Group Without a Name

Arithmetic Expression Evaluator
Software Requirements Specifications

Version 2

Revision History

Date	Version	Description	Author
10/05/2023	1.0	Base iteration of document	Jake Bernard
10/19/2023	1.0.1	Clarified some specifications: - User-defined variables should not contain spaces and should only contain letters and underscores - Only expressions that are successfully evaluated should be added to the history	Jake Bernard
11/30/2023	2	Updated document for final release	Jake Bernard

Table of Contents

Software Requirements Specifications

1. Introduction

1.1 Purpose

The purpose of the *Software Requirements Specifications* is to define and describe the necessary and desired functionality of the *A2E* software. This is includes all units of desired behavior (or *features*), all non-functional requirements (or *qualities*), any intended units of user interaction (*use cases*), and any constraints in the design or runtime behavior or environment of the application.

1.2 Scope

The *Software Requirements Specification* applies to the full *A2E* application, most importantly concerned with the interface the application presents to the end user and the functionality available to the end user provided by that interface.

1.3 Definitions, Acronyms, and Abbreviations

Feature: a unit (or several units grouped tgoether) of functionality.

Quality: a description of how the system should operate that is not related to functionality.

Constraint: a limitation placed upon the design, implementation, or deployment process

Use Case: a specific interaction a user may have with the program, or sub-interaction.

GUI: Graphical user interface

For all else, see the *Project Glossary*.

1.4 References

These documents are referenced by the *Software Requirements Specifications*:

- *Project Vision*, artifacts/project_vision.pdf, Group Without a Name, 2023
- *Project Description*, artifacts/project_description.pdf, University of Kansas, 2023
- *Project Plan*, artifacts/project_plan.pdf, University of Kansas, 2023
- *Google C++ Style Guide*, https://google.github.io/styleguide/cppguide.html, Alphabet Inc., 2023
- Style Notes, team_resources/style_notes.md, Group Without a Name, 2023

1.5 Overview

The outline of the rest of the *Software Requirements Specifications* is as follows:

Overall Description — Contains general information about the application and gives context for the requirements, qualities, and constraints of the application.

Specific Requirements — Defines the requirements of the program with enough specification to enable a satisfactory implementation of the application and for which tests can be developed to determine if the requirements were satisfied.

Classification of Functional Requirements — A list of all of the functional requirements, sorted and categorized as "Necessary", "Desirable", or "Optional".

Appendices — Contains all extra information referenced within the document that did not fit within the other categories.

2. Overall Description

2.1 Product perspective

2.1.1 System Interfaces

Most interfacing with the user's operating system will be handled at runtime and is expected to be mostly implemented by the compiler as the source code is turned into machine code. The application should be an executable file, so it will interact with the operating system at runtime in various ways, such as in the creation of a window for the GUI.

2.1.2 User Interfaces

The user is presented either with a command line interface or GUI depending upon the configurable state of the program during runtime, and possibly based upon the last state the program was opened in. The command line interface affords the user with text-based input from the user's keyboard for the creation of user-defined arithmetical expressions. The desktop GUI application will afford the user with both input from the keyboard and from buttons for operations and numerals within the GUI. The user should be able to write, clear, and evaluate expressions in both scenarios. Errors in input should be handled with an appropriate alert to the user that the specified expression is not valid.

2.1.3 Hardware Interfaces

The application should be a compiled program written in C++, so all interfacing with the hardware the application runs on should be handled during compilation. The application will be designed to run on general purpose computers with keyboard and mouse input and output to a monitor.

2.1.4 Software Interfaces

There may be several libraries used in the application for handling the GUI or other related functionality.

2.1.5 Communication Interfaces

The application does not require any interfaces for outside communication during runtime.

2.1.6 Memory Constraints

The application should be able to run on any reasonably priced consumer-grade desktop or laptop computer released within the last 10 years.

2.1.7 Operations

The application should allow input of arithmetic expressions and evaluation of arithmetic functions, with various flourishes to make the process better, more feature rich, and/or more convenient to use.

