**Analysis**

**The Problem**

I am aiming to produce an offline graphical calculator for use by students and teachers in maths lessons. The software should plot and solve functions, and display important information such as maximum and/or minimum points, roots, and points of intersection.

As an A-Level Mathematics and Further Mathematics student, I was drawn to this concept, as it would not only be useful when solving problems and checking solutions, but as a project is interesting, as I will be able to create, and improve a system that my classmates and myself use often.

Currently, graphical calculators are not accessed by all students since they are expensive. Other systems do exist, but they are often containing an impractical user interface and lack certain features that would add to the usefulness of the system. As well as this, they require an internet connection, which may not always be available.

Despite this, it is obvious that graphical calculators are in demand from students, with many choosing to purchase one. They are valued for the extra convenience that they provide through additional features that regular scientific calculators do not provide. Therefore, creating an offline system, that is able to provide a user-friendly interface that has an adequate number of additional features would be highly beneficial.

**Current Systems**

The most popular systems currently used are the Desmos graphing calculator and the Casio fx-CG50 calculator.

Desmos

Desmos is a graphical calculator accessed online through a website.

Good features:

* Ability to display many graphs at once, and select which ones to be displayed. They are shown in different colours, to easily distinguish between them.
* Calculates values of interest – points of intersection, maximum, minimum points etc.
* Can easily zoom in and out, and drag to see different parts of the graph.
* Graphs drawn on a square grid for easy visuals and for a scale.
* Can plot point, functions and other graphs (including circles).

Areas of improvement:

* To see important values, the user must hover over them, which can be awkward, and impractical as not all values can be seen at once.

Chart, line chart

Description automatically generated

The user hovers the mouse to see points, e.g. the intersection between two graphs.

Desmos also allows the use of constants and constant changers in order to alter graphs drawn:

Chart

Description automatically generated with low confidence

Although this feature may be useful, it is not often that students make use of this, and it does not provide enough additional functionality to justify its implementation into this project. Therefore, at this stage, it remains a possible extension objective.

Casio fx-CG50

Good features:

* Ability to display many graphs at once, and select which ones to be displayed. They are shown in different colours, to easily distinguish between them.
* Can get important values about the graphs plotted e.g., points of intersection, maximum and minimum points etc.
* Can zoom in and out, a grid being displayed for scale as the background when zoomed in enough.

Areas of improvement:

* When zooming in and out, the center point must be chosen, which can be more of a slow and awkward process compared to the dragging of the image Desmos offers.
* There is not a grid displayed for scale when zoomed out to a certain point.
* Many students may struggle to identify the features of the calculator and use them with ease due to the particular use of menus.
* Not all important values can be seen at once.

Diagram

Description automatically generated

User navigates menus to see values of interest, zoom in/out and make use of other features.

**Prospective Users and Requirements**

After conducting a survey on A-Level Mathematics and Further Mathematics students’ experiences of graphical calculators, I found the following data about user experience:

1. Of the respondents that use the Casio fx-CG50, 68% said it was easy to navigate, but 84% said that it was difficult to initially get used to. This could be due to the heavy use of menus. The respondents that used Desmos all said it was both easy to navigate and easy to get used to. Therefore, implementing a more application-style user interface should provide the user with more ease of use.
2. Only 41% of respondents said they made use of constant changers, with some unaware of what they were. This means that the implementation of these may be useful for some, but is not a widespread need for users as a whole.
3. 100% of respondents said that they would find a system that does not require an internet connection useful. Therefore, I will be creating a software that can be used offline that should be more easily accessible to users.
4. Respondents identified the usefulness and necessity of various features:

Chart, bar chart

Description automatically generated with medium confidence

Overall, all features listed have relatively high demand, but points of intersection and other useful information being displayed were identified as most important. Therefore, it will be important to ensure that these features are implemented in a very user-friendly way.

It may prove difficult to computationally calculate points of intersection and turning points, as this would require working with the derivative, but this can remain an extension objective for now.

1. Other general comments included:
   1. “It would be helpful if each page had a small help window that would explain how to use the features. As currently I have to google how to do something.” – A help window can be included in the main menu that gives the user the option of a small tutorial, although this should hopefully not be required to a great extent, as the user interface should be simple to use.
   2. “It would also be useful if it could restrict graphs through domains and range.” – this is an additional feature that can be implemented as part of the functionality of the system.

