CMP-6048A/7009A Advanced Programming

Project Report - Due 12 January 2026 before 15:00

Maths Interpreter software

Group members:
Mason Buckle and Anthony De Cruz.

School of Computing Sciences, University of East Anglia

Version 2.0

Abstract

This reprt documents both the design and implementation of a custom mathematical intpreter and graphing calculator. The primary objective was to o develop a capable maths intepreter and GUI with a user friendly interface. The development methodology adopted a hybrid programming approach, using the functional paradigm of F Sharp for the interpreter logic and the object oriented nature of C Sharp for the user interface.

The project evolved through a series of iterative stages. Early development focuses on creating a foundational GUI, this was expanded to support advanced intpreter features such as operator precedence, floating point arithmetic and variable storage using a symbol table. An external plotting library was used to enable the visulization of linear and polynomial functions.

The final result is a fully functional mathematical software. It demonstrates successful compatibility between two langaes, with the front end handling the user input and the backend using lexical analysis and evaluation. The inclusion of the optional features: control flow and booleans represents the flexibility of the project.

Chapter 1

Introduction

1.1 Project statement

This project focuses on developing a desktop maths software solution with GUI that uses an interpreter. This project play an important role in areas such as education and research, offering a platform to test mathematical concepts easier. Fsharp is used for the interpreter and Csharp (WPF) is ued for the GUI. The software has been developed over a period of 4 months and was split into sprints (see Development History 3). We accomplished this via modular design and testing each part at every stage. Git was used for version control and Github's kanban board feature was used to break down tasks. The final deliverable is a capable desktop maths software solution with a GUI that succesfully links a maths interpreter with a user-friendly interface.

1.2 Aims and objectives

The main overarching goal of the project is to develop a capable maths intepreter and GUI with a user friendly interface, this is broken down further into the main project objectives below:

Project Objectives

1. To implement a interpreter capable of correctly intepreting and executing arithmetic expressions, managing variable assignment and executing control flow loops.
2. To create a responsive GUI that is capable of accepting user commands, displaying results and displaying errors
3. To create a plotting section in the GUI to plot both linear and polynomial functions and have interactive capabilities such as zooming in and out.

Table 1.1: (Functional) MoSCoW

Priority	Task	Comments
Must	To implement a intpreter cabable of correctly parsing and executing arithmetic expressions	Is the most essential task within the brief
	To implement variable management allowing assignment of values to variables and to use variables in expressions	An important feature needed to allow polynomials later in the project
	To develop a basic GUI that has a command prompt for user input and a text field for displaying results or errors	Essential for user interaction and error reporting
	To be able to plot both linear and polynomial functions within the GUI	The main visualization requirement, needed to visualise mathematical functions.
Should	To extend the interpreter with control flow	Implementing for loops
	To implement interactive plotting features E.g. zooming in and out	Enhances the user experience by allowing the user to explore the plane
Could	To visualize the parse tree	Helpful for debugging the parsing logic
	To implement GPU acceleration	To optimise the rendering of the grid line during real-time interaction
Should not	To implement a compiler/transpiler	Overly ambitious given the development time.
	To implement advanced mathematical features (differentiation/integration)	We wanted to ensurte the core intpreter was robust and also dropped to the development time.

Chapter 2

Background

The development of maths platforms and plotting software has a lot of history, ranging from early command line maths tools to web based educational tools. Desmos is one of such tools that represents a modern approach to mathermatical softwate. Desmos was launched in 2011, it is a web-based graphing calculator built mainly in Javascript [Desmos Studio PBC, 2023]. It mainly focuses on graphing and visualization allowing a user to input complex equations instantly in a browser, furthermore it is a great educational tool allowing students to get instant feedback making it more engaging to learn.

Matlab is a programming and numeric computing platform used to analyze data, develop algorithms and create models [MathWorks ®, 2023]. Matlab was originally released in the late 1970's by Cleve Moler, it was designed to be a interactive matrix calculator [Wikipedia contributors, 2026]. Its launch as a commercial product in 1984 by MathWorks gave matlab a significant redesign transforming it into what it is today allowing not only matrix operations but also plotting capabilities and algorithm implementation.

The project follows some of the methodologies outlined in Crafting Interpreters by Robert Nystrom [Nystrom, 2021]. Nystrom's intpreter is implemented in java using an object oriented approach whilst this project adapts those concepts into a functional paradigm using Fsharp. The Abstract SyntaxTree (AST) was implemened using Fsharp Discriminated Unions. This approach allowed the data structure to mirror our Backus-Naur Form (BNF) directly.

The graphical user interface (GUI) was developed using Window Presentation Foundation (WPF) [Microsoft Learn, WPF, 2023]. The Csharp frontend invokes the Fsharp lexer and parser functions directly. One of the challenges faced in this was converting functional data types such as FsharpList into Csharp arras for the UI logic.

The graphic library Oxyplot was used for the plotting capabilities. Oxyplot is an open source .NET plotting library [OxyPlot, 2023]. It handles the rendering of polynomial coeffcients generated by our intpreter, allowing the frontend To display graphs based on the user input.

