartValue*StartValue));
      false: if Frac(Sqrt(StartValue))=0 then
OutputList.Items.Add(IntToStr(StartValue));
    end;
end;
{-----------------------------------------------------------------------------}
procedure TNumberGenForm.GenTriangle;
var
  StartValue, EndValue, count, next, total: integer;
  Method: boolean;
begin
  GetMethod(StartValue,EndValue,Method);
  count:=0; total:=0; next:=0;
  if Method then
    while count<EndValue do
      begin
        Inc(next);
        total:=total+next;
        if total>=StartValue then
          begin
            OutputList.Items.Add(IntToStr(total));
            Inc(count);
          end;
      end
  else
```

```
    while total<EndValue do
      begin
        Inc(next);
        total:=total+next;
        if (total>=StartValue) and (total<=EndValue) then
OutputList.Items.Add(IntToStr(total));
      end;
end;
{------------------------------------------------------------------------------}
procedure TNumberGenForm.GenComposite;
var
  StartValue, EndValue, count: integer;
  method: boolean;
begin
  GetMethod(StartValue,EndValue,Method);
  if StartValue<=2 then StartValue:=3;
  if Method then
    begin
      count:=0;
      while count<EndValue do
        begin
          if not(IsPrime(StartValue)) then
            begin
              Inc(count);
              OutputList.Items.Add(IntToStr(StartValue));
            end;
          Inc(StartValue);
        end;
    end
  else
    while StartValue<=EndValue do
      begin
        if not(IsPrime(StartValue)) then
OutputList.Items.Add(IntToStr(StartValue));
        Inc(StartValue);
      end;
end;
{------------------------------------------------------------------------------}

initialization
  {$I numbergen.lrs}

end.
```

Unit Converter From Code

```
unit UnitConvert;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, LResources, Forms, StdCtrls, ConvUtils, StdConvs;

type

  { TUnitConvertForm }

  TUnitConvertForm = class(TForm)
    ConvertValue: TEdit;
    ResultValue: TEdit;
    Filter: TComboBox;
    ResultList: TListBox;
    ConvertList: TListBox;
    procedure ConvertListSelectionChange(Sender: TObject; User: boolean);
    procedure ConvertValueChange(Sender: TObject);
    procedure ConvertKeyPress(Sender: TObject; var Key: char);
    procedure FilterChangeBounds(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure ResultListSelectionChange(Sender: TObject; User: boolean);
    procedure ResultValueChange(Sender: TObject);
  private
    { private declarations }
  public
    procedure ReturnConvTypes(var ConvertTo, ConvertFrom: TConvType);
  end;

var
  UnitConvertForm: TUnitConvertForm;

implementation

{ TUnitConvertForm }
{-----------------------------------------------------------------------------}
procedure TUnitConvertForm.FormCreate(Sender: TObject);
var
  LFamily: TConvFamilyArray;
  FilterList: TStringList;
  i: byte;
begin
  GetConvFamilies(LFamily); // Get ConvertList of unit families.
  FilterList:=TStringList.Create;
  try
    for i:=0 to Length(LFamily)-1 do
      FilterList.Add(ConvFamilyToDescription(i));
    Filter.Items:=FilterList;
    Filter.ItemIndex:=0;
  finally
    FilterList.Free;
  end;
  FilterChangeBounds(Sender);
end;
{-----------------------------------------------------------------------------}
procedure TUnitConvertForm.ResultListSelectionChange(Sender: TObject;
  User: boolean);
begin
  ResultValue.OnChange:=nil;
  ConvertValueChange(Sender);
  ResultValue.OnChange:=@ResultValueChange;
end;
{-----------------------------------------------------------------------------}
procedure TUnitConvertForm.ResultValueChange(Sender: TObject);
var ConvertTo, ConvertFrom: TConvType;
begin
  if ConvertValue.Focused=true then Exit;
  ReturnConvTypes(ConvertTo,ConvertFrom);
  try

ConvertValue.Text:=FloatToStr(Convert(StrToFloat(ResultValue.Text),ConvertTo,Con
vertFrom));
  except
    ConvertValue.Text:='';
  end;
end;
{-----------------------------------------------------------------------------}
procedure TUnitConvertForm.FilterChangeBounds(Sender: TObject);
var
  LFamily: TConvFamily;
  LTypes: TConvTypeArray;
  i: byte;
begin
```

