canvas.pasadena.edu

Test 1

Test 1
"The Nerds Calculator"

Introduction:

The goal of this test is for you to develop a calculator that will accept an arbitrary length algebraic expression that contains the standard operations (+,-,/, *, and ^ (raise to a power)) and parentheses. The calculator must be able to accept real numbers (numbers containing decimals), rational numbers (numbers containing a fraction, which includes mixed numbers), or integers. Once the user presses the return key, the calculator will display the RPN translation of the algebraic expression, a space, an equal sign, and then the mixed number result.

When the program starts, a prompt such as INPUT: or EXPRESSION: must be printed. After each successful calculation, this prompt will re-appear asking for the next algebraic expression to enter. Naturally, if the user enters just the return key, then the program will terminate.

When the program starts, an appropriate message should appear that describes the functionality of the program.

What is RPN?

Reverse Polish Notation is a way of expressing arithmetic expressions that avoids the use of brackets to define priorities for evaluation of operators. In ordinary notation, one might write

(3 + 5) * (7 - 2)

and the brackets tell us that we have to add 3 to 5, and then subtract 2 from 7, and multiply the two results together. In RPN, the numbers and operators are listed one after another, and an operator always acts on the most recent numbers in the list. The numbers can be thought of as forming a stack, like a pile of plates. The most recent number goes on the top of the stack. An operator takes the appropriate number of arguments from the top of the stack and replaces them by the result of the operation.

In this notation the above expression would be

3 5 + 7 2 - *

Reading from left to right, this is interpreted as follows:

- Push 3 onto the stack.
- Push 5 onto the stack. The stack now contains (3, 5).
- Apply the + operation: take the top two numbers off the stack, add them together, and put the result back on the stack. The

stack now contains just the number 8.

- Push 7 onto the stack.
- Push 2 onto the stack. It now contains (8, 7, and 2).
- Apply the - operation: take the top two numbers off the stack, subtract the top one from the one below, and put the result back on the stack. The stack now contains (8, 5).
- Apply the * operation: take the top two numbers off the stack, multiply them together, and put the result back on the stack. The stack now contains just the number 40

Polish Notation was devised by the Polish philosopher and mathematician Jan Lucasiewicz (1878-1956) for use in symbolic logic. In his notation, the operators preceded their arguments, so that the expression above would be written as

* + 3 5 - 7 2

the 'reversed' form has however been found more convenient from a computational point of view. Hewlett-Packard championed reverse Polish Notation for years, though it is rumored that their most recent calculators do not support it anymore, which is a pity. RPN is one of those things that can be difficult to learn, but wonderful to use once it is understood.

Algorithm to Convert Algebraic to RPN:

Edsger Dijkstra (Links to an external site.) (Links to an external site.), one of the pilars of Computer Science, invented the Shunting-yard algorithm (Links to an external site.) (Links to an external site.) to convert infix expressions to postfix (RPN), so named because its operation resembles that of a railroad shunting yard. The link above is crucial to solving this problem.

In order to implement the Shunting-yard algorithm, you will be needing stacks and queues.

The Assignment Particulars:

Design a C++ program that will implement the following specifications for our command line calculator. You and a single teammate may use any resource that you wish to complete this assignment, ***excepting seeking help from any other individual***.

Clarification of the requirements may be asked during lab or during office hours.

Specifications:

- The calculator must read a complete algebraic expression that can consist of integers, decimals, fractions, or mixed numbers and present the result in proper form (fractions reduced, integer results must have just the integer and no 0/1, purely fractional results must not have a leading integer 0, negative signs must be in the proper places within mixed numbers, and so forth) Any expression that contains both decimals and fractions will provide

the result in fractional / mixed number form.

- Your program must have the following classes:
  - Mixed, which will inherit from a class called Fraction, that allows for Mixed numbers (eg. 3 1/2)
  - Fraction, because you will be reading fractions on their own.
  - Parser, a class that "figures out" what the user entered and prepares three things: an operand stack, an operator stack, and a queue which holds the expression that needs to be translated from algebraic to RPN. (NOTE: This implies that the queue must hold different data types -- operands, parens, and operators)
  - A queue class which accepts tokens from the parser. Tokens can be either a mixed number, a fraction, an operator, or a parenthesis. (You cannot "cheat" by having a queue of C++ strings. The queue must "legitimately" hold tokens of different types. Hint: pointers to type void)
  - Two instances of the stack class. One which holds operators, the other holds mixed numbers and fractions.
- You ***MAY NOT*** use the STL stack or queue for this, as in you are being tested on your ability to write their equivalents. You must write the appropriate stacks and queues. The stacks and queues must be as generic as possible; meaning that I should be able to use any other students stack or queue in your program and have it work correctly.

- The stack and queue classes must implement their processes using linked-lists. Additionally, the stack and queue must not be adjusted between each user input line. This means that your parser class must check for exceptions and handle them accordingly.

- The calculator should provide error messages for "algebraically incorrect" entries (such as a missing closing parentheses, fractions that have a negative sign in the denominator, and so forth)

- ***The use of switch statements and "ladder-ifs" is prohibited***

- The calculator must have 26 “memories” named A to Z such that the user can “store” the result of an expression for later use.

    - To store the result of an expression into a memory location, the user would enter the name of the memory location followed by a space, then the equal sign, then another space followed by the expression.

        - When computation is completed, the program will output as above, with “ is now stored in memory location <M>”

    - To clear a particular memory location, the user would enter CLEAR <N> where N is the memory location

    - To clear ALL memory locations, the user will just enter CLEAR

- To use the value stored in a memory location, the user will just enter the name of the memory location instead of a numeric value. Example, supposed 3.5 was stored in memory A, then the user could write A^3 -3 as an expression. On output, the value at the memory location will be displayed.
    - Note: Users may write A = B^2 – C as a valid input!!!!
- Memories can only store ***MIXED NUMBERS!***

- For ease of parsing, the following rules must be followed:
    - All operators must have at least one space on either side of them (excepting the unary minus)
    - Mixed numbers can have only one space between the integer and the fraction
    - Unary minus signs must appear directly beside the value that they are making negative
    - Visual display of the calculator should be aesthetically pleasing.
    - The calculator exits when the user types EXIT or QUIT. Upon exiting, the program must ask the end user if they wish to store the contents of memory to a file unless the statement prior to EXIT or QUIT is WRITE <filename>. When this is done, the EXIT or QUIT command will write automatically to this file name. Conversely, if the user enters EXIT <filename> or QUIT <filename> the program will write the contents of

memory to a file before terminating the program.

- READ or LOAD to re-load stored memory locations.

- You may use any resource you wish, as long as it can be ***FULLY DOCUMENTED*** within your code.

- You must be prepared to answer any question about your program when you demonstrate its functionality during lab.

Hints:

- Read the input character by character into a queue.
  - If the character is a "(" or ")" this will dictate what to do with the input according to the Shunting-yard algorithm

- Do not wait to start on this program. ***This is a standard problem for this level of a course at the four-year universities***. At this level, ***this program should take no more than 8 hours to complete*** if you have been keeping up with the homework.

Extra Credit:

Using the Qt graphics libraries, develop a windowing application that "look like" a calculator. "Look like" means that there will be keys for digits, operators, the fraction symbol, and an equal sign (which will signify end of input). As each key is pressed, the user should see their expression being built on the calculator (as in real life). Once the equal sign is pressed, the converted expression will

appear on another display with the answer. (Note: calculators have a CLEAR and CLEAR ENTRY key, and they are required too as well as 26 memory locations!).