Chapter 3

Development History

Describe the history of your development in terms of the iterations or sprints in your project (your Github repository or other version control should help you to retrospectively identify these). Use different sections for different sprints and subsections for specific details on the same sprint. Feel free to use subsubsections or paragraphs (which are not numbered) if needed.

3.1 Sprint 1: Basic expressions and GUI

This was the starting point of the project, focusing on the development of a Graphic User Interface (GUI) using Window Presentation Format (WPF). A challenge encountered during this phase was the teams unfamiliarity with the WPF framework. This was overcome by consulting the technical documentation [Microsoft Learn, WPF, 2023]. The result was a basic functional frontend capable of accepting basic user input.

3.1.1 Grammar in BNF

```
<E>    ::= <T> <Eopt>
<Eopt> ::= "+" <T> <Eopt> | "-" <T> <Eopt> | <empty>
<T>    ::= <NR> <Topt>
<Topt> ::= "*" <NR> <Topt> | "/" <NR> <Topt> | <empty>
<NR>   ::= "Num" <value> | "(" <E> ")"
```

3.1.2 Basic GUI

We used WPF with C# to develop a basic GUI - see Figure 3.1.

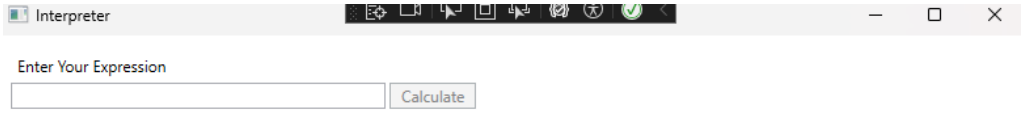

Figure 3.1: A very basic GUI!

3.1.3 Testing

A summary of the critical test cases is presented in Table 3.3. These results represent a subset of the complete test suite found in Table B.2 (Appendix B).

Expression	ResE	ResA	Status
`9 - 3 - 2`	4	4	Pass
`5 + 3`	8	8	Pass
`+ 3`	`Error`	`Syntax Error`	Pass

Table 3.1: Key testing outcomes (Subset of full data in Table B.2).

3.2 Sprint 2: Adding unary minus, powers and mod

Building upon the basic arithmetic implemented in Sprint 1, this sprint focused on expanding the interpreter's mathematical capable to support more complex expressions. The main objective was to implement exponents(^), the modulues operator (mod) and unary minus. There were two major issues in this sprint which were operator precedence and unary minus representing both subtraction and negation. These were overcome by restructuring the parser to handle operator precedence correctly and changing the parser logic to distinguish between subtraction and negation.

3.2.1 BNF

```
<E>    ::= <T> <Eopt>
<Eopt> ::= "+" <T> <Eopt> | "-" <T> <Eopt> | <empty>
<T>    ::= <U> <Topt>
<Topt> ::= "*" <U> <Topt> | "/" <U> <Topt> | "%" <U> <Topt> | <empty>
<U>    ::= "-" <U> | <P>
<P>    ::= <NR> <Popt>
<Popt> ::= "^" <NR> <Popt> | <empty>
<NR>   ::= "Num" <value> | "(" <E> ")"
```

