Group 0

Arithmetic Expression Evaluator in C++
Software Development Plan
Version <1.0>

Revision History

Date	Version	Description	Author
19/09/2024	1.0	Initial Meeting	Evan Rogerson, Mason West, Nick Heyer, Rahul nesan, Ben Haney, Samantha Adorno

Table of Contents *[keep this; say N/A when inapplicable]*

Software Development Plan

-

1. Introduction

1.1 Purpose

This project aims to create a calculator to take input from the user as a string, turn said string into an equation, and return the result. The calculator must be capable of doing addition subtraction multiplication division floor division and exponentials. The calculator should be able to handle parenthesis and order of operations and deal with constants defined earlier.

The purpose of the *Software Development Plan* is to gather all information necessary to control the project. It describes the approach to the development of the software and is the top-level plan generated and used by managers to direct the development effort.

The following people use the *Software Development Plan*:

- The **project manager** uses it to plan the project schedule and resource needs, and to track progress against the schedule.
- **Project team members** use it to understand what they need to do, when they need to do it, and what other activities they are dependent upon.

1.2 Scope

This *Software Development Plan* describes the overall plan to be used by the <project name> project, including deployment of the product. The details of the individual iterations will be described in the Iteration Plans.
The plans as outlined in this document are based upon the product requirements as defined in the *Vision Document.*

The Project must be completed by the end of the semester. Additionally, we must progressively work on the project to not fall behind and meet smaller deadlines (such as our initial plan on Sept 29). We must have a fully completed project submitted by the deadline in such a way that it doesn't have errors.

This Software Development Plan applies to the development of the Arithmetic Expression Parser project. It covers all aspects of the project, including design, implementation, testing, and error handling. Each phase of the project will be influenced by this plan, ensuring that all requirements are met, and the final product is delivered on time and operates as expected.

1.3 Definitions, Acronyms, and Abbreviations

See the Project Glossary

.References

Our vision for the Arithmetic Expression Parser project is to create a robust, efficient, and user-friendly C++ program capable of parsing and evaluating complex arithmetic expressions with accuracy and precision. The parser will not only handle standard arithmetic operators but will also correctly interpret expressions with parentheses and maintain operator precedence (PEMDAS), ensuring reliability in various scenarios.

This vision includes:

- Developing a C++ program capable of parsing and evaluating arithmetic expressions involving *+, -, , /, %, and ** operators.
- Ensuring the program handles parentheses correctly and follows the PEMDAS order of operations.
- Accommodating numeric constants
- Our vision for this project is to have a fully functional calculator. This calculator must follow the order of operations and be resistant to crashing.

- s (initially integers, with future flexibility for floating-point numbers).
- Implementing robust error handling to manage invalid expressions or operations such as division by zero.

1.4 Overview

This *Software Development Plan* contains the following information:

Project Overview — provides a description of the project's purpose, scope, and objectives. It also defines the deliverables that the project is expected to deliver.

Project Organization — describes the organizational structure of the project team.

Management Process — explains the estimated cost and schedule, defines the major phases and milestones for the project, and describes how the project will be monitored.

Applicable Plans and Guidelines — provide an overview of the software development process, including methods, tools and techniques to be followed.

.

2. Project Overview

2.1 Project Purpose, Scope, and Objectives

The purpose of this project is to develop a versatile arithmetic expression evaluator that processes mathematical expressions, parses them, and computes their results following the correct order of operation (PEMDAS). The project scope includes Input handling, parsing, Order of operations, Error handling, Output. The Objectives are Expression parsing, Algorithm design, Handling parentheses and precedence, Robustness and accuracy, Interface.

2.2 Assumptions and Constraints

Some assumptions that this plan is based on are that all team members will contribute equally and to the best of their ability. We will meet in person biweekly and communicate over discord between meetings. Some constraints include varying schedules and levels of experience with C++ coding.

2.3 Project Deliverables

The deliverables will be the project management document, the requirements document, the design specs, the implementation and the test cases. The implementation of the calculator must follow the order of operations for addition, subtraction, multiplication, division, modulus, and exponents. Our code must be formatted to have good code practices and be easy to read. The code will adequately comment and have efficient usage of memory allocation. Here are the test cases we will be testing. We know their prior results of these operations and thus can compare our calculator's results to the known values.

