CS/CE 1337 – PROJECT 4 – Physics Engine Calculator

Pseudocode Due: 11/4 by 11:59 PM

Project Due: 11/24 by 11:59 PM

KEY ITEMS: Key items are marked in red. Failure to include or complete key items will incur additional deductions as noted beside the item.

Submission and Grading:

- The pseudocode will be submitted in eLearning as a Word or PDF document and is not accepted late.
- All project source code will be submitted in zyLabs.
  - Projects submitted after the due date are subject to the late penalties described in the syllabus.
- Programs must compile using gcc 7.3.0 or higher with the following flags enabled
  - -Wall
  - -Wextra
  - -Wuninitialized
  - -pedantic-errors
  - -Wconversion
- Each submitted program will be graded with the rubric provided in eLearning as well as a set of test cases. These test cases will be posted in eLearning after the due date.
  - zyLabs will provide you with an opportunity to see how well your project works against the test cases. Although you cannot see the actual test cases, a description will be provided for each test case that should help you understand where your program fails.
- **Type your name and netID in the comments at the top of all files submitted. (- 5 points)**

Objective:

- Create multiple classes that interact with each other
- Implement a linked list with classes.
- Implement overloaded operators in a class

Problem: Gearbox Software in Frisco is working on a new physics engine to use with future games. As a recently hired intern, your job is to build a calculator to use in the engine. The calculator must be able to handle multiple operations and preserve the order of operations. To evaluate the expressions, you must convert the expression from infix to postfix notation and then evaluate. Both the conversion and the evaluation require the use of a stack.

Pseudocode: Your pseudocode should describe the following items

- Main.cpp
  - Detail the step-by-step logic of the main function
  - List other functions you plan to create with the main function
    - Determine the parameters
    - Determine the return type
    - Detail the step-by-step logic that the function will perform

- Do not include any pseudocode for the classes. The classes are tools to help you solve the problem. Discuss how you will use the objects created from the classes within the functions above.

File Submission

- Submit 5 files
    - main.cpp
    - node.h
    - node.cpp
    - stack.h
    - stack.cpp
- You may design the classes as you see fit.
    - If your header files contain all inline functions, you must submit a .cpp file for ZyLabs.
        - In this case, the .cpp file will be an empty file.

Class Details

- Node class – created as a homework assignment
    - Attributes
        - Operand
        - Operator
            - Consider using a union for the operand and operator to save space
        - Next pointer
    - Methods
        - Default constructor
        - Overloaded constructor
        - Accessors
        - Mutators
- Stack – created as a homework assignment
    - Attributes
        - Head pointer
    - Default constructor
    - Overloaded constructor
        - Takes node and assigns head to point at the node passed in
    - Copy constructor
        - Given a stack object, it creates a copy of the contents
    - Destructor
        - Recursively delete linked list (-5 points if not recursive)
    - Accessor
    - Mutator
    - Overloaded << operator for output
        - Print the contents of the stack
        - The stack will contain all operators or all operands
        - You must decide what the stack contains
        - The output should be the data stored in each node separated by a comma and a space
            - `1, 2, 3 or +, -, *`

    - Overloaded >> operator
        - Not used for input
        - Used as pop
        - Remove node from head of list
        - Contains 1 node pointer parameter by reference
        - Disconnect head node from list and store in parameter
    - Overloaded << operator
        - Not used for output
        - Used as push
        - Add node to head of list
        - Contains 1 constant node pointer parameter by reference
        - Adds the node to the head of the linked list

Postfix Notation:

- Each expression will be given in infix notation
- You will convert to postfix notation for evaluation
- Example
    - Infix notation - A + B / C * D
    - Postfix notation - A B C / D * +
- You may devise your own solution to store the postfix expression
    - Keep in mind that operands are not limited to 1 digit
- Postfix notation preserves the order of operations
- Requires a stack
    - Implement as a linked list (-10 points if no linked list used)
        - Must use the linked list classes listed above (-10 points if not)
    - All interaction happens at head of linked list
- Possible operators:
    - ^ (exponentiation)
    - * (multiplication)
    - / (division)
    - + (addition)
    - – (subtraction)
- Creating postfix expression
    - Add a space to the expression after each operand and operator
    - When an operand is encountered, append it to the postfix expression
        - Operands are never put onto the stack
    - When (is encountered, push it onto the stack
    - When an operator is encountered
        - Pop the stack (appending to postfix expression) until an operator of lower precedence is on top of the stack or the stack is empty

        - Precedence levels:
            - ^ has highest precedence
            - * and / have middle precedence
            - + and - have lowest precedence
    - Push the operator
  - When) is encountered, pop the stack (appending each element popped to postfix expression) until (is encountered in stack
    - Pop (but don't add to postfix array
  - When end of input is reached, pop the stack (appending to postfix expression) until stack is empty

Evaluating Postfix Expressions:

- Using the postfix notation, calculate the value of the expression
- Requires a stack
  - Implement as a linked list (-10 points if no linked list used)
    - Must use the linked list classes listed above (-10 points if not)
  - All interaction happens at head of linked list
- Read each piece of postfix expression
- If operand, push onto stack
- If operator, pop top 2 operands off stack and perform calculation
  - Push result onto stack
  - No operators are pushed onto stack
  - Be careful of the order
    - Only addition and multiplication can be applied to the two operands in any order
- When end of expression encountered, value is on top of stack
  - Pop the stack to retrieve the value

Input:

- All expressions will be read from a file.
  - Prompt the user to enter the filename. This would normally be hardcoded in an application, but zyLabs requires a filename for multiple test files.
    - This is the first thing the program will do.
- There is no limit to number of expressions in the file
- Operands can be integer or floating-point values
- There will be no spaces in the expressions
- There is no input validation required for the expressions

Output:

- All output will be written to the console
- For each expression, output with the following format
  - `<postfix expression><tab><value><newline>`
- The calculated values will be rounded to 3 decimal places