3.2.2 Updated GUI

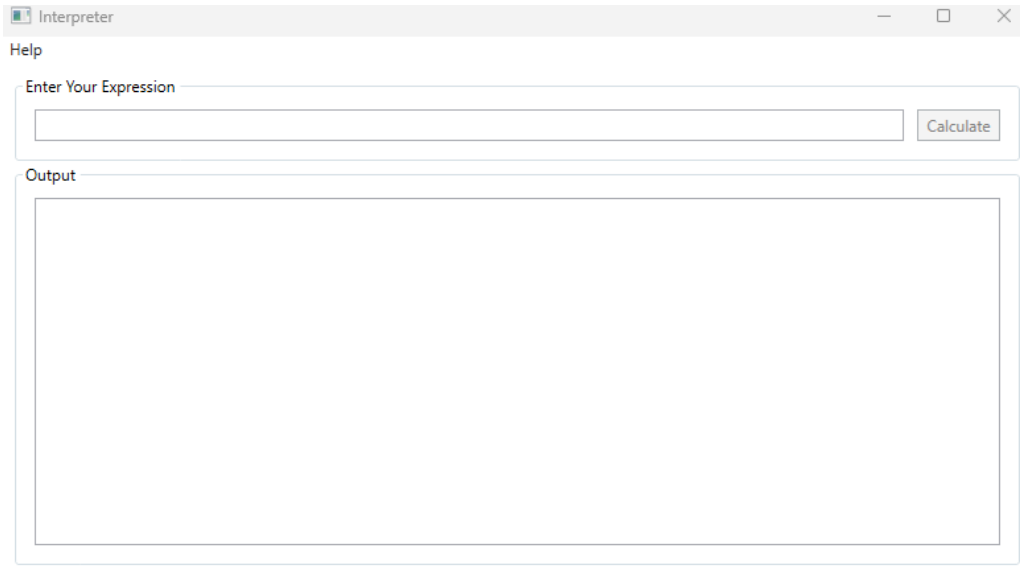

Figure 3.2: Update GUI

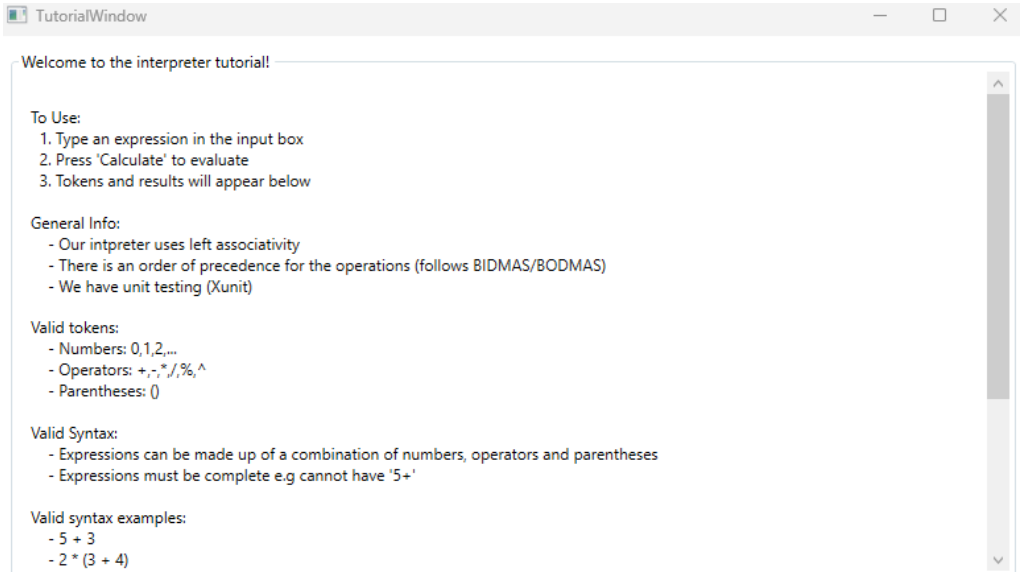

Figure 3.3: Tutorial Page

3.2.3 Testing

A summary of the critical test cases is presented in Table 3.3. These results represent a subset of the complete test suite found in Table B.2 (Appendix B).

Expression	ResE	ResA	Status
$10/3$	`3`	`3`	Pass
`10 - - 2`	`12`	`12`	Pass
$2 * 3^2$	`18`	`18`	Pass

Table 3.2: Key testing outcomes (Subset of full data in Table B.2).

3.3 Sprint 3: Added floating point

Sprint 3 addressed the limitation of interger only arithmetic by introducing floating poinr numbers. This was done by modifying the lexer to recognize decimal points and parse fractional values. There was also an update to the BNF, splitting number into distinct integer (`<IN>`) and Float (`<FL>`) definitions.

3.3.1 BNF

```
<E>    ::= <T> <Eopt>
<Eopt> ::= "+" <T> <Eopt> | "-" <T> <Eopt> | <empty>
<T>    ::= <P> <Topt>
<Topt> ::= "*" <P> <Topt> | "/" <P> <Topt> | "%" <P> <Topt> | <empty>
<P>    ::= <U> <Popt>
<Popt> ::= "^" <U> <Popt> | <empty>
<U>    ::= "-" <U> | <NM>
<NM>   ::= <IN> | <FL> | "(" <E> ")"
<IN>   ::= <digit>+
<FL>   ::= <digit>+ "." <digit>+
```

3.3.2 Testing

A summary of the critical test cases is presented in Table 3.3. These results represent a subset of the complete test suite found in Table B.2 (Appendix B).

Expression	**ResE**	**ResA**	**Status**
`10/3.0`	`3.333`	`3.333`	Pass
`3 + 1.1`	`12`	`12`	Pass
$2^{1.1}$	`2.144`	`2.144`	Pass

Table 3.3: Key testing outcomes (Subset of full data in Table B.2).

3.4 Sprint 4: Added linear plotting

Sprint 4 focused on transforming the project into a visual graphic tool. The main objective was the intergration og the OxyPlot library [OxyPlot, 2023] into the frontend. The Csharp was significantly updated to translate constant functions e.g. y=5, this established a good framework for more complex vizulationsn in later springs.