```
  LFamily:=Filter.ItemIndex;
  GetConvTypes(LFamily, LTypes);
  ConvertList.Clear;
  for i:=0 to Length(LTypes)-1 do
    ConvertList.Items.Add(ConvTypeToDescription(LTypes[i]));
  ConvertList.ItemIndex:=0;
  ResultList.ItemIndex:=0;
end;
{-------------------------------------------------------------------------------}
procedure TUnitConvertForm.ConvertListSelectionChange(Sender: TObject;
  User: boolean);
var
  LFamily: TConvFamily;
  LTypes: TConvTypeArray;
  CurrentType: TConvType;
  i: byte;
begin
  CurrentType:=TListBox(Sender).ItemIndex;
  LFamily:=Filter.ItemIndex;
  GetConvTypes(LFamily, LTypes);

  ResultList.Clear;
  for i:=0 to Length(LTypes)-1 do
    if i<>CurrentType then
ResultList.Items.Add(ConvTypeToDescription(LTypes[i]));
  ResultList.ItemIndex:=0;
  ConvertValueChange(Sender);
end;
{-------------------------------------------------------------------------------}
procedure TUnitConvertForm.ConvertValueChange(Sender: TObject);
var ConvertTo, ConvertFrom: TConvType;
begin
  if ResultValue.Focused=true then Exit;
  ReturnConvTypes(ConvertTo,ConvertFrom);
  try

ResultValue.Text:=FloatToStr(Convert(StrToFloat(ConvertValue.Text),ConvertFrom,C
onvertTo));
  except
    ResultValue.Text:='';
  end;
end;
{-------------------------------------------------------------------------------}
procedure TUnitConvertForm.ConvertKeyPress(Sender: TObject; var Key: char);
var i: integer;
begin
  if not(Key in['0'..'9','.',Chr(8)]) then Key:=Chr(0)
  else if Key='.' then
    for i:=1 to Length(TEdit(Sender).Text) do
      if TEdit(Sender).Text[i]='.' then
        begin
          Key:=Chr(0);
          Break;
        end;
end;
{-------------------------------------------------------------------------------}
procedure TUnitConvertForm.ReturnConvTypes(var ConvertTo, ConvertFrom:
TConvType);
var
  LFamily: TConvFamily;
  LTypes:  TConvTypeArray;
begin
  LFamily:=Filter.ItemIndex;
  ConvertFrom:=ConvertList.ItemIndex;
  ConvertTo:=ResultList.ItemIndex;
  GetConvTypes(LFamily, LTypes);
  ConvertFrom:=ConvertFrom+LTypes[0];
  ConvertTo:=ConvertTo+LTypes[0];
  if ConvertFrom<=ConvertTo then ConvertTo:=ConvertTo+1;
end;
{-------------------------------------------------------------------------------}

initialization
  {$I unitconvert.lrs}

end.
```

Graph Form Code

```
unit Graph;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, LResources, Forms, Controls, Graphics, Dialogs, StdCtrls,
  ExtCtrls, Parser;

type

  { TGraphForm }

  TGraphForm = class(TForm)
    FunctionInput: TEdit;
    PaintArea: TPaintBox;
    procedure FunctionInputKeyPress(Sender: TObject; var Key: char);
    procedure FormCreate(Sender: TObject);
    procedure PaintAreaPaint(Sender: TObject);
  private
    Parser: TParser;
    XRange: integer;
    YRange: integer;
    XRangePixels: integer;
    YRangePixels: integer;
    CorrectX, CorrectY: integer;
    Origin: TPoint;
  public
    x: extended;
    procedure DrawAxis;
    procedure DrawGraph;
  end;

var
  GraphForm: TGraphForm;

implementation

{ TGraphForm }
{-------------------------------------------------------------------------------}
procedure TGraphForm.FunctionInputKeyPress(Sender: TObject; var Key: char);
begin
  if Ord(Key)=13 then
  if FunctionInput.Text<>'' then PaintArea.Refresh;
end;
{-------------------------------------------------------------------------------}
procedure TGraphForm.FormCreate(Sender: TObject);
begin
  Parser:=TParser.Create;
  Parser.Variable.AddVariable('x',0);
  Origin.x:=PaintArea.Width div 2;
  Origin.y:=PaintArea.Height div 2;
```