- 3 + 4
- 8 - (5 - 2)
- 10 * 2 / 5
- 2 ** 3
- 4 * (3 + 2) % 7 - 1
- (((2 + 3))) + (((1 + 2)))
- ((5 * 2) - ((3 / 1) + ((4 % 3))))
- (((2 ** (1 + 1)) + ((3 - 1) ** 2)) / ((4 / 2) % 3))
- (((((5 - 3))) * (((2 + 1))) + ((2 * 3))))
- ((9 + 6)) / ((3 * 1) / (((2 + 2))) - 1)

- +(-2) * (-3) – ((-4) / (+5))
- -(+1) + (+2)
- -(-(-3)) + (-4) + (+5)
- +2 ** (-3)
- -(+2) * (+3) - (-4) / (-5)
- 2 * (4 + 3 - 1
- * 5 + 2
- 4 / 0
- 5 (2 + 3)
- 7 & 3
- (((3 + 4) - 2) + (1)
- ((5 + 2) / (3 * 0))
- ((2 -) 1 + 3)
- ((4 * 2) + (-))
- ((7 * 3) ^ 2)

Deliverables for each project phase are identified in the Development Case. Deliverables are delivered towards the end of the iteration, as specified in section *4.2.4 Project Schedule*.

2.4 Evolution of the Software Development Plan

The *Software Development Plan* will be revised prior to the start of each Iteration phase.

We will start the unscheduled revision and plans as time goes on in our meeting after this initial meeting ends.

3. Project Organization

3.1 Organizational Structure

We have divided ourselves into separate roles and determined our skills from there and we will all be reviewing and helping each other as time goes on.

3.2 External Interfaces

3.3 Roles and Responsibilities *[the more details here, the easier your job; include contact info, availability info, expertise, …]*

Project Manager: Mason West

Contact info: mason.west@ku.edu

- Keeps track of the project schedule, assigns tasks, and makes sure everyone meets deadlines. They also handle any issues that come up.

Scrum Master: Ben Haney

Contact info: benhaney05@ku.edu

- Helps the team follow Agile practices, organizes daily check-ins, and plans work sessions.

Technical Lead: Nick Heyer

Contact info: nickheyer@ku.edu

- Offers technical advice and ensures the team follows good coding practices. They also help solve any technical problems.

Quality Assurance Lead: Evan Rogerson

Contact info: e890r383@ku.edu

- Makes sure the project artifacts meet quality standards. During coding, they plan and run tests to find and fix bugs.

Configuration Manager: Rahul Nesan

Contact info: rahulnesan@ku.edu

- Ensures the integrity, traceability of project configurations and code, facilitating smooth collaboration and minimizing risk in the software development lifecycle.

UX/IU Designer: Samantha Adorno

Contact info: samantha.adorno@ku.edu, sadorno1

Availability: Thursdays after 4pm, Mondays after 3pm.

- Designs the user interface if we decide to make a GUI. Creates sketches and prototypes.

Person	**Unified Process for EDUcation Role**

Anyone on the project can perform Any Role activities.

4. Management Process

4.1 Project Estimates

We plan to complete the project over the course of 6 meetings. We will also communicate over discord to ensure we are all up to date on our objects.

4.2 Project Plan

We will be meeting bi-weekly on Thursday at 4PM and having small meetings in between via zoom and discord.

4.2.1 Phase Plan

4.2.2 Iteration Objectives

For iteration 1 the goal is to plan the project and list out requirements. For iteration 2 we will discuss the architecture and design of the parser. For iteration 3 we will go through initial development and prototyping. For iteration 4 we will test our designs/prototypes and figure out any needed modifications and feedback. For iteration 5 we will go through final testing and discussion relating to the project.

4.2.3 Releases

- *Release 1: Initial Demo (October 22)*

Type: Demo

Description: This release will showcase the basic functionality of the arithmetic expression parser, including tokenization of simple arithmetic expressions and basic evaluation of operations like addition and subtraction. It will focus on demonstrating the program's core structure and tokenization logic, ensuring that the foundation is solid before further feature implementation.

- *Release 2: Beta Version (November 12)*

Type: Beta

*Description: This release will include full operator support (+, -, *, /, %, **) and parentheses handling for correct operator precedence (PEMDAS). It will be a feature-complete version of the parser with initial error handling in place, but further testing and optimizations will still be needed. Feedback from testing will be integrated into the next iteration.*

- *Release 3: Final Version (December 1)*

Type: Final Release

Description: This final release will be a fully functional version of the arithmetic expression parser, including comprehensive error handling (e.g., division by zero, invalid expressions) and performance optimizations. It will be thoroughly tested with all features working as expected. The release will be accompanied by a user manual/README, design documentation, and a final set of test cases

4.2.4 Project Schedule

Milestone	**Target Date**	**Description**
Project Management Plan	September 20	Complete the document with the roles, goals and ideas.
Requirements Document	September 30	Complete the requirements document, clearly defining what the program needs.
Design Document	October 10	Finalize the design of the system, including diagrams.
Prototype Implementation	October 20	Implement basic functionality to ensure a working foundation.
Milestone 1 Demo	October 22	Demo basic features..
Full Implementation	November 10	Complete the implementation with operator precedence, parentheses handling, and error checking.
Milestone 2 Demo	November 12	Demo the completed parser with all features and error handling.
Testing and Bug Fixing	November 20	Conduct thorough testing and debugging of the program.
Final Submission	December 1	Submit the final C++ program, README file, and all required documentation.