3.4.1 Updated GUI

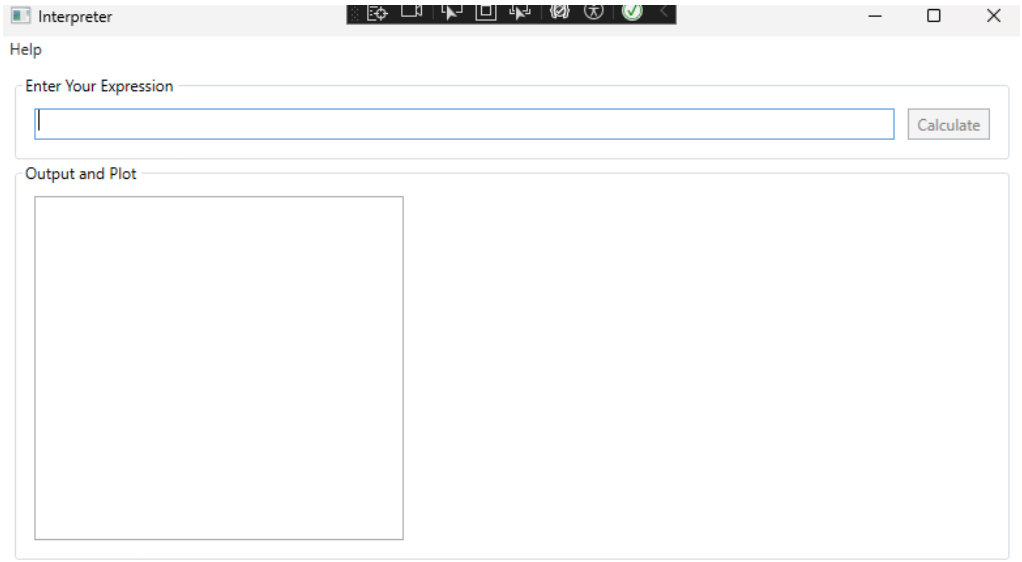

Figure 3.4: Plot GUI

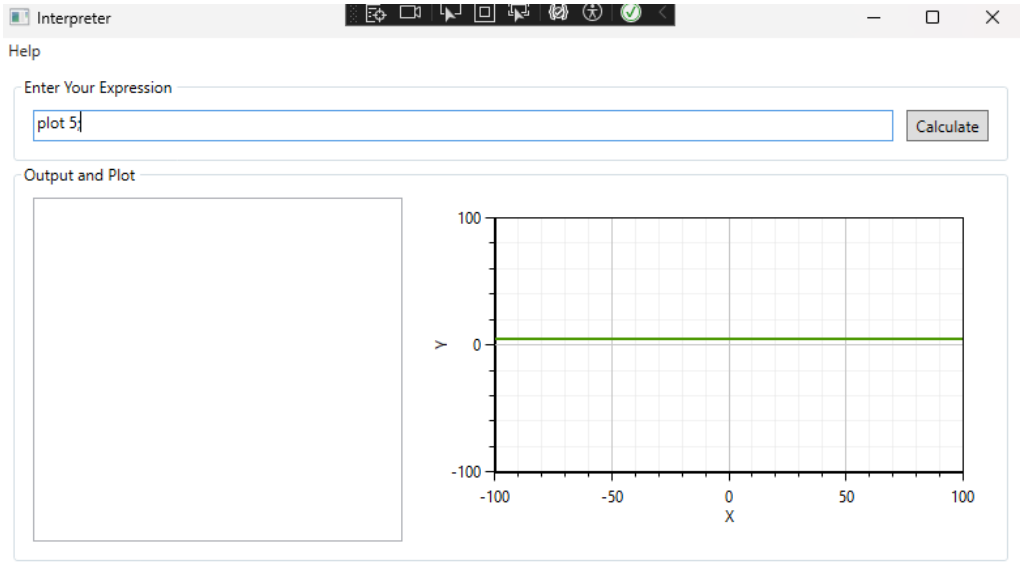

Figure 3.5: Plot GUI with test plot

3.5 Sprint 5: Added polynomial plotting

The focus of sprint 5 was to extend the interpreter's capabilities to support variables and polynomial functions. To achieve this a symbol table was integrated.

3.5.1 BNF

```
STATEMENTS
<STA>     ::= ( <ASN> | <PLT> | <PTR> ) ";"
            | <STA> <STA>
             | <empty>
<ASN>     ::= "let" <SYM> "=" <NM>
<PLT>     ::= "plot" <NM> <PLTopt>
```

```
<PLTopt> ::= "," <NM> <PLopt> | <empty>
<PRT>    ::= "print" <NM>
<SYM>    ::= <alpha+>

EXPRESSIONS
<E>    ::= <T> <Eopt>
<Eopt> ::= "+" <T> <Eopt> | "-" <T> <Eopt> | <empty>
<T>    ::= <P> <Topt>
<Topt> ::= "*" <P> <Topt> | "/" <P> <Topt> | "%" <P> <Topt> | <empty>
<P>    ::= <U> <Popt>
<Popt> ::= "^" <U> <Popt> | <empty>
<U>    ::= "-" <U> | <NM>
<NM>   ::= <IN> | <FL> | <SYM> | "(" <E> ")"
<IN>   ::= <digit+>
<FL>   ::= <digit+> "." <digit+>
```