2.2 Product functions

The application is expected to act as a calculator and expression evaluator for arbitrary arithmetic expressions containing any order of operations from a pre-defined set. Various extra functions may be implemented to embellish the base functionality, including possibly: user-defined variables, calculation history, and limited graphing capabilities.

2.3 User characteristics

The user is expected to have a basic knowledge of the meaning of various basic mathematical operators related to arithmetic, to be able to enter input via command line, to have familiarity with common GUI elements, and have some need for calculating arithmetical expressions.

2.4 Constraints

The application is expected to be written in C++ and be able to run on the computers at the University of Kansas.

2.5 Assumptions and dependencies

The application will depend upon the GCC compiler to compile the C++ code to a machine executable file and will be reliant on outside libraries to provide certain functionality for the GUI. GTK will most likely be used for creating the GUI. It is implicitly assumed that the PEMDAS order of operations will be used.

2.6 Requirements subsets

Functional requirements may be sub-categorized as those which deal with the runtime of the program and those which deal with the interface of the program, though there is some overlap between the two.

3. Specific Requirements

3.1 Functionality

3.1.1 Parse and evaluate user input

The application is expected to be able to take user input, either through the GUI via clickable buttons, or through command line and/or GUI as keyboard input which is then parsed as an arithmetic expression, evaluated, and an answer returned.

3.1.2 Basic operations

The application should be able to support the operations of addition (+), subtraction (-), multiplication (*), division (/), exponentiation (^), and modulo (%).

3.1.3 Order of operations

Mathematical operators should follow the PEMDAS rules when an expression is being evaluated. The modulo operator should have priority below division but above addition.

3.1.4 Parenthetical Grouping

The application should be able to evaluate nested expressions enclosed in parentheses, with the priority of expressions becoming higher with higher parenthetical grouping depth. Empty parentheses should return a syntax error.

3.1.5 Usage of Numeric Constants

The application should be able to recognize and utilize expressions involving common numeric constants, eg pi and Euler's number.

3.1.6 Usage of floating point numbers

The application should accept both integers and floating point numbers as input.

3.1.7 Error Handling

Syntax and arithmetic errors hould be handled within the program gracefully and the user should be notified of the error as the application continues functioning.

3.1.8 Command-Line Interface

A command-line interface should be available to the user for the input of expressions and for displaying the results of evaluated expressions

3.1.9 Alternative syntax for certain operators

The software should include alternative syntax like ** for exponentiation or [ ] for parentheses.

3.1.10 Advanced operations

The application should have support for more advanced mathematical functions/operators, eg trigonometric functions, logarithms, roots, etc.

3.1.11 Multiple Numerical Modes

The application should be able to display numbers in multiple ways, eg as decimal numbers with a radix point, or as whole numbers, possibly multiplied by irrational constants, with fractional parts.

3.1.12 Arbitrary Precision Arithmetic

The application should be able to handle numbers of arbitary size (within the system's memory capacity).

3.1.13 Graphical User Interface (GUI)

The application should be able to run as a windowed desktop application with an easy-to-use graphical interface with a display similar to a calculator, with buttons that can be clicked on to add numbers and operators to the expression, to clear or evaluate expressions, or to change the mode of display for numbers.

3.1.14 Floating Point Support

The application should be able to accept floating point numbers as input and perform all operations upon them.

3.1.15 Variable Storage

The application should be expanded to add support for user-defined variables which can hold previously calculated values and may be used in future calculations during the program's runtime. The variable names should contain only letters and underscores.

3.1.16 Graphing Capabilities

The application should be able to graph 2D functions by using a special, reserved variable, which will act as the input variable to the resulting function produced by the user.

3.1.17 Multiple GUI Styles

The user should be able to switch between several different styles of GUI from within the program.