Final Demo	December 3	Presentation

4.2.5 Project Resourcing

We will have 6 people working on this project. All members will have prior coding experience.

4.3 Project Monitoring and Control

We will be prioritizing saving frequently and making sure to not make any decisions on our own without talking to one of our team members first. We want to minimize major mistakes and identify problems that can lead to the elimination of our code and read and rewrite as applicable.

4.4 Requirements Management

All changes to requirements will be approved by all team members and documented.

4.5 Quality Control

Defects will be recorded and tracked as Change Requests and defect metrics will be gathered (see Reporting and Measurement below).

All deliverables are required to go through the appropriate review process, as described in the Development Case. The review is required to ensure that each deliverable is of acceptable quality, using guidelines and checklists.

Any defects found during the review that are not corrected before releasing for integration must be captured as Change Requests so that they are not forgotten.

We will record each of the errors we encounter while writing the code and our solutions for how we fixed them. By keeping a detailed error log, we will ensure that if 1 person on our team encounters an error, others will know what to do if they find the same error. This will smooth the debugging process.

4.6 Reporting and Measurement

Updated schedule estimates, and metrics summary reports, will be generated at the end of each iteration.

The Minimal Set of Metrics, as described in the RUP Guidelines: Metrics will be gathered every week. These include:

Earned value for completed tasks. This is used to re-estimate the schedule and budget for the remainder of the project, and/or to identify need for scope changes.

Total defects open and closed – shown as a trend graph. This is used to help estimate the effort remaining to correct defects.

Acceptance test cases passing – shown as a trend graph. This is used to demonstrate progress to stakeholders.

We will track the results of our test cases and include them in our final report.

Refer to the Project Measurements Document (AAA-BBB-X.Y.doc) for detailed information.

4.7 Risk Management

Risks will be identified in Inception Phase using the steps identified in the RUP for Small Projects activity "Identify and Assess Risks". Project risk is evaluated at least once per iteration and documented in this table.

We will run numerous test cases to ensure that the program does not crash on the user. We will also ensure the program does not get stuck in an infinite loop as this causes excessive cpu usage.

Refer to the Risk List Document (CCC-DDD-X.Y.doc) for detailed information.

4.8 Configuration Management

Appropriate tools will be selected which provide a database of Change Requests and a controlled versioned repository of project artifacts.

All source code, test scripts, and data files are included in baselines. Documentation related to the source code is also included in the baseline, such as design documentation. All customer deliverable artifacts are included in the final baseline of the iteration, including executables.

The Change Requests are reviewed and approved by one member of the project, the Change Control Manager role.

Refer to the Configuration Management Plan (EEE-FFF-X.Y.doc) for detailed information.

Change Process:

- *Submission: Any issues or change requests are logged in Git.*
- *Review: Changes are reviewed during weekly team meetings to assess their impact and priority.*
- *Approval: Minor changes are approved quickly, while significant changes are discussed and voted on by the team.*

Naming & Versioning:

- *Code: module_function_vX.X.cpp (e.g., parser_eval_v1.0.cpp)*
- *Documentation: DocType_Project_vX.X.extension (e.g., Design_ExprParser_v1.0.pdf)*
- *Test Cases: Test_Feature_TestID.extension (e.g., Test_Parentheses_TC101.txt)*
- *Executables: Project_vX.X_OS.exe (e.g., ExprParser_v1.0_Win.exe)*

Versioning will follow semantic versioning (Major.Minor.Patch) where major changes increment the first number, minor updates increment the second, and patches increment the third.

Retention:

- *All project files will be retained in the GitHub repository for the duration of the project and archived for one year afterward.*

Backup & Recovery:

- *Backups: GitHub provides automatic version control and cloud storage for continuous backups.*
- *Recovery: In case of issues, the latest stable version will be restored from the repository within 24 hours.*

-

5. Annexes

The project will follow the UPEDU process.

Other applicable process plans are listed in the references section, including Programming Guidelines.

Programming Guidelines:

- Follow Google's C++ Style Guide for consistent, readable code.
- Use proper error handling and add comments where needed.

Design Guidelines:

- Stick to object-oriented design and modular structure.
- Use UML diagrams as per standard practices.

Testing Guidelines:

- Use Google Test (GTest) for unit tests.
- Cover all cases, especially operator precedence and parentheses.