Assignment 4 – Calc Report Template

Debi Majumdar

CSE 13S – Winter 24

Purpose

Audience for this section: Pretend that you are working in industry, and write this paragraph for your boss. You are answering the basic question, "What does this thing do?". This section can be short. A single paragraph is okay.

Do not just copy the assignment PDF to complete this section, use your own words.

The RPN calculator project aims to develop a software application that enables users to perform arithmetic operations and evaluate trigonometric functions in Reverse Polish Notation (RPN). This calculator utilizes a stack-based approach for expression evaluation, offering an efficient and intuitive way for users to input and compute mathematical expressions. By implementing mathematical function approximations and error handling mechanisms, the calculator provides accurate results while adhering to industry standards.

Questions

Please answer the following questions before you start coding. They will help guide you through the assignment. To make the grader's life easier, please do not remove the questions, and simply put your answers below the text of each question.

- Are there any cases where our sin or cosine formulae can't be used? How can we avoid this?

  Yes, our sine or cosine formulae may not produce accurate results for very large or very small input values due to limitations in floating-point arithmetic and the approximation methods used. To avoid this, we can limit the range of input values or implement additional error checking and handling mechanisms to detect and handle out-of-range inputs gracefully.

- What ways (other than changing epsilon) can we use to make our results more accurate?

  Other ways to improve the accuracy of our results include increasing the number of terms in the Taylor series expansion, using higher precision arithmetic (e.g., double-double arithmetic), refining the algorithm used for approximation, and optimizing the handling of special cases and edge conditions. [1]

- What does it mean to normalize input? What would happen if you didn't?

  Normalizing input means transforming it into a standard form or range to ensure consistency and facilitate processing. In the context of trigonometric functions, normalizing input typically involves adjusting angles to a specified range (e.g., 0 to 2pi for radians) to avoid redundancy and simplify calculations. If input is not normalized, it may lead to inconsistencies in results, erroneous calculations, and difficulty in interpreting the output.

- How would you handle the expression 321+? What should the output be? How can we make this a more understandable RPN expression?

  To handle the expression 321+, we would interpret it in Reverse Polish Notation (RPN) by pushing the numbers 3, 2 onto the stack, and then applying the addition operator, resulting in the output of 5. To make this expression more understandable in RPN, we could write it as 3 2 +, where the numbers

[1] hint: Use trig identities

are entered first, followed by the operator, which aligns with the stack-based evaluation approach of RPN calculators.

- Does RPN need parenthesis? Why or why not?

  No, RPN does not require parentheses because it eliminates the need for explicit grouping of operations by its inherent nature. In RPN, the order of operations is determined solely by the sequence in which operands and operators are entered, removing ambiguity and eliminating the need for parentheses. Each operator in RPN acts on the most recent operands pushed onto the stack, making the expression unambiguous and eliminating the need for parentheses to specify precedence.

Testing

List what you will do to test your code. Make sure this is comprehensive. [2] Be sure to test inputs with delays and a wide range of files/characters.

To ensure comprehensive testing of the code, I will conduct unit testing for each function, covering normal, edge, and error cases. Integration testing will verify module interaction, while CLI testing will ensure correct command-line behavior and error handling. Error handling testing will focus on scenarios like invalid inputs and stack overflows, while performance testing will assess speed and memory usage with large inputs. Edge cases will be tested for extreme input values and complex expressions, and file input testing will verify reading from files. Character input testing will cover various character inputs, including special characters and whitespace. If applicable, concurrency testing will assess behavior under concurrent execution, and regression testing will be performed after code changes to maintain existing functionality. This approach ensures the reliability and correctness of the calculator program across diverse scenarios.

How to Use the Program

Audience: Write this section for the user of your program. You are answering the basic question, "How do I use this thing?". Don't copy the assignment exactly; explain this in your own words. This section will be longer for a more complicated program and shorter for a less complicated program. You should show how to compile and run your program. You should also describe any optional flags or inputs that your program uses, and what they do.

To show "code font" text within a paragraph, you can use `\lstinline{}`, which will look like this: `text`.

For a code block, use `\begin{lstlisting}` and `\end{lstlisting}`, which will look like this:

```
Here is some code in lstlisting.
```

And if you want a box around the code text, then use `\begin{lstlisting}[frame=single]` and `\end{lstlisting}`

which will look like this:

```
Here is some framed code (lstlisting) text.
```

Want to make a footnote? Here's how.[3]

Do you need to cite a reference? You do that by putting the reference in the file `bibtex.bib`, and then you cite your reference like this[1][2][3].

The program is a Reverse Polish Notation (RPN) calculator designed to perform basic arithmetic and trigonometric calculations. To use the program, first, compile it using a C compiler such as GCC. For example, you can compile the program by running `gcc -o calculator calculator.c`. Once compiled, you can run the program by executing the generated executable file, typically named `calculator`. Optionally, you can use command-line flags to enable certain features. For instance, using the `-m` flag will utilize trigonometric functions from the math library, while the `-h` flag will display the help message, providing guidance on using the program.

[2] This question is a whole lot more vague than it has been the last few assignments. Continue to answer it with the same level of detail and thought.

[3] This is my footnote.

When the program is running, you'll be prompted with a `>` character, indicating that the calculator is ready to accept input. Enter expressions in RPN format, where numeric values are entered first, followed by operators. For example, to calculate $2 + 3$, you would enter `2 3 +`. Press Enter after each expression to compute the result. You can chain multiple expressions together on a single line, separating each with a space.