```
end;
{------------------------------------------------------------------------------}
procedure TGraphForm.PaintAreaPaint(Sender: TObject);
begin
  if FunctionInput.Text='' then DrawAxis
  else DrawGraph;
end;
{------------------------------------------------------------------------------}
procedure TGraphForm.DrawAxis;
begin
  PaintArea.Canvas.MoveTo(Origin.x+9,0);
  PaintArea.Canvas.LineTo(Origin.x+9,PaintArea.Height);
  PaintArea.Canvas.MoveTo(PaintArea.Width,Origin.y+16);
  PaintArea.Canvas.LineTo(0,Origin.y+16);
end;
{------------------------------------------------------------------------------}
procedure TGraphForm.DrawGraph;
var
  Points: array[0..41] of TPoint;
  i, j: integer;
  res: extended;
begin
  DrawAxis;
  Origin:=Point(PaintArea.Width div 2,PaintArea.Height div 2);
  XRange:=20;
  YRange:=20;
  XRangePixels:=PaintArea.Width div (XRange*2);
  YRangePixels:=PaintArea.Height div (YRange*2);
  correctx:=XRangePixels mod XRange;
  correcty:=YRangePixels mod YRange;
  j:=0;
  try
    for i:=(XRange*-1) to XRange do
      begin
        Parser.Variable.ChangeVariable('x',i);
        res:=Parser.EvalInfix(FunctionInput.Text);
        Inc(j);
        Points[j].X:=((i+XRange)*XRangePixels)+CorrectX;
        Points[j].Y:=PaintArea.Height-((Trunc(res)+YRange)*YRangePixels)
+CorrectY;
      end;
    PaintArea.Canvas.Polyline(Points);
  except
  end;
end;
{------------------------------------------------------------------------------}

initialization
  {$I graph.lrs}

end.
```

Revision Frame Code

```
unit MathTopic;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, LResources, Forms, ComCtrls, ExtCtrls, Buttons, StdCtrls,
  Arrow, Controls;

type
  TPageData = record
```

```
    Stage: shortint;
    Difficulty: byte;
    X: integer;
    Y: integer;
    XCo1: integer; // Co for Coefficient.
    YCo1: integer;
    XCo2: integer;
    YCo2: integer;
    Res1: integer;
    Res2: integer;
  end;

type

  { TMathTopicFrame }

  TMathTopicFrame = class(TFrame)
    Blank: TPage;
    BlankLabel: TLabel;
    SelDif: TComboBox;
    SimEquLabelExp1: TLabel;
    SimEquLine: TShape;
    SimEquLabel3: TLabel;
    SimEquLabel4: TLabel;
    SimEquAdd: TLabel;
    SimEquNext: TButton;
    SimEquCombo: TComboBox;
    ListBox1: TListBox;
    SimEquLabel1: TLabel;
    SimEquLabel2: TLabel;
    Item1: TPage;
    TriandImp: TPage;
    ShowHideTree: TArrow;
    Button3: TButton;
    Notebook: TNotebook;
    SimEqu: TPage;
    TopicTree: TTreeView;
    procedure ShowHideTreeClick(Sender: TObject);
    procedure SimEquNextClick(Sender: TObject);
    procedure TopicTreeSelectionChanged(Sender: TObject);
  private
    SimEquData: TPageData;
  public
    procedure GenVariables(const PageName: string);
    procedure GenSimEqu(const GenStage: byte);
  end;

implementation

{------------------------------------------------------------------------------}
function CheckMinus(const Data: integer): string;
begin
  if Data<0 then Result:='x'
  else Result:='x+';
end;
{------------------------------------------------------------------------------}

