CMP-6048A Advanced Programming

Project Report - 15 December 2021

Maths Interpreter Software

Group members:
James Mason and Nia Preston

School of Computing Sciences, University of East Anglia

Version 1.0

Abstract

An abstract is a brief summary (maximum 250 words) of your entire project. It should cover your objectives, your methodology used, how you implemented the methodology for your specific results and what your final results are, your final outcome or deliverable and conclusion. You do not cover literature reviews or background in an abstract nor should you use abbreviations or acronyms. In the remainder of the report the chapter titles are suggestions and can be changed (or you can add more chapters if you wish to do so). This template is designed to help you write a clear report but you are welcome to modify it (at your peril ...). Finally, a guideline in size is approximately 3,500 words (not including abstract, captions and references) but no real limit on figures, tables, etc.

Chapter 1

Introduction

1.1 Project statement

The purpose of this project is to create a piece of software that is an interpreter for mathematical functions. It will run both as an interactive prompt and a utility to interpret and execute pre-written files. It will additionally provide the ability to visualise mathematical functions such as on a graph.

1.2 MoSCoW

1.2.1 Musts

These elements largely comprise the minimum functional requirements of the program and are as such the most neceassary items.

- Expressions
- Statements
- Commands
- Custom syntax using a custom lexer and parser

1.2.2 Shoulds

- Function visualisation
- Intuitive UI
- Flexible and scalable architecture
- Parallel processing of intensive operations
- Simple trigonometric functions

1.2.3 Coulds

- Interactive visualisations
- Zero-crossings finder
- Differentiation and integration of functions
- Simple equation solver (for example finding a variable)

1.2.4 Won'ts

- Imaginary numbers
- 3D function graphing
- Multi-variable equation solver (beyond simple simultaneous equations)

1.3 Report structure

Breifly describe what you will cover in the remainder of the report, chapter by chapter.

Chapter 2

Background

To better understand what we do and don't want from our software, we analysed systems on the market that have a similar function to our own. We analysed the good and bad features of each and identfied any we would take forward into our own product.

2.1 Similar Systems

2.1.1 MATLAB[1]

MATLAB is a standalone piece of mathematical software designed with a focus on manipulation of mathematical data types such as matrices as well as including generic programming. It is a turing complete language, so it is has a very wide range of possible uses, but it is very much geared towards easily running numeric calculations. It has its own syntax, which means that users will have to learn it, but the custom syntax it uses is very similar to many popular programming languages, and so users with that background may find transitioning easier. From this program, we like the layout of the user interface. It is intuitive and everything you ened is in easy reach. This is the sort of UI we would liek to implement in our own product. However, this program takes up alot of storage and does not run well on lower specification machines. We would like our own product to be able to run on most desktop devices so it is more accessible.

2.1.2 Math Inspector[2]

Math Inspector is a package of addons for Python which allow for mathematical functions to be graphically visualised, both by displaying a flow-type diagram of the function as well as graphing the numeric outputs across a range. Since it is an addon for Python, people who are familiar with the language already shouldn't have much difficulty using it. It may, however, mean that people have to install more than what they need in order to get it to run. Whilst the python basis may make it easier for some people, many who wish to use this program will have no prior python knowledge. For this reason, we feel the benefits of a familiar language are outweighed bythe cost of having to install the extra programs. Therefore, we will not be doing something similar in own own system.

2.1.3 Wolfram Mathematica[3]

Wolfram Mathematica is an interactive tool used to define and visualise mathmatical functions. It's main foucs is using technology to visualise mathmatics and to allow the user to directly implement a vast variety of features. It offers over 5000 different in-built maths functions, meaning it contains very few restrictions. It runs on it's own in-built language which may take the user some time to learn and become familiar with. Its' main strength is definately its advanced visulisation features. The vast variety of tools it offers in nice but will be a waste for most people. For our system, we will focus on the most common maths functions to maximise use whilst minimising memory cost. In the future,

our system could use addons to cover more areas of mathematics without the user having to install hundreds of tools they will never need.

2.1.4 Maple[4]

Maple offers a number of different versions of its' software allowing the user to tailor their experience before they even download. It's main goal is solving equpations but it still offers an intuetive UI with function visulisation features. It allows the user to explore a range of mathmatical fields by offering a large variety of in-built functions. Similarly to the previous system, it operates using it's own programming language which may take users time to learn and understand. The different versions of the software (tailored based on who is using it) is this systems' main strength. Our system should also include an intuative user interface. The different systems versions are a definate positive and something we would take forward in furture versions of our product. For now, our product will focus on the most commonly used areas of mathmeatics to maximise usability.