**Shunting Yard Parser vs. Recursive Descent Parser**

An important and challenging aspect of this project is validating and interpreting the mathematical expressions which are input by the user. There are two main approaches to do this: Shunting Yard and Recursive Descent.

The Shunting Yard is a bottom-up technique that parses expressions input in infix form into postfix or Reverse Polish Notation (RPN). It uses two stacks, reading the input from left to right. Recursive descent is a top-down technique that recursively breaks down the input into its smaller parts. This typically produces an Abstract Syntax Tree (AST).

Both methods have benefits and drawbacks, and are suited for different situations:

|  |  |
| --- | --- |
| Shunting Yard | |
| Benefits | Drawbacks |
| Can handle expressions with multiple levels of precedence and parentheses, efficiently. | Evaluating the postfix expression can be quite costly in terms of processing time and complexity. |
| Can be extended to handle functions and other non-operator tokens in the input. | Requires the use of two data structures – a stack and a queue. |
| Can be implemented using a stack and queue, which are quite simple as data structures. | Adds unnecessary steps to the conversion of simple expressions with low levels of precedence. |
| On more complex operations, parsing is easier and more efficient. | Can be more difficult to understand. |
| Recursive Descent |  |
| Benefits | Drawbacks |
| Relatively easy to implement and understand, specifically for simple expressions with low levels of precedence. | Can have issues with operator precedence and associativity – each function is called in a fixed order. |
| Can be used to generate code or ASTs, by each parsing function representing each sub-expression. | Can be more difficult to implement for complex expressions- each parsing function must be designed to handle the levels of precedence. |
| Efficient for small or medium sized expressions. | Parsing functions are tightly coupled to grammar, so may not be flexible. |
| Handles left-recursive grammars. | Inefficient on larger and more complex expressions, as many function calls are required. |

Although both methods are likely to handle the requirements of my system well, I have decided to implement recursive descent as my system is unlikely to have expressions of complexity high enough to invalidate its use. The grammar can also be defined to easily suit the requirement for non-left-recursion.

**Abstract Syntax Trees (ASTs)**

An Abstract Syntax Tree is a data structure that represents the hierarchical structure of a mathematical expression to be evaluated and is generally the output of the Recursive Descent Parser.

ASTs are advantageous since they give a clearer representation and can be modified quite easily. This is because the relationships between nodes determine the order of operations.

ASTs are made of terminal and non-terminal nodes. To evaluate an expression, the nodes are visited from the bottom towards the top, with the operation in the non-terminal being performed on its child nodes. In this way, the correct order of operation is maintained as the nodes towards the top contain the operations which have the lowest precedence.

An example of an expression to be handled by the system is the cubic y = 3x^3 + 5x^2 + 4x - 6. When running recursive descent on this, a tree is created where the expression is first split up by the + and - operators, which have the lowest precedence. Then the \* and ^ operators.

An AST representation of this would be:

**Components of the Recursive Descent Parser**

The building of the Recursive Descent Parser can be broken down into the following components:

1. A grammar – this defines the syntax of the language used, for example breaking down the expressions into variables, powers, factors, terms etc.
2. A lexer – this is used to break the input down into tokens, which are components of the expression that have been defined in the grammar, and to remove whitespace.
3. Recursive procedures – a set of procedures that are each responsible for recognising one syntactic element. These call each other recursively until the input has been completely converted into a tree.
4. Error handling – this is necessary to ensure that if invalid data is present in the input string, this can be dealt with efficiently.

**Data Volumes**

The need for persistent storage of data is limited, with the only possible requirement being the settings the user has set. Therefore, the settings of the system should be saved, and updated only when prompted by the user. This would improve the user’s experience as they would not have to readjust the view window each time they open the application, for example.

**Acceptable Limitations**

There are many features that already existing systems implement that would provide useful functionality, but it would not be realistic for me to implement these in the time frame I have.

