CMP-6048A Advanced Programming

Project Report - 13 January 2025

Maths Interpreter software

Group members:
Connor Stansfield, Ethan Colman, Taylor Holton, Aaron Hurrell

School of Computing Sciences, University of East Anglia

Version 2.0

Abstract

Please replace this section with your own abstract. An abstract is a brief summary (maximum 250 words) of your entire project. It should cover your objectives, your methodologies used, a brief developmental history, your final results, in particular covering the optional tasks, and a discussion and conclusion. You do not cover the literature or background in an abstract nor should you use abbreviations or acronyms. The remainder of this report template has clear chapter titles and we suggest to stick to these although you can organise your material inside each chapter to your own preferences. A guideline in size is approximately 3,500 words (not including abstract, captions and references) but no real limit on figures, tables, diagrams, pseudo-code etc.

Chapter 1

Introduction

The introduction should be brief and comprise the following:

1.1 Project statement

A (brief) statement including the nature of the project, i.e. what you will do, how you will accomplish it and what the final deliverable will be.

1.2 Aims and objectives

Clearly specify the main aim and objective(s) and subsequent tasks of your project. You can use a MoSCoW to further elaborate on each individual task. If you do so, use a table with three columns (e.g. Table 1.1), the first one having the four categories; the second column a brief description of the task and the third column further elaboration or comments on the task. Note that a MoSCoW is normally for functional requirements only. You should treat non-functional requirements in a separate MoSCoW.

Table 1.1: MoSCoW

Priority	Task	Comments
Must	Task M1	Comment M1
	Task M2	Comment M2
	...	...
Should	Task S1	Comment S1
	Task S2	Comment S2
	...	...
Could	Task C1	Comment C1
	Task C2	Comment C2
	...	...
Should not	Task SN1	Comment SN1
	...	...

Chapter 2

Background

Give a brief background on similar software, e.g. [**?**], [**?**], etc. Also cite the books [**?**] or documentation [**?**] that you consulted. You should add additional references to the corresponding bib file (References.bib) referred to in the bottom of this document.

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

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.

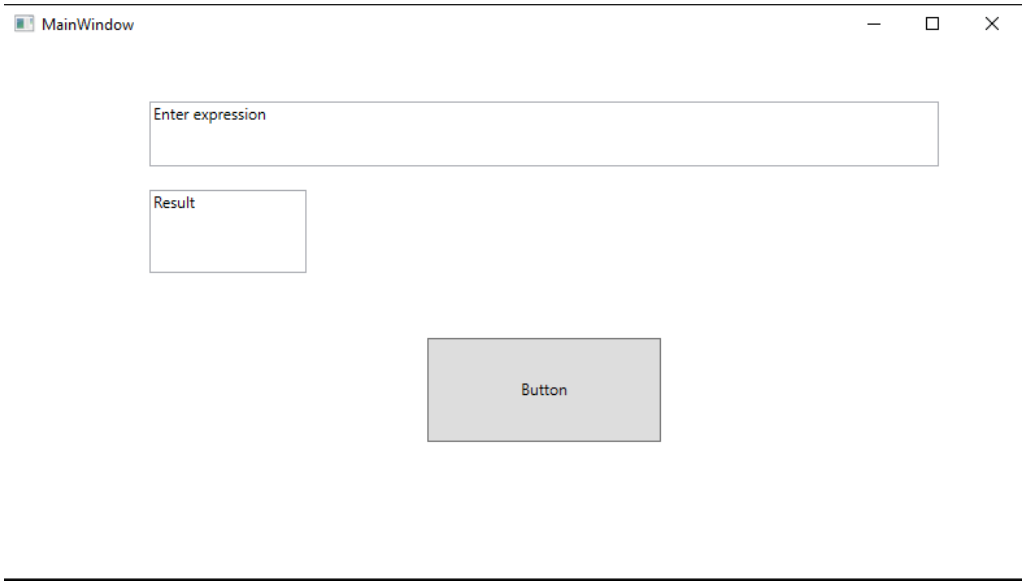

Figure 3.1: A very basic GUI!

3.1.3 Testing

A subset of Table B.1 in Appendix B could be referred to from here.

3.2 Sprint 2: Adding unary minus, powers and mod

3.2.1 BNF

3.2.2 Updated GUI

3.3 Sprint 3: ...

3.4 Sprint n: Your penultimate version

Make sure that the total number of sprints n(+1) does not become too unwieldy. A rule of thumb would expect it to be somewhere between 6 and 12.

3.4.1 BNF

3.4.2

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.