{ TMathTopicFrame }
{------------------------------------------------------------------------------}
procedure TMathTopicFrame.TopicTreeSelectionChanged(Sender: TObject);
var
  temp, NoSpaces: string;
  i: byte;
begin
  temp:=TopicTree.Selected.Text;
  NoSpaces:='';
  NoSpaces:=temp[1]+temp[2]+temp[3];
  for i:=4 to Length(temp) do
    if temp[i]=' ' then NoSpaces:=NoSpaces+temp[i+1]+temp[i+2]+temp[i+3];
  Notebook.ActivePage:=NoSpaces;
  GenVariables(NoSpaces);
end;
{-------------------------------------------------------------------------------}
procedure TMathTopicFrame.ShowHideTreeClick(Sender: TObject);
begin
  case TopicTree.Visible of
    true: begin
      ShowHideTree.Left:=8;
      Notebook.Left:=32;
      ShowHideTree.ArrowType:=atRight;
    end;
    false: begin
      ShowHideTree.Left:=TopicTree.Width+8;
      Notebook.Left:=TopicTree.Width+ShowHideTree.Width+8;
      ShowHideTree.ArrowType:=atLeft;
    end;
  end;
  TopicTree.Visible:=not TopicTree.Visible;
  ShowHideTree.Refresh;
end;
{-------------------------------------------------------------------------------}
procedure TMathTopicFrame.SimEquNextClick(Sender: TObject);
begin
  GenSimEqu(SimEquData.Stage);
end;
{-------------------------------------------------------------------------------}
procedure TMathTopicFrame.GenVariables(const PageName: string);
begin
  case Notebook.IndexOf(TPage(FindComponent(PageName))) of
    0: GenSimEqu(1);
  end;
end;
{-------------------------------------------------------------------------------}
procedure TMathTopicFrame.GenSimEqu(const GenStage: byte);
const br = sLineBreak;
var
  temp1, temp2: integer;
  space: string = '';
  tempStr: string = '';
begin
  case GenStage of
    1:begin
        with SimEquData do
          begin
            SelDif.Visible:=true;
            SimEquLine.Visible:=false;
            SimEquCombo.Visible:=true;
            SimEquNext.Visible:=true;
            SimEquLabelExp1.Caption:='Here there are two equations.';
            SimEquLabelExp1.Visible:=true;
            X:=Random(21)-10;
            Y:=Random(21)-10;
            XCo1:=Random(21)-10;
            YCo1:=Random(21)-10;
            XCo2:=Random(21)-10;
            YCo2:=Random(21)-10;
            Res1:=(XCo1*X)+(YCo1*Y);
            Res2:=(XCo2*X)+(YCo2*Y);
            space:=CheckMinus(YCo1);

SimEquLabel1.Caption:=IntToStr(XCo1)+space+IntToStr(YCo1)+'y='+IntToStr(Res1);
            space:=CheckMinus(YCo2);
```

```
SimEquLabel2.Caption:=IntToStr(XCo2)+space+IntToStr(YCo2)+'y='+IntToStr(Res2);
          end;
      end;
    2:begin
        SelDif.Visible:=false;
        case SimEquCombo.ItemIndex of
          0:begin
              tempStr:='(1) '+SimEquLabel1.Caption+'
*'+IntToStr(SimEquData.XCo2);
              space:=  '(2) '+SimEquLabel2.Caption+'
*'+IntToStr(SimEquData.XCo1);
            end;
          1:begin
              tempStr:='(1) '+SimEquLabel1.Caption+'
*'+IntToStr(SimEquData.YCo2);
              space:=  '(2) '+SimEquLabel2.Caption+'
*'+IntToStr(SimEquData.YCo1);
            end;
        end;
        SimEquCombo.Visible:=false;
        SimEquLabel1.Caption:=tempStr;
        SimEquLabel2.Caption:=space;
      end;
    3:begin
        with SimEquData do
          begin
            if SimEquCombo.ItemIndex=0 then
              begin
                temp1:=XCo1;
                temp2:=XCo2;
              end
            else
              begin
                temp1:=YCo1;
                temp2:=YCo2;
              end;
            XCo1:=XCo1*temp2;
            YCo1:=YCo1*temp2;
            XCo2:=XCo2*temp1;
            YCo2:=YCo2*temp1;
            Res1:=(X*XCo1)+(Y*YCo1);
            Res2:=(X*XCo2)+(Y*YCo2);
            space:=CheckMinus(YCo1);

SimEquLabel1.Caption:=IntToStr(XCo1)+space+IntToStr(YCo1)+'y='+IntToStr(Res1);
            space:=CheckMinus(YCo2);