3.1.18 Calculation History

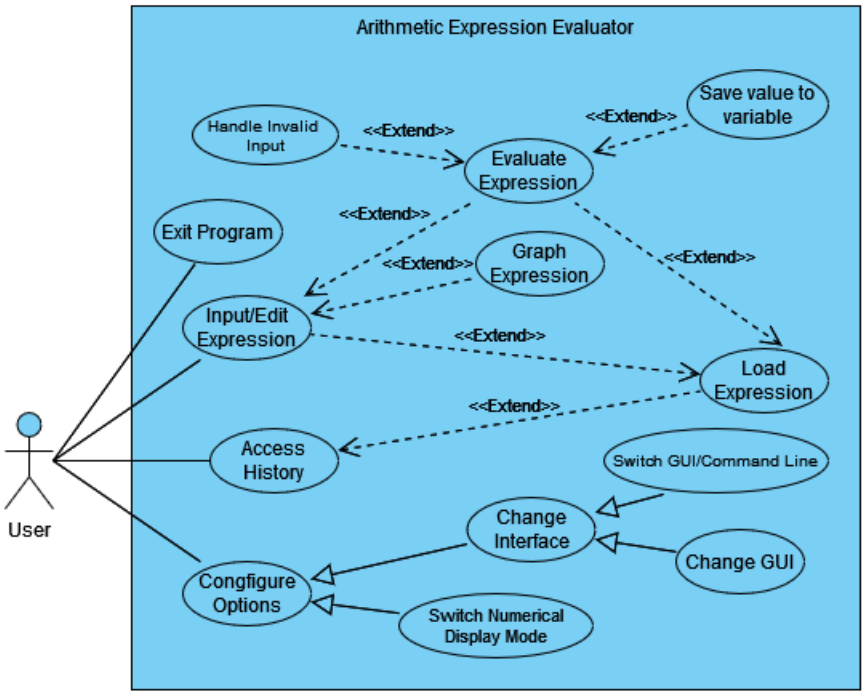
The user should be able to see a history of calculations performed since the program started running within a reasonable limit. Only expressions that succeeded in evaluation should be added to history.

3.1.19 Easter Eggs

Certain numbers (eg developer birthdays, holidays, numbers of symbolic significance) should result in unique displays or effects when they are the result of an expression.

3.2 Use-Case Specifications

Below is a use-case diagram made with Visual Paradigm capturing a high-level overview of the basic use-cases of the Arithmetic Expression Evaluator from a user-goal perspective. Below the diagram are brief descriptions of each use-case.



3.2.1 Input/Edit Expression

Scope: Application Runtime

Level: User Goal

Context: The intention of the User is to input arithmetic expressions into the program for later evaluation. Input is taken from the keyboard or from mouse clicks on buttons in the GUI.

Multiplicity: Per each instance of the application, it is most likely that there is only one user, but the user may input multiple expressions during the program's runtime.

Actor: User

Main Success Scenario:

1. User inputs/changes numbers, numerical constants, user-defined variables, or operators via keyboard input or mouse clicks on GUI elements. User may also remove previous inputs.
2. User is given visual feedback as the characters from input populate the GUI/command line.
3. User finishes entering/changing the expression.

Extensions:

3a. User evaluates the expression using the keyboard input or GUI element that takes the completed expression and returns a result.

3.2.2 Evaluate Expression

Scope: Application Runtime

Level: Sub-Function

Context: The User has input an expression and has signalled to the program to evaluate the program and return the answer.

Multiplicity: Only one (possibly comound) expression from user-input is evaluated at a time.

Actor: User

Main Success Scenario:

1. Expression is parsed from user input
2. Expression is evaluated internally.
3. The answer is displayed to the user.
4. The user may access the other base functionality of the program.

Extensions:

1a. The input was invalid. The error is handled internally.

1a.1 The error is displayed to the user; use case resumes at step 3.

2a. The contains an undefined. The error is handled internally.

2a.1 The error is displayed to the user; use case resumes at step 3.

3a. The user stores the answer as a user-defined variable to use in later expressions.

3.2.3 Access History

Scope: Application Runtime

Level: User Goal

Context: The User has most likely input and evaluated previous expressions and has clicked on the GUI element or entered the command line keyword to view a history of previous calculations.

Multiplicity: There is one record of the history of the runtime of the application which may contain multiple expressions. There is most likely only a single user per instance of the application.

Actor: User

Main Success Scenario:

1. The User enters the command line keyword or clicks on the GUI element to see history.
2. A list of the last few expressions that had been evaluated during runtime is diplayed.
3. The User exits the history and returns to the base functionality of the program.