Chapter 3

Methodology

Describe here various methods that will be used in your project. Use different sections for distinctly different subjects and use subsections for specific details on the same subject. Only use subsubsections or paragraphs (which are not numbered) if you believe this is really necessary. Since implementation will happen in sprints, this section may need several updates with parts being added and deleted across the project.

3.1 Method 1

3.1.1 Method 1 specific detail 1

In case you need maths, here is an example to write an equation:

$$\int_{\Omega_0} \delta u \frac{\partial \mathbf{P}}{\partial X} d\Omega_0 + \int_{\Omega_0} \delta u \mathbf{b} d\Omega_0 + \int_{\Omega_0} \delta u \rho_0 \ddot{\mathbf{u}} d\Omega_0 = 0 \tag{3.1}$$

And here we show how to write a matrix equation:

$$\mathbf{X} \frac{\partial N}{\partial e_c} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & -1 & -1 \end{bmatrix} \tag{3.2}$$

3.1.2 Method 1 specific detail 2

blablabla

b lablabla

3.2 Method 2

3.2.1 Method 2 specific detail 1

3.3 Etc.

3.3.1 Etc.

Chapter 4

Implementation

In this chapter you cover the actual implementation of your project. Implementation will be done in sprints so you may wish to use different sub-sections for each sprint and then dedicate a section to your final deliverable. Section **??** with figures should not remain in the final report.

4.1 Early sprints

4.1.1 Sprint 1

For our first sprint, we defined the following set of requirements:

1. Create the program skeleton we can build upon
2. Build the Lexer and parser for our language
3. Allow the program to identify integers
4. Allow the program to recognize basic operators, including addition, multiplication, subtraction, division and brackets
5. The program should be able to execute the basic operation is can identify

Requirement 1:

The program now contains the different classes it will need to run. These include the Lexer Class which is the class used to identify the different tokens, The parser class that checks the syntax of the languages and the execution class that runs through the stack and performs the operations held there in the correct order. There is also the main class that is responsible for the input display and parsing that input to the Lexer.

Requirement 2:

Once all the classes were created, we were able to build the Lexer and parser for our language. The Lexer can correctly identify integers, plus signs, minus signs, multiplication and division signs and brackets as well as ignoring blank space. The only limitation of our current Lexer is that it is unable to handle integer outside of the integer limit imposed by Java.

The parser can check the syntax of the language by making sure the tokens are written the correct order and lets the user know if there is an issue.

Requirement 3:

When the Lexer encounters a digit character, a loop begins and checks each concurrent characters' digit status. If the next character is a digit also, the loop continues. When the Lexer encounters a non-digit character, the loop ends and a number token is added to

the token array and the string of digits is converted into a numerical representation which is put into the symbol table. The symbol table is implemented as a map. The index key for the numeric value is the index in the array of the corresponding number token.

Requirement 4:

The recognition of basic operators works in the same manner as the integer recognition except without the need for a looping function. Each operator is assigned a given token which is added to the token array upon recognition.

Requirement 5:

Each token from the token array is copied onto the operator stack and the integers corresponding to the number tokens are copied onto the number stack from the symbol table. When a complete expression has been transferred onto the stack, it will be evaluated and the result of that expression with replace the operands used on the number stack. Any operators used are removed from the stack. Token will then continue to be copied onto the two stacks until the end of the token array is reached.

When the program encounters a right bracket, we know a left bracket must exist on the stack as the parser has validated the code. This means that when a right bracket is encountered, the program continually runs calculate until the left bracket reaches the top of the stack. This ensures the number at the top of the number stack is the evaluation of the expression within brackets, ensuring the order of operations is preserved.

4.1.2 Sprint 2

For our second sprint, we defined the following set of requirements:

1. Allow the program to recognise variable identifiers
2. Be able to assign value to a variable
3. The program should be able to evalute expression that contain pre-defined variables

Requirement 1:

The lexer class cycles through the chracters typed by the user. If the character is not a number or a reserved symbol, the program identifies it as a variable. the concecutive characters are then checked and added to the variable name if they are alphanumeric. This process ends when a non-alphanumeric charcter is reached. The symbol table has also been updated so it can hold variable names as well as numbers. This requirement has been succefully met.

Requirement 2:

To begin developing the assignmnet functionality, we have used the equals sign to represent assignment for simplicity. This sign has be added as a reserved character to identify assignment operations. When an equals gin is preceded by a varible and proceeded by an integer, the interger value is mapped to the varible. If the assignment value is an expression, the expression is evaluated and the soltuion mapped to the variable. This requirement has been successfully met.