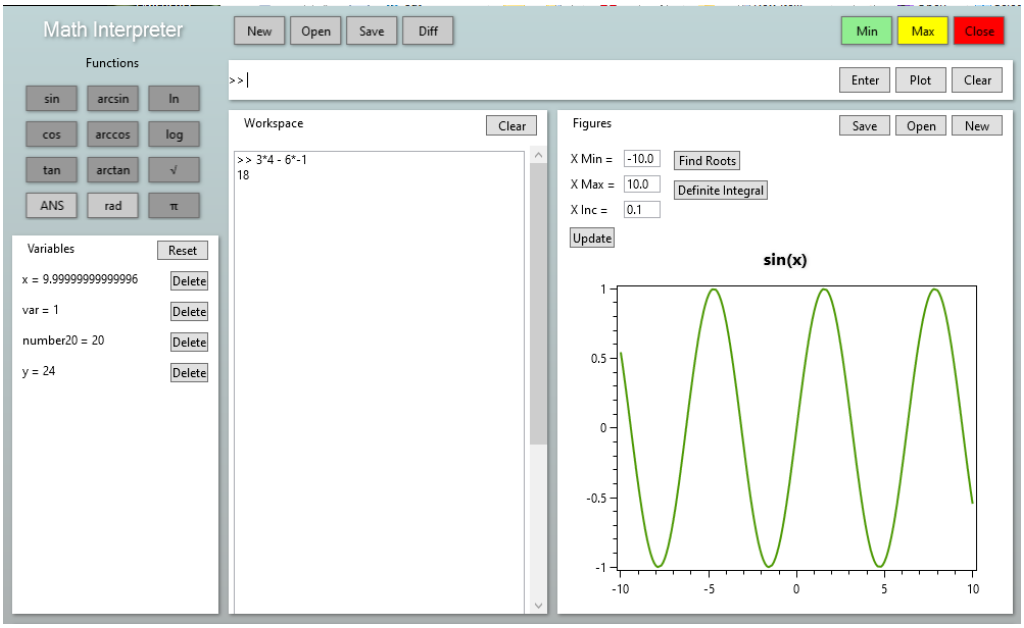

Figure 4.1: A potentially final GUI!

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.

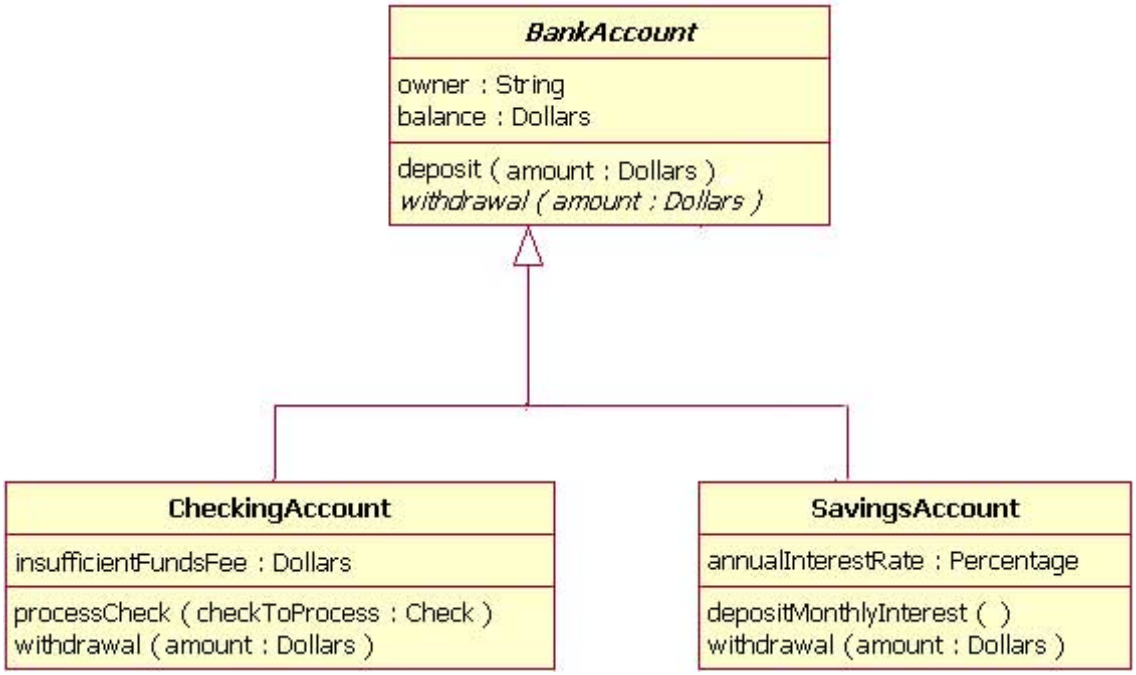

Figure 4.2: A UML class diagram to be replaced with yours!

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 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 C (or further appendices).

4.4.1 Testing

Describe what testing you have done on the interpreter (lexer, parser and execution), GUI and GUI-Interpreter communication, plotting, etc. Table B.1 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!

Appendix A

Contributions

State here the % contribution to the project of each individual member of the group and describe in brief what each member has done (if this corresponds to particular sections in the report then please specify these).

Appendix B

Testing

B.1 Arithmetic expression testing

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

B.2 GUI testing

B.3 Plot testing

Appendix C

Other stuff