SimEquLabel2.Caption:=IntToStr(XCo2)+space+IntToStr(YCo2)+'y='+IntToStr(Res2);
            case SimEquCombo.ItemIndex of
              0:begin
                  if (XCo1>0) and (XCo2<0) or (XCo1<0) and (XCo2>0) then
SimEquAdd.Caption:='+'
                  else SimEquAdd.Caption:='-';
                end;
              1:begin
                  if (YCo1>0) and (YCo2<0) or (YCo1<0) and (YCo2>0) then
SimEquAdd.Caption:='+'
                  else SimEquAdd.Caption:='-';
                end;
            end;
            SimEquLine.Visible:=true;
            SimEquAdd.Visible:=true;
          end;
      end;
    4:begin
```

```
        with SimEquData do
          begin
            case SimEquCombo.ItemIndex of
              0:begin
                  space:=IntToStr(YCo1-YCo2)+'y='+IntToStr(Res1-Res2)+br;
                  space:=space+'    y='+IntToStr((Res1-Res2) div (YCo1-YCo2));
                end;
              1:begin
                  space:=IntToStr(XCo1-XCo2)+'x='+IntToStr(Res1-Res2)+br;
                  space:=space+'    x='+IntToStr((Res1-Res2) div (XCo1-XCo2));
                end;
            end;
            SimEquLabel3.Caption:=space;
          end;
      end;
    5:begin
        with SimEquData do
          begin
            case SimEquCombo.ItemIndex of
            0:begin
                space:=CheckMinus(YCo1);
                space:=IntToStr(XCo1)+space+IntToStr(yCo1)+'('+IntToStr(y)
+')='+IntToStr(Res1)
                +br+inttostr(XCo1)+'x='+inttostr(Res1-(yCo1*y))
                +br+'x='+inttostr((Res1-(yCo1*y)) div XCo1);
              end;
            1:begin
                if YCo1<0 then space:='-' else space:='+';
                space:=IntToStr(XCo1)+'('+IntToStr(x)
+')'+space+IntToStr(YCo1)+'y='+IntToStr(Res1)
                +br+IntToStr(YCo1)+'y='+IntToStr(Res1-(XCo1*x))
                +br+'y='+IntToStr((Res1-(XCo1*x)) div YCo1);
              end;
            end;
            SimEquLabel4.Caption:=space;
            SimEquNext.Visible:=false;
            Stage:=-1;
          end;
      end;
  end;
  SimEquData.Stage:=GenStage+1;
end;
{--------------------------------------------------------------------------------}

initialization
  {$I mathtopic.lrs}
  Randomize;

end.
```

About Form Code

```
unit About;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, LResources, Forms, Controls, Graphics, ExtCtrls, StdCtrls,
  ParticleClasses;

type

  { TAboutForm }
  TAboutForm = class(TForm)
    Bevel1: TBevel;
    Bevel2: TBevel;
    btnClose: TButton;
    CircleMove: TTimer;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    procedure btnCloseClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure CircleMoveTimer(Sender: TObject);
    procedure FormHide(Sender: TObject);
    procedure FormShow(Sender: TObject);
  private
    ParticleControl: TParticleControl;
  end;

var
  AboutForm: TAboutForm;

implementation

{ TAboutForm }
{-----------------------------------------------------------------------------}
procedure TAboutForm.btnCloseClick(Sender: TObject);
begin
  Close;
end;
{-----------------------------------------------------------------------------}
procedure TAboutForm.FormCreate(Sender: TObject);
var i: byte;
begin
  ParticleControl:=TParticleControl.Create(Bevel2.Parent);
  ParticleControl.SetBounds(Bevel2.Left+1,Bevel2.Top+1,Bevel2.Width-
2,Bevel2.Height-2);
  for i:=1 to 3 do
    begin
      ParticleControl.AddParticle;
      TParticle(FindComponent('Circle'+IntToStr(i))).SendToBack;
    end;
end;
{-----------------------------------------------------------------------------}
procedure TAboutForm.CircleMoveTimer(Sender: TObject);
begin
  ParticleControl.MoveParticles;
end;
{-----------------------------------------------------------------------------}
procedure TAboutForm.FormHide(Sender: TObject);
begin
  CircleMove.Enabled:=false;
end;
{-----------------------------------------------------------------------------}
procedure TAboutForm.FormShow(Sender: TObject);
begin
  CircleMove.Enabled:=true;
end;
{-----------------------------------------------------------------------------}

initialization
  {$I about.lrs}
  Randomize;

end.
```