Extensions:

2a. The user selects a previous expression.

2a.1 The user exits the history.

2a.2 The selected expression is loaded as input and may be edited or evaluated as normal input.

3.2.4 Configure Options

Scope: Application Runtime and Display

Level: Summary

Context: The User wishes to configure the program in some manner and has input a specific command to the command line interface or has navigated to the area of the GUI which allows for configuring options.

Multiplicity: A single user running an instance of the application may configure several or no options.

Actor: User

Main Success Scenario:

1. The User enters the command line keyword or navigates to the area of the GUI which allows for the configuration of options.
2. The User selects an option.
3. The User changes that option.
4. The program updates to accomodate the change.
5. The User is returned to the base functionality of the program.

Extensions:

2a. The user doesn't change the option and exits the options. The use resumes at step 5.

4a. The user selects another option and the use case returns to step 3.

3.2.5 Change Interface

Scope: Application Runtime and Display

Level: Summary

Context: The User is has already accessed the area of the program in which options can be changed and wishes to change something about the interface.

Multiplicity: A single user is changing one or more options about the interface

Actor: User

Main Success Scenario:

1. The user selects an option.
2. The User changes that option.
3. The program's interface updates to accomodate that change.
4. The User is returned to the base functionality of the program.

Extensions:

1a. The user doesn't change the option and exits the options. The use case resumes at step 4.

3a. The user selects another option and the use case returns to step 1.

3.2.6 Switch GUI/Command Line

Scope: Application Runtime and Display

Level: User Goal

Context: The User has already accessed the area of the program in which options can be changed and changes either to the command line interface from the GUI or to the GUI from the command line interface.

Multiplicity: A single user is changing to one program interface.

Actor: User

Main Success Scenario:

1. The User switches the interface in one of two ways:
    a) The User switches from the GUI to the command line interface
    b) The User switches from the command line interface to the GUI.
2. The application updates to accomodate the change.
3. The User is returned to the base functionality of the program.

Extensions:

1a. The user doesn't change the interface mode. The use case resumes at step 3.

3.2.7 Change GUI

Scope: Application Runtime and Display

Level: User Goal

Context: The User is in GUI mode already and has accessed the area of the program in which options can be changed and wishes to change the style of the GUI.

Multiplicity: A single user is selecting one GUI style from multiple GUI styles.

Actor: User

Main Success Scenario:

1. The user is presented with mutliple options for the GUI.
2. The User picks one of the GUI styles.
3. The program's interface updates to accomodate that change.
4. The User exits the GUI selection process and is returned to the base functionality of the program.

Extensions:

1a. The user doesn't change the GUI and exits the selection process. The use case resumes at step 4.

3a. The user continues to look at other GUI styles and the use case returns to step 1.

3.2.8 Switch Numerical Display Mode

Scope: Application Runtime and Display

Level: User Goal

Context: The User has navigated to/entered the options context and wishes to change how the results of expressions are displayed.

Multiplicity: A single user is selecting one style of displaying numbers from multiple.

Actor: User

Main Success Scenario:

1. The user is presented with mutliple options for how the results of expressions should be displayed, numerically.
2. The User picks one of the display modes.

3. The program updates to accomodate the change and any currently displayed result is updated to use the new display style.
4. The User returns to the base functionality of the program.

Extensions:

1a. The user doesn't change the display mode. The use case resumes at step 4.

3.2.9 Exit Program

Scope: Application Runtime

Level: User Goal

Context: The application has been launched and is currently running and the user wishes to exit the application.

Multiplicity: Per each instance of the application, it may be exited once by the current User.

Actor: User

Main Success Scenario:

1. The User navigates to and clicks the exit button on the application's window.
2. All memory used by the application is freed, any additional windows are closed, and the User exits the application.

Extensions:

1a. In the command line interface, the user may enter a specific keyword to exit the application. Use case resumes at step 2.

3.2.10 Handle Invalid Input

Scope: Application Runtime

Level: Sub-Function

Context: The User has asked the program to evaluate an expression that is eithers syntactically or mathematically invalid (eg, divide by zero).