Therefore, the system should only be expected to:

1. Plot functions on the real plane (no use of complex numbers).
2. Plot Cartesian functions in the form y = .
3. Plot 2D functions.
4. Plot up to 5 functions at a time.
5. Accept the view window preferences to be within -100 and 100, and integers.
6. Plot exponential, logarithmic, linear, trigonometric functions, and polynomials, or a combination of these.
7. Use angles in the form of radians and/or degrees.
8. Calculate and display roots and y-intercepts.

I have chosen to only use radians and degrees as angle types because these are the only two that are used by students in the A-Level Mathematics and Further Mathematics course, with gradians having very little use in general.

If extension objectives are met, then the system should also be able to calculate and display turning points and intersection points, as well as cope with domains, and calculate and display the range.

**Numbered Objectives of the Project**

1. The Main Menu form is displayed.
   1. There will be the buttons “Help”, “Input Functions”, “Settings”, “Exit” displayed.
   2. When clicked, the respective form should open.
   3. The Main Menu form should close once another form opens.
2. If the “Help” button is pressed, a help form opens providing a brief explanation of how to use the application.
3. If the “Settings” button is pressed, a Settings form should load that allows the user to enter preferences.
   1. Relevant text boxes/combo boxes should be present allowing the user to input information about their preferences.
      1. Text boxes for “Max Y”, “Max X”, “Min X”, “Min Y”.
      2. Combo box for “Angle Type”, users can select between radians and degrees.
   2. A Save button is available to be pressed to save the information.
      1. If data is invalid, an error message is displayed.
         1. Invalid if view window settings empty.
         2. Invalid if view window settings equal.
         3. Invalid if view window settings outside of range (-100 and 100).
         4. Invalid if max values less than min.
      2. If all data is valid, the Settings table in the database is to be updated.
4. If the “Input Functions” button is pressed, a “Input” form loads, allowing the user to input functions to be plotted.
   1. 5 text boxes should be available to input up to 5 functions at once.
   2. A “Plot” button should be available that the user presses when the functions have been input.
      1. Once pressed, validation of the data input occurs.
         1. Invalid if letters other than “x” or “e” used.
         2. Invalid if functions other than “sin”, “cos”, “tan”, “ln” used.
         3. Invalid if characters other than “+”, “-“, “/”, “\*”, “^” used.
         4. Invalid if double operators used, except in the case of “- -”.
         5. Invalid if unpaired brackets used.
         6. Invalid if all boxes empty.
      2. If any invalid data entered, an appropriate message is displayed.
      3. If all data is valid, the function should be accepted.
5. The “Graph” form should load, where the functions will be drawn.
   1. A square grid with a correctly scaled set of axes appears, based on the Settings selected by the user.
   2. Tick boxes should be present next to all functions to allow the user to choose which are to be plotted.
      1. The Recursive Descent algorithm should be run on selected functions to create an AST, which is then evaluated in appropriate increments of x values.
      2. Roots correctly obtained and stored.
      3. Intercepts correctly obtained and stored.
      4. Functions selected plotted based on coordinates obtained from the AST evaluation.
   3. The form should allow zooming in and out on the view window.
   4. Different graphs should be plotted in different colours.
   5. Text boxes containing roots and y-intercepts for each function should be displayed.
   6. Linear functions should be correctly plotted.
   7. Polynomials should be correctly plotted.
   8. Exponential and logarithmic functions should be correctly plotted.
   9. Trigonometric functions should be correctly plotted.
6. “Back” buttons should be available on all forms apart from “Main Menu”, which when pressed, return to “Main Menu” and close the current form, or in the case of the “Graph” form, return to the “Input” form.

**Extension Objectives**

1. Allow for inputs of domains in the “Input” form and display the ranges in the “Graph” form.
2. Work out the derivative of the functions and use to calculate and display turning points.
   1. Turning points should be stored.
3. Find points of intersection of functions plotted together.
   1. Store and display these in relevant textboxes on the “Graph” form.

**Proposed Method of Solution**

I will be using Visual Studio for this project, programming in C#, as this is the language I am most comfortable with. Additionally, I will be using a Windows Forms Application, which I believe to be appropriate as users should find this easy to navigate.