3.5.2 Updated GUI

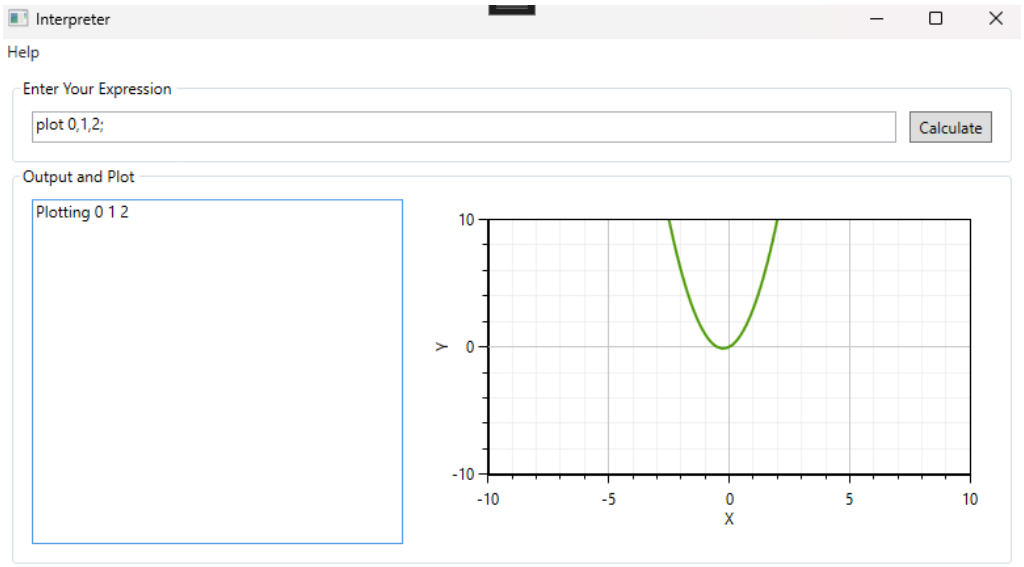

Figure 3.6: Plot GUI with a parabola

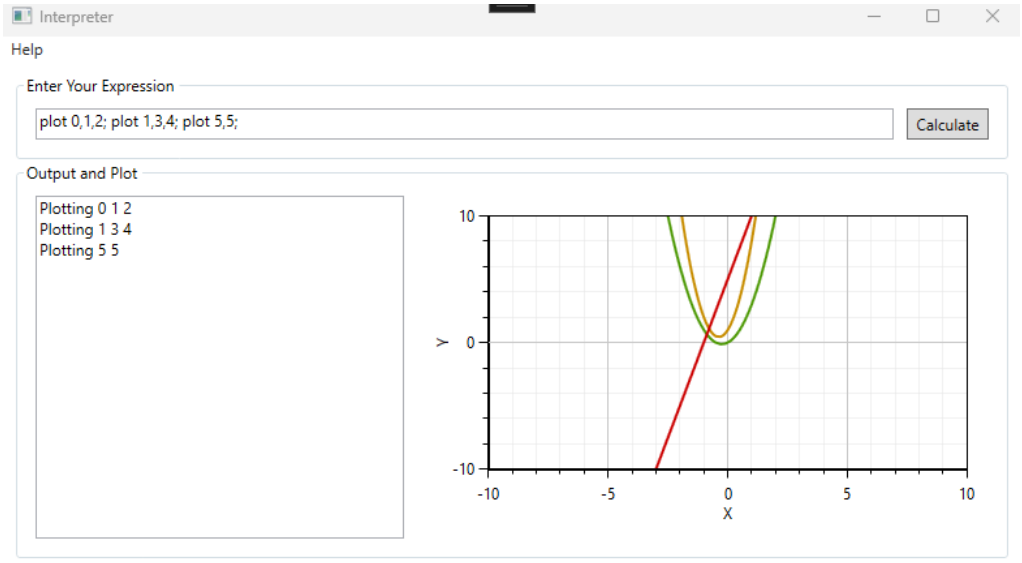

Figure 3.7: Plot GUI with multiple lines and parabola

Chapter 4

Final deliverable

In this chapter you cover the final or "ultimate" version of your project. It will show the final BNF, the final GUI, the architecture (which should be MVVM or MVC) that includes UML diagrams, additional algorithms if not already included in the previous sprint sections.

4.1 Final BNF

4.2 Final GUI

See Figure 4.1.

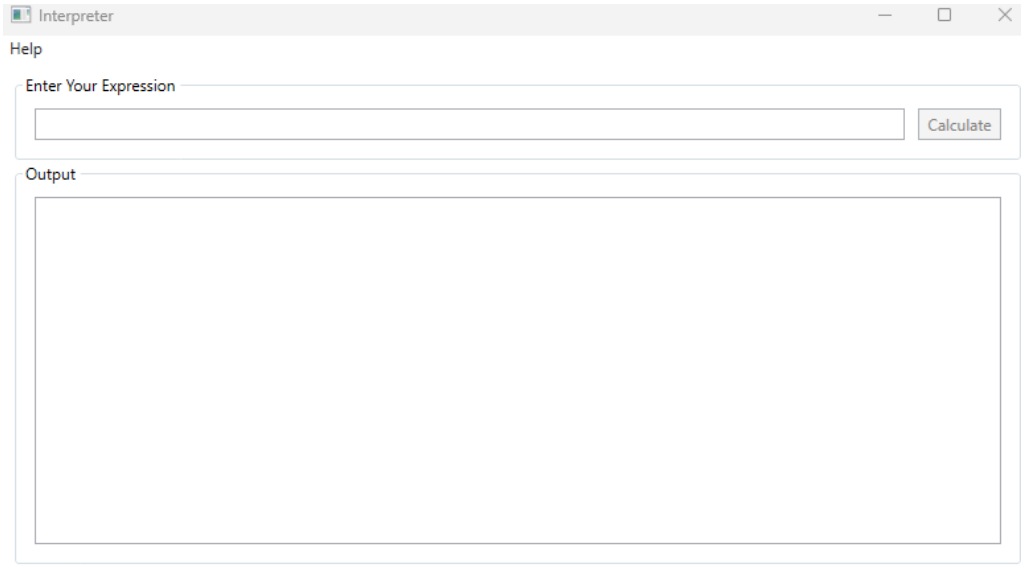

Figure 4.1: A potentially final GUI!