Requirement 3:

When the program evaluates an expression that contains a varible, the execution class identifes the value mapped to the variable and replaces it in the expression. The xpression is then evalutated as normal and the solution displayed. This requirement has been succefully met.

4.1.3 Sprint n

4.2 Final implementation

4.3 Figures, tables, etc.

The purpose of this section is to just show you how to integrate figures, tables, etc. and should disappear in the final report. Figures and tables should be distributed across the document wherever they are needed but you should use an appendix if they are many figures/tables of the same kind. Fig.

Note that code snippets or lists of crucial programming code or large UML diagrams should go in the Appendix/Appendices.

	S-D		L-P old		L-P new	
Diameter	length	strain	length	strain	length	strain
MaVD	140.5	+1.90	129.3	+0.30	129.3	+1.43
OrOD	131.4	+0.10	-	-	119.9	+1.85
OrVD	126.9	+2.20	119.3	+0.25	119.3	+1.24
OFD	134.0	+0.40	-	-	119.7	+1.82
SOFD	-	-	-	-	113.2	-0.85
SOBD	117.1	-1.70	88.7	-1.07	88.7	-2.52
BPD	105.0	0.00	89.7	-0.21	89.7	-0.83

Chapter 5

Testing

This section details the testing we performed at the end of each sprint to make sure we meet each of the requirements set out at the beginning of the sprint.

5.0.1 Sprint 1

Test No.	Test Case	Test Data	Result
1	The program is able to identify a number	5	PASSED
		83	PASSED
		324876312487124	FAILED
2	The program can idenify and evaluate an addition expression	5+5	PASSED
		54+6	PASSED
		2+7	PASSED
3	The program can idenify and evaluate a subtraction expression	5-5	PASSED
		18-9	PASSED
		4-0	PASSED
4	The program can idenify and evaluate a multiplication expression	5*5	PASSED
		4*7	PASSED
		32*8	PASSED
5	The program can idenify and evaluate a division expression	5/5	PASSED
		42*7	PASSED
		100/10	PASSED
6	The program can evalute expressions that include blank space	5 + 5	PASSED
		10 - 8	PASSED
		32 - 23	PASSED
7	The program can idenify and evaluate expression with multiple operators	4+7-2	PASSED
		23+6+4	PASSED
		1*8/4	PASSED
8	The program can idenify and handle incorrect syntax	5+	PASSED
		5+(5	PASSED
		*8	PASSED
9	The program can idenify and evaluate expression involving brackets	5+(4-2+3)	PASSED
		(4*2)+7	PASSED
		10/5+(6*8)	PASSED

5.0.2 Sprint 2

Test No.	Test Case	Test Data	Result
1	The program can assign a value to a variable	x=5	PASSED
		foo=56	PASSED
		y=9	FAILED
2	The program can assign an expression to a variable	a = 19+7	PASSED
		y = 7-5	PASSED
		bar = 3*6	PASSED
3	The program can evaluate an expression that inclues a variable	x-10	PASSED
		y = 7+x	PASSED
		foo*x	PASSED
4	The program can idenifyincorrect assignment expression	5x	FAILED
		x465	FAILED
		foo x+6	FAILED

Chapter 6

Discussion, conclusion and future work

Briefly discuss and conclude your achievements and put them in perspective with the MoSCoW analysis you did early on. Be honest by declaring for example 'S' categorised objectives which did not make it to the final deliverable rather than reversely modifying your MoSCoW in Chapter 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 (pandemic, group partners, etc.) as there are other means for this (labs, e-mail MO, ...) that should be used well before any such problems become an issue.

Bibliography

[1] MathWorks, Natick, Massachusetts, United States. *MATLAB*, 2017.

[2] *Math Inspector*, 2021.

[3] Wolfram, The Wolfram Centre, Lower Road, Long Hanborough, Oxfordshire OX29 8FD, United Kingdom. *Wolfram Mathematica*, 2021.

[4] Maplesoft, 615 Kumpf Drive, Waterloo, ON, Canada, N2V 1K8. *Maple*, 2021.

[5] B. Sorbe and S. Dahlgren. Some important factors in the molding of the fetal head during vaginal delivery - a photographic study. *International Journal of Gynecology & Obstetrics*, 21(3):205–212, 1983.

[6] R. J. Lapeer and R. W. Prager. Fetal head moulding: finite element analysis of a fetal skull subjected to uterine pressures during the first stage of labour. *Journal of Biomechanics*, 34(9):1125–1133, September 2001.

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 A

Put in tables of data or protocols (e.g. for testing) or code listings or UML diagrams which may take up several pages and do not sit well in the main body text.