For trigonometric functions like sine, cosine, and tangent, simply input the angle in radians after the function name. For example, to calculate the sine of 1 radian, enter `1 sin`. If you encounter any errors during input or calculation, the program will display an appropriate error message to guide you.

It's important to note that the program employs trigonometric identities and approximation techniques to compute trigonometric functions, ensuring accurate results within the specified constraints. Additionally, you can refer to trigonometric identities for more complex calculations, leveraging the program's functionality to simplify expressions.

Program Design

Audience: Write this section for someone who will maintain your program. In industry you maintain your own programs, and so your audience could be future you! List the main data structures and the main algorithms. You are answering the basic question, "How is this thing organized so that I can have a chance of fixing it?". This section will be longer for a more complicated program and shorter for a less complicated program.

In our RPN calculator program, the main way we organize information is through a stack, which is like a virtual pile of numbers. We use this stack to keep track of the numbers we're working with and the results of calculations. Think of it as a scratch pad where we jot down numbers and perform operations. When it comes to actually crunching the numbers, we've got algorithms in place to understand and process the expressions you give us. These algorithms handle everything from reading what you type to figuring out what math needs to be done and then actually doing it. We also have checks in place to catch any mistakes you might make, like typing something that doesn't make sense or trying to do something impossible.

Pseudocode

Give the reader a top down description of your code! How will you break it down? What features will your code have? How will you implement each function.

Give the reader a top-down description of your code! How will you break it down? What features will your code have? How will you implement each function.

1. **Main Functionality:**
    - The main function will handle the core functionality of the RPN calculator.
    - It will initialize the stack and continuously prompt the user for input until they decide to exit.
    - For each input expression, it will parse the RPN expression, evaluate it, and display the result.
2. **Parsing Input:**
    - A function will be implemented to parse the user input, breaking it down into tokens (numbers, operators, trigonometric functions).
    - It will handle error cases such as invalid input or unsupported operations.
3. **Stack Operations:**
    - Functions will be created to handle stack operations like push, pop, and peek.
    - These functions will be used to manage the stack during expression evaluation.
4. **Expression Evaluation:**
    - An algorithm will be developed to evaluate the RPN expression.

- It will iterate through the tokens, performing calculations based on the operators and operands encountered.
- Special care will be taken to handle trigonometric functions and their respective calculations.

5. **Error Handling:**
   - Error handling mechanisms will be implemented to catch and handle errors gracefully.
   - Messages will be displayed to guide the user on how to correct their input.
6. **Trigonometric Functions:**
   - Specific functions will be developed to handle trigonometric calculations like sine, cosine, and tangent.
   - These functions will utilize appropriate mathematical libraries or custom algorithms to compute accurate results.
7. **Testing:**
   - Comprehensive testing procedures will be devised to ensure the correctness and robustness of the code.
   - Various test cases will be executed, including edge cases, to validate the behavior of the calculator under different scenarios.

Overall, the code will be structured in a modular and organized manner, with each function responsible for a specific aspect of the RPN calculator's functionality. This approach will make the code easier to understand, maintain, and debug.

Function Descriptions

For each function in your program, you will need to explain your thought process. This means doing the following

- The inputs of every function (even if it's not a parameter)
- The outputs of every function (even if it's not the return value)
- The purpose of each function, a brief description about a sentence long.
- For more complicated functions, include pseudocode that describes how the function works
- For more complicated functions, also include a description of your decision making process; why you chose to use any data structures or control flows that you did.

Do not simply use your code to describe this. This section should be readable to a person with little to no code knowledge. **DO NOT JUST PUT THE FUNCTION SIGNATURES HERE. MORE EXPLANATION IS REQUIRED.**

evaluateExpression(inputExpression): This function processes and evaluates RPN expressions. It splits the input expression into tokens and processes each token, applying operators or pushing numeric values onto the stack. It ensures that the expression is well-formed and provides the result.

applyBinaryOperator(op): This function applies binary operators such as addition, subtraction, multiplication, and division. It ensures there are enough operands on the stack for the operation.

applyUnaryOperator(op): This function applies unary operators such as sine, cosine, tangent, and square root. It checks for a valid number of operands on the stack.

parseDouble(s, d): This function attempts to parse a double from the input string 's'. If successful, it stores the parsed value in 'd' and returns true. If parsing fails, it returns false, indicating an invalid token.

stackPush(item): This function pushes a double item onto the stack. It returns true if the operation succeeds, indicating a successful push. If the stack is full, it returns false.

stackPeek(item): This function retrieves the top item from the stack without removing it. If the stack is not empty, it stores the top item in 'item' and returns true. If the stack is empty, it returns false.

stackPop(item): This function pops the top item from the stack. If the stack is not empty, it stores the popped item in 'item' and returns true. If the stack is empty, it returns false.

stackClear(): This function resets the stack, clearing all items and setting the stack size to zero.

These functions collectively handle various aspects of RPN expression evaluation, including the processing of numeric values, binary operators, unary operators, and stack management.

Results

Follow the instructions on the pdf to do this. In overleaf, you can drag an image straight into your source code to upload it. You can also look at `https://www.overleaf.com/learn/latex/Inserting_Images`

References

[1] Wikipedia contributors. C (programming language) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/C_(programming_language), 2023. [Online; accessed 20-April-2023].

[2] Robert Mecklenburg. *Managing Projects with GNU Make, 3rd ed.* O'Reilly, Cambridge, Mass., 2005.

[3] Walter R. Tschinkel. Just scoring points. *The Chronicle of Higher Education*, 53(32):B13, 2007.