4.3 Code architecture

Fig. 4.2 shows a UML class diagram (class, sequence and state diagrams are the most frequently used UML diagrams). Illustrating your code architecture - that should be of the MVC family and, considering it is developed in C# with WPF more specifcally the MVVM pattern - is very important.

4.4 Algorithms

Algorithms can be described in this chapter if not already covered in previous sections. Pseudo-code is preferred over code snippets. If you use the latter then make sure it is well

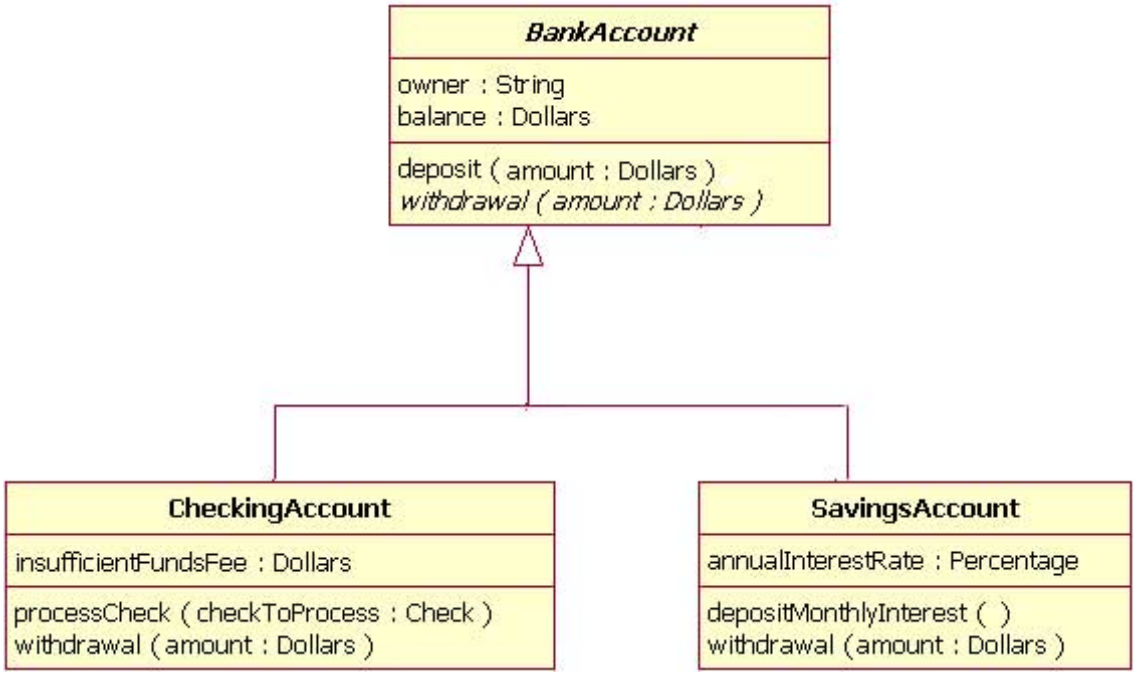

Figure 4.2: A UML class diagram to be replaced with yours!

commented inside the code or via the figure caption.

Algorithm 1 The Newton-Raphson method

1: Initialise root based on estimate
2: Set stop criterion
```
const double error = 0.000001;
```
3: **while** stop criterion not met **do**
4: Compute f(root)
5: Compute f'(root)
6: root := root - f(root)/f'(root)
7: **end while**

Note that code snippets or lists of crucial programming code or large UML diagrams should go in Appendix **??** (or further appendices).

4.4.1 Testing

Extensive unit testing as been written for all aspects of the interpreter.

Describe what testing you have done on the interpreter (lexer, parser and execution), GUI and GUI-Interpreter communication, plotting, etc. Table B.2 in Appendix B should be completed to do basic arithmetic expression tests.

Chapter 5

Discussion, conclusion and future work

Briefly discuss your achievements and put them in perspective with the MoSCoW analysis you specified in Table 1.1. Also discuss future developments and how you see the deliverable improving if more time could be spent. Note that this section should not be used as a medium to vent frustrations on whatever did not work out (group issues, not enough time, illness, etc.) as this should be dealt with separately - keep it professional!

Bibliography

[Desmos Studio PBC, 2023] Desmos Studio PBC (2023). Desmos website. https://desmos.com [Accessed: 30/11/2023].

[MathWorks ®, 2023] MathWorks ® (2023). Matlab website. https://uk.mathworks.com/products/matlab.html [Accessed: 30/11/2023].

[Microsoft Learn, WPF, 2023] Microsoft Learn, WPF (2023). *Windows Presentation Foundation documentation*. Microsoft.