**Design**

**System Diagram**

The application has 5 forms: Main Menu, Settings, Help, Input and Graph:

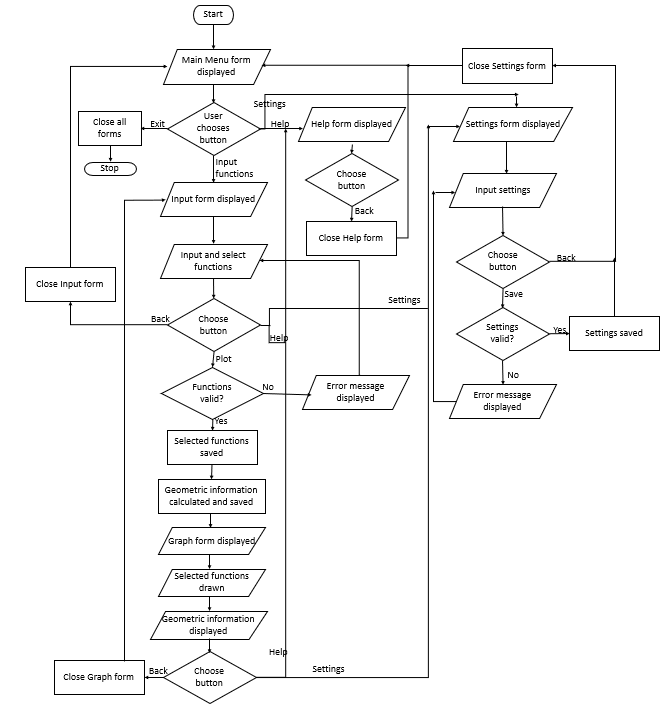
* Upon loading the application, the Main Menu form is loaded.
* The Input form can be accessed uniquely from the Main Menu form.
* The Graph form can be accessed uniquely from the Input form.
* The Help form can be accessed from any form.
* The Settings form can be accessed from any form, besides from the Help form.
* Back buttons on all forms allow previous forms to be accessed.

Diagram

Description automatically generated

**System Flowcharts**

The main flowchart of the user’s navigation through the application is shown below:



**Class Diagrams – to be completed during implementation.**

**Data Structures**

As discussed in the Analysis section, I will be using trees (Abstract Syntax Trees) to store the parsed expression that is entered by the user. These will be the result of running the Recursive Descent parser. I plan on implementing the trees by writing a node class.

Additionally, I will be using a lexer to break the expression down into tokens. This means that user-defined types are necessary as part of this. The enumerated type containing the various aspects of the grammar is defined below:

enum TokenType

{

Variable,

Operator1,

Operator2,

Exp,

LParen,

RParen,

Number,

Trig,

End,

Error

}

**Recursive Descent Parser**

The token class defines the properties of a token and allows for the token object to be passed into the parser:

class Token

{

public TokenType type;

public string value;

public Token(TokenType Type, string Value)

{

type = Type;

value = Value;

}

}

The Lexer class will break down the expression into tokens, by assigning each token with a type and value, and removing whitespace:

class Lexer

{

public string input;

public int pos;

public char currchar;

public Lexer(string Input)

{

input = Input;

pos = 0;

currchar = Convert.ToChar(input.Substring(0, 1));

}

private void Next()

{

pos++;

if (pos == input.Length)

{

currchar = '!';

}

else

{

currchar = input[pos];

}

}

private void RemoveWhitespace()

{

if (currchar == ' ')

{

Next();

}

}

private string GetNumber()

{

string num = "";

if (char.IsDigit(currchar) || currchar == '.')

{

num = num + currchar;

Next();

return GetNumber();

}

else

{

return num;

}

}

private string GetWord()

{

string word = "";

if (char.IsLetter(currchar))

{

word = word + currchar;

Next();

return GetWord();

}

else

{

return word;

}

}

public Token GetNextToken()

{

while (currchar != '!')

