Individual work

Due Date: Tuesday, December 12, 2023
Handover Date: Wednesday, November 1, 2023

1 Introduction

The Individual work consists of several tasks that you will need to implement by yourself, NO EXTERNAL LIBRARIES ALLOWED but stdlib and stdio(applies to all the tasks besides the hard one). There will be no base task so you have to build up your mark based only on the tasks provided below.

2 Input

Each task will have its own input.

3 Task(scroll down for the hard task section)

to be updated...

3.1 Easy

to be updated...

3.2 Medium

to be updated...

4 Grading

to be updated...

5 Reporting

You have to use latex to create/format your report.

6 Hard - Implement your own calculator DSL

In this task, you will create your own simple calculator Domain-Specific Language (DSL), similar to the example provided during our class. You have the freedom to make choices in the following aspects:

- **Syntax:** You can design a syntax of your choice, which can be as unconventional as languages like Brainfuck, as long as it adheres to the constraints specified in the constraints section.
- **Grammar:** You have the freedom to define the grammar for your DSL as you see fit.
- **External Libraries:** You are allowed to use external libraries to assist in your implementation, but they should not directly solve your problem. For example, during labs, you might have had a task to count the number of punctuations in a string. Using a function like 'ispunct()' to count the punctuations was not be considered a valid solution. Instead, a valid solution might involve using 'strlen()' to facilitate string parsing.
- **Tree Parsing Algorithm:** You are free to choose any parsing algorithm you prefer.

6.0.1 Constraints

To ensure that your DSL implementation meets specific criteria, you must adhere to the following constraints:

- YOU MUST USE C OR C++ for the implementation.
- You must implement a lexer, a parser, and an interpreter for your DSL.
- Your interpreter should parse the tree generated by the parser, not just the tokens produced by the lexer.
- Variable Declaration: Each variable declaration must specify its data type explicitly(e.g., 'var int x', 'float y = 9.8').
- Arithmetic Operations: Your DSL must support basic arithmetic operations, such as addition, subtraction, multiplication, and division.
- Control Structures: Implement at least one control structure, such as conditionals (e.g., if statements) and one loop (e.g., for or while loops) in your DSL.
- Error Handling: Your DSL should provide clear error messages for syntax and runtime errors, making it user-friendly.
- Interactive Mode: Implement an interactive mode where users can enter DSL code line by line(you can use external text file for code input).
- Modular Design: Organize your DSL implementation into separate modules or components for the lexer, parser, and interpreter to maintain a clean and maintainable codebase.
- Source Code: Your codebase should be organized into multiple files to improve clarity and ease of understanding. Consider dividing your code into separate modules or components. The entire codebase should be uploaded to a GitHub repository as you won't be able to upload multiple files on ELSE.

For a visual representation of the DSL's structure, you can refer to this diagram, I found it easy to understand yet very insightful(skip the visitor pattern under the interpreter, its too much for you)

6.0.2 Reporting

The report must be created using Latex. You will have a different report structure.

Your report must provide a clear and understandable vision of your DSL, including everything you consider will help in understanding your DSL. The clearer the report is, the fewer questions I will have. However, there are several things you must include:

- Syntax Keyword Meanings: It's a good idea to create a table with the meanings of your syntax keywords.
- Instructions on How to Write a Sample Program: Provide step-by-step instructions on how to create a sample program using your DSL, from start to finish.
- Program Examples: Include both error-free program examples and examples with errors to illustrate how to use your DSL effectively and how to handle errors.

Good luck with your DSL implementation!