[Nystrom, 2021] Nystrom, R. (2021). *Crafting Interpreters*. Genever Benning, Great Brittain.

[OxyPlot, 2023] OxyPlot (2023). Oxyplot: A cross-platform plotting library for .net. Accessed: 2023-11-30.

[Wikipedia contributors, 2026] Wikipedia contributors (2026). Matlab - wikipedia. https://en.wikipedia.org/wiki/MATLAB.

Appendix A

Contributions

A.1 Individual Contributions

Name	Contribution
Anthony de Cruz	50%
Mason Buckle	50%

Both members equally contributed to the project. We both worked on designing the application/BNF in equal time as well as performing testing of the application. Anthony developed most of the F# interpreter and Mason developed the plotting and GUI in C#. This complete report was also written in equal parts by both members with Mason writing chapters 1, 2 and 3 and Anthony writing chapters 4 and 5.

Appendix B

Testing

B.1 Arithmetic expression testing

Table B.1: Original set of arithmetic expression tests. Note that floating pointing values are accurate to three decimal places for the fractional part. ResE is expected result and ResA is actual result.

Expression	ResE	ResA	Pass/Fail	Action/comment
$5*3+(2*3-2)/2+6$	23	23	PASS	...
$9-3-2$	4	4	PASS	left assoc.
$10/3$	3	3	PASS	int division
$10/3.0$	3.333	3.333	PASS	float division
$10\%3$	1	1	PASS	
$10--2$	12	12	PASS	unary minus
$-2+10$	8	8	PASS	
$3*5\hat{}(-1+3)-2\hat{}2*-3$	87	87	PASS	power test
$-3\hat{}2$	-9(*) or 9	9	PASS	precedence
$-7\%3$	2(*) or -1	-1	PASS	precedence (*)Python
$2*3^2$	18	18	PASS	precedence pow ¿ mult
$3*5\hat{}(-1+3)-2\hat{}-2*-3$	75.750 or 75	75	PASS	
$3*5\hat{}(-1+3)-2.0\hat{}-2*-3$	75.750	75.750	PASS	
$(((3*2--2)))$	8	8	PASS	
$(((3*2--2))$	Error	Syntax Error	PASS	syntax error
$-((3*5-2*3))$	-9	-9	PASS	minus expression
$x=3;(2*x)-x\hat{}2*5$	-39	39	PASS	var assign
$x=3;(2*x)-x\hat{}2*5/2$	-16	-16	PASS	
$x=3;(2*x)-x\hat{}2*(5/2)$	-12	-12	PASS	
$x=3;(2*x)-x\hat{}2*5/2.0$	-16.5	-16.5	PASS	
$x=3;(2*x)-x\hat{}2*5\%2$	5	5	PASS	
$x=3;(2*x)-x\hat{}2*(5\%2)$	-3	-3	PASS	

Table B.2: Boolean expression & statement tests.

Expression	ResE	ResA	Pass/Fail	Action/comment
5 > 3	1	1	PASS	
3 > 5	0	0	PASS	
3 < 5	1	1	PASS	
5 < 3	0	0	PASS	
2 == 2	1	1	PASS	
2 == 1	0	0	PASS	
2! = 2	0	0	PASS	
2! = 1	1	1	PASS	
!1	0	0	PASS	
!0	1	1	PASS	
1 and 1	1	1	PASS	
0 and 1	0	0	PASS	
1 and 0	0	0	PASS	
0 and 0	0	0	PASS	
5 > 2 and 10 == 4	0	0	PASS	
5 > 2 and 10! = 4	1	1	PASS	
1 or 1	1	1	PASS	
0 or 1	1	1	PASS	
1 or 0	1	1	PASS	
0 or 0	0	0	PASS	
5 > 2 or 10 == 4	1	1	PASS	
5 > 2 or 10! = 4	1	1	PASS	
>	Error	Syntax Error	PASS	
< 3	Error	Syntax Error	PASS	
!	Error	Syntax Error	PASS	
3 == 5! =	Error	Syntax Error	PASS	
i = 0; while i ¡ 5 { i = i + 1; } y = i;	5	5	PASS	
i = 7; if i == 7 { y = 8; }	8	8	PASS	
i = 0; while i ¡ 5 y = i;	Error	Syntax Error	PASS	
i = 0; if i ¡ 5 y = i;	Error	Syntax Error	PASS	
i = 0; while { y = i; }	Error	Syntax Error	PASS	
i = 0; if { y = i; }	Error	Syntax Error	PASS	

Table B.3: Expression tests.