{

if (currchar == ' ')

{

RemoveWhitespace();

}

else if (char.IsDigit(currchar))

{

string value = GetNumber();

return new Token(TokenType.Number, value);

}

else if (char.IsLetter(currchar))

{

string value = GetWord();

if (value == "sin" || value == "cos" || value == "tan")

{

return new Token(TokenType.Trig, value);

}

else if (value == "x" || value == "e")

{

return new Token(TokenType.Variable, value);

}

else

{

return new Token(TokenType.Error, currchar.ToString()); ;

}

}

else if (currchar == '^')

{

return new Token(TokenType.Exp, currchar.ToString());

}

else if (currchar == '\*' || currchar == '/')

{

return new Token(TokenType.Operator1, currchar.ToString());

}

else if (currchar == '+' || currchar == '-')

{

return new Token(TokenType.Operator2, currchar.ToString());

}

else if (currchar == '(')

{

return new Token(TokenType.LParen, currchar.ToString());

}

else if (currchar == ')')

{

return new Token(TokenType.RParen, currchar.ToString());

}

else

{

return new Token(TokenType.Error, currchar.ToString());

}

}

return new Token(TokenType.End, "");

}

}

The parser will use token types in order to break down the expression into terms, and then factors. The Expression() method breaks down the input into terms by operators + and -. The Term() method breaks down the expression into factors by operators \* , / and ^. The Factor() fully breaks down the expression into numbers, variables and trigonometric functions – these are the leaf nodes.

class Parser

{

private Lexer lexer;

private Token currtok;

public Parser(Lexer Lexer)

{

lexer = Lexer;

currtok = lexer.GetNextToken();

}

private void Consume()

{

currtok = lexer.GetNextToken();

}

public Node Parse()

{

return Expression();

}

private Node Expression()

{

Node node = Term();

if (currtok.type == TokenType.Operator2)

{

string value = currtok.value;

Consume();

node = new OpNode(node, Term(), value);

}

return node;

}

private Node Term()

{

Node node = Factor();

if (currtok.type == TokenType.Operator1)

{

string value = currtok.value;

Consume();

node = new OpNode(node, Factor(), value);

}

else if (currtok.type == TokenType.Exp)

{

node = new ExpNode(node, Factor(), "^");

Consume();

}

return node;

}

private Node Factor()

{

Token tok = currtok;

if (tok.type == TokenType.LParen)

{

Consume();

Node node = Expression();

Consume();

return node;

}

else if (tok.type == TokenType.Variable)

{

Consume();

return new VarNode(null, null, tok.value);

}

else if (tok.type == TokenType.Number)

{

Consume();

return new NumNode(null, null, tok.value);

}

else if (tok.type == TokenType.Trig)

{

Consume();

Consume();

Node node = Expression();

Consume();

return new TrigNode(null, null, node, tok.value);

}

else

{

return null; //error

}

}

}

**UI Design**

Main Menu Form

Graphical user interface, application

Description automatically generated

The Input Functions button can be pressed to navigate to the Input form, where functions are to be input by the user.

The user can press the button Help to navigate to the “Help” form.

The Settings button can be pressed to navigate to the Settings form, where the user can edit the settings.

The user can press the Exit button to close the entire application.

Settings Form

The Help button allows the user to navigate to the Help window.

Graphical user interface

Description automatically generated

Settings form closes and returns to the Main Menu form.

The angle type is selected by the user using a combo box. Options are degrees and radians.

The user can type the max and min x and y values, which must be valid.

Once inputs are validated, the settings stored by the system are updated when the Save button is pressed. Then, the Settings form closes and returns to the Main Menu form.

Input Form

The Help button allows the user to navigate to the Help window.

The Settings button allows the user to navigate to the Settings form and change the settings.

Graphical user interface

Description automatically generated

The user can input functions into these textboxes, in the accepted form e.g. x^2 + 3x -5.

The user selects the tick boxes of the functions they want to plot.

Once validated, the functions are converted to an AST representation, and the Graph form loads, ready to plot.

Input form closes and returns to the Main Menu form.

Graph Form

The Help button allows the user to navigate to the Help window.

The Settings button allows the user to navigate to the Settings form and change the settings.

Graphical user interface, application, Word

Description automatically generated

The Back button can be pressed by the user to return to the Input form.

Important information, such as the roots of the functions, will be displayed here.

The functions selected by the user, with the colour coding, will be written here.

The functions selected by the user will be plotted here, in different colours.

Help Form

– to be added during implementation.