Multiplicity: A single expression may contain multiple errors, but only the first error encountered by the parser is brought to the User's attention.

Actor: User

Main Success Scenario:

1. The error within the expression is is identified.
2. User is given feedback from the program that the input is errored.
3. The User is returned to the normal flow of the program.

3.2.11 Save Value to Variable

Scope: Application Runtime

Level: User Goal

Context: A user-defined expression has been evaluated and the result is now displayed to the user.

Multiplicity: The user may save the single output to multiple user-defined variables.

Actor: User

Main Success Scenario:

1. The User chooses the option in the GUI or enters the keyword in the command line interface to save the answer to a variable.
2. The user is prompted to give a name for the variable.
3. The value is saved to the user-defined variable, which may now be used in expressions.
4. The User is returned to the normal flow of the program.

Extensions:

2a. If user-defined variables are already present, in GUI mode the user may select the variable the user wishes to overwrite from a drop-down list. The use case resumes at step 3.

3a. If the name is a pre-defined keyword in the program, the user is informed of this and the variable is not created. The use case resumes at step 1 or 4.

3b. If the name matches a variable already defined by the user, the current value of that variable is overwritten by the new value. The use case resumes at step 4.

3.2.12 Graph Expression

Scope: Application Runtime and Display

Level: User Goal

Context: The User is in GUI mode and a user-defined expression contains the pre-defined variable used to graph expressions. The User clicks on the GUI element which is used for graphing.

Multiplicity: The user may save the single output to multiple user-defined variables.

Actor: User

Main Success Scenario:

1. The User has entered the predefined variable used for graphing into a valid expression.
2. The User clicks on the GUI element to graph the expression.
3. A new window is created showing the graph of the expression using the pre-defined graphing variable's value as the x-axis and the output of the expression as the y-axis within a fixed range.
4. The User closes the graphing window.
5. The User is returned to the normal flow of the program.

Extensions:

2a. If the expression is invalid, the User will be notified and no graph will be made. Use case resumes at step 5.

4a. The User does not close the graphing window and it remains open until the program is shut down. Use case resumes at step 5.

3.3 Supplementary Requirements

3.3.1 Made in C++

The application should be developed primarily using the C++ programming language.

3.3.2 Object-Oriented Programming

The application should be developed using object-oriented programming principles.

3.3.3 *Follows Style Guidelines*

The source code for the program should follow the Google C++ Style Guide and the Style Notes.

3.3.4 *Comments and Documentation*

The application should include comments in the source code and documentation that explains the functionality.

3.3.5 *Informative Error Messages*

The error messages output by the program at runtime should be informative enough that the user understands where in the expression the error occured.

3.3.6 *Easy to Maintain, Scalable Codebase*

The codebase of the application should be organized in such a manner that modifications and expansions to the program require minimal effort.

3.3.7 *Testable*

The application must have a suite of test cases designed for it which can thoroughly expose any defects present within that iteration.

3.3.8 *Accessible*

The end-user, and especially graders, should be able to seamlessly access and run the program without any configuration or extra steps.

3.3.9 *Portable*

The final application should have builds able to run on different operating systems. Any external libraries included should emphasize portability.

3.3.10 *Enjoyable and Intuitive User Experience*

The application should provide a high quality, intuitive user experience.

3.3.11 *Performant*

The application should perform all operations reasonably quick and efficiently. The amount of memory taken up by the application during runtime should be minimal.

4. Classification of Functional Requirements

Note: a **bolded** name indicates presence in the current stable release.

Functionality	Type
Parse and evaluate user input	Essential
Basic operations	Essential
Order of operations	Essential
Parenthetical Grouping	Essential
Usage of Numeric Constants	Essential

Error Handling	Essential
Command-Line Interface	Essential
Alternative syntax for certain operators	Essential
Floating Point Input	Desirable
Advanced operations	Desirable
Multiple Numerical Modes	Desirable
Arbitrary Precision Arithmetic	Desirable
Graphical User Interface (GUI)	Desirable
Variable Storage	Optional
Graphing Capabilities	Optional
Multiple GUI Styles	Optional
Calculation History	Optional
Easter Eggs	Optional

5. Appendices

N/A.