Expression	ResE	ResA	Pass/Fail	Action/comment
$5 + 3$	8	8	PASS	
$200 + 13 + 45$	258	258	PASS	
$3 + 1.1$	4.1	4.1	PASS	
$5 + 3$	8	8	PASS	
$+$	Error	Syntax Error	PASS	
$+3$	Error	Syntax Error	PASS	
$3+$	Error	Syntax Error	PASS	
$3 + 5+$	Error	Syntax Error	PASS	
$3 * 3$	9	9	PASS	
$8 * 4 * 3$	96	96	PASS	
$3 * 1.1$	3.3	3.3	PASS	
$3.256 * 1.59$	5.177	5.177	PASS	
$*$	Error	Syntax Error	PASS	
$*3$	Error	Syntax Error	PASS	
$3*$	Error	Syntax Error	PASS	
$3 * 5*$	Error	Syntax Error	PASS	
$6/3$	2	2	PASS	
$5/3.0$	1.667	1.667	PASS	
$12/3/2$	2	2	PASS	
$3.2/2$	1.6	1.6	PASS	
$3.4/2.3$	1.478	1.478	PASS	
$/$	Error	Syntax Error	PASS	
$/3$	Error	Syntax Error	PASS	
$3/$	Error	Syntax Error	PASS	
$3/5/$	Error	Syntax Error	PASS	
$3/0$	Error	Divide By Zero	PASS	
$3/0.0$	Error	Divide By Zero	PASS	

Table B.4: Expression tests continued.

Expression	ResE	ResA	Pass/Fail	Action/comment
6%3	0	0	PASS	
5%3	2	2	PASS	
19%5%3	1	1	PASS	
3.2%2	1.2	1.2	PASS	
3.4%2.3	1.1	1.1	PASS	
%	Error	Syntax Error	PASS	
%3	Error	Syntax Error	PASS	
3%	Error	Syntax Error	PASS	
3%5%	Error	Syntax Error	PASS	
3%0	Error	Divide By Zero	PASS	
3%0.0	Error	Divide By Zero	PASS	
5^2	25	25	PASS	
5^2^2	625	625	PASS	
2^1.1	2.144	2.144	PASS	
^	Error	Syntax Error	PASS	
^3	Error	Syntax Error	PASS	
3^	Error	Syntax Error	PASS	
3^5^	Error	Syntax Error	PASS	
-2	-2	-2	PASS	
$5--2$	7	7	PASS	
$5+-2$	3	3	PASS	
$5+--3$	8	8	PASS	
2^--2	0	0	PASS	
2^--2	4	4	PASS	
$-$	Error	Syntax Error	PASS	
$5++-3$	Error	Syntax Error	PASS	
2^-	Error	Syntax Error	PASS	
$2--+--2$	Error	Syntax Error	PASS	

Table B.5: Lexer tests for operators, floating point numbers, and symbols.

Expression	ResE	ResA	Pass/Fail	Action/comment
3 + 5	[Int 3; Add; Int 5]	[Int 3; Add; Int 5]	PASS	
1 * 2	[Int 1; Mul; Int 2]	[Int 1; Mul; Int 2]	PASS	
3 ^ 8	[Int 3; Pwr; Int 8]	[Int 3; Pwr; Int 8]	PASS	
3 £ 4	Error	Syntax Error	PASS	
5 5 :	Error	Syntax Error	PASS	
3.8	[Flt 3.8]	[Flt 3.8]	PASS	
3.008	[Flt 3.008]	[Flt 3.008]	PASS	
3.811	[Flt 3.811]	[Flt 3.811]	PASS	
0.811	[Flt 0.811]	[Flt 0.811]	PASS	
100.001	[Flt 100.001]	[Flt 100.001]	PASS	
7.	Error	Syntax Error	PASS	
.7	Error	Syntax Error	PASS	
5 .2	Error	Syntax Error	PASS	
2. 5	Error	Syntax Error	PASS	
x	[Sym "x"]	[Sym "x"]	PASS	
y	[Sym "y"]	[Sym "y"]	PASS	
varname	[Sym "varname"]	[Sym "varname"]	PASS	
myvar	[Sym "myvar"]	[Sym "myvar"]	PASS	
3 + variable	[Int 3; Add; SymT "variable"]	[Int 3; Add; Sym "variable"]	PASS	

B.2 GUI testing

Table B.6: A series of manual/visual tests to determine functionality.

Action	ResE	ResA	Pass/Fail
Write code into main text field and hit the run button.	The program is executed, with expected plots and output.	The program is executed as expected.	PASS
Press the clear button.	Any existing plots should be cleared.	Existing plots are cleared.	PASS
Press the open button.	A dialogue is shown allowing the user to select a text file to load into the buffer.	A file is loaded into the text buffer via a dialogue.	PASS
Press the save button.	The existing text buffer should be saved to a text file via a dialogue.	The current text buffer is written to a text file.	PASS
Press the help button.	The tutorial window should be displayed.	The tutorial window is displayed.	PASS

B.3 Plot testing

Table B.7: A Series of manual/visual tests to determine functionality.

Action	ResE	ResA	Pass/Fail
Execute "plot x + 2;" and set minimum X = -10, maximum X = 10, resolution = 10. Observe the plotted values in the GUI.	Where Y = 0, X should be 2. Where Y = 1 X should be 3.	Where Y = 0, X = 2. Where Y = 0, X = 3.	PASS
Execute "plot x^2 - 2;" and set minimum X = -10, maximum X = 10, resolution = 10. Observe the plotted values in the GUI.	Where Y = 0, X = -2. Where Y = 5, X = 23.	Where Y = 0, X = -2. Where Y = 0, X = 23.	PASS