CMPSC 122.1
March 29, 2018

# Homework 4

Due: Refer to last page for due dates

---

<u>Overview</u>

This assignment is comprised of two unrelated programs. The first program is an exercise in utilizing stacks, and the second program will be able utilizing queues.

*Stacks and postfix notation (or Reverse Polish Notation)*

In class, we have discussed how to perform postorder tree traversal, as well as briefly looked at evaluating expression trees. Postfix notation is a related method of representing mathematical expressions. Postfix notation is also called Reverse Polish Notation (RPN).

Review this resource as well as the Lecture Notes on Queues and Stacks (last portion of the notes) for an in depth overview of RPN and strategies for completing this program.

*Priority Queue*

In computer science, a priority queue is an ADT which is similar to a regular queue; however, each entry in the queue is also has a "priority" associated with it. If two elements have the same priority, they are served according to their order in the queue.

```
PriorityQueue<string> *pq = new PriorityQueue<string>();
pq->enqueue("a", 30);
pq->enqueue("b", 10);
pq->enqueue("c", 20);
pq->enqueue("d", 10);
cout << pq->dequeue() << pq->dequeue() << pq->dequeue() << pq->dequeue();
```

The string "acbd" is outputted is the "a" has the highest priority, followed by "c", followed by "b", and then followed by "d".

Program 1. (50 points) **Reverse Polish Notation Calculator**

Write a program (*rpn.cpp)* that does the following:

- Leverages the provided Stack class (in *stack.h*) accomplish the other goals of the assignment
- Accepts a valid infix expression, converts it into RPN, and then evaluates the expression
- If an invalid infix expression is provided, write an appropriate message to standard out and end the program
- Print the convert postfix expression before evaluation, as well as its evaluated result
- The major pieces of the program are broken into reusable functions
- Infix expressions will contain the following operators: ( ) * / + -
- When reading an infix expression, be sure to consider order of operators when converting to postfix
- You may add new functions to the List and Stack classes in *stack.h.* if you feel they are necessary, but ***do not*** modify any existing functions
- If you are having issues parsing input, you can create arrays with predetermined input to test the conversion process

# Homework 4

Examples of infix and postfix expressions:

| **Infix** | **Postfix** |
|---|---|
| 3 + 4 | 3 4 + |
| 2 + 3 * 4 | 2 3 4 * + |
| ((15 ÷ (7 − (1 + 1))) × 3) − (2 + (1 + 1)) | 15 7 1 1 + − ÷ 3 × 2 1 1 + + − |

Input/Output Format:

I recommend capturing the entire infix expression into one string, and then parse its contents with the approach of your choosing. The program only needs to get one line of input from the user each time it is run.

```
Enter an infix expression: 3 + 4

The equivalent postfix expression of 3 + 4 is:
      3 4 +

The result of 3 4 + is: 7
```

Program 2. (50 points) **Priority Queue**

For this assignment you are to implement a class named **PriorityQueue<T>**, with the same methods Queue discussed in class. The only difference is that the enqueue method now takes two arguments: an element to enqueue (of generic/templated type T), and an integer representing the element's priority. Feel free to implement the class using either a linked list or an array for its underlying type.

Write a program (*pq.h*) that contains:

**1.** Contains the PriorityQueue class as specified in the above paragraph

**2.** As well as any other classes need to implement the PriorityQueue class (such as ListNode and List).

Write a program (*pq.cpp*) that:

**1.** Includes the *pq.h* file

**2.** Create an interactive "command" driven prompt system that parses "commands" from the user that are used to invoke the appropriate functions:
- Prompt the user to specify what data type they want the priority queue to contain (like for the generic vector program in Assignment 2)
- Enter a looping prompt that parses the following commands with the appropriate parameters:
    - *enqueue* <value> <priority>
    - *dequeue*
        - Print the returned value to standard out
    - *first*

  - Print the returned value to standard out
- *empty*
  - Print the returned value to standard out
- *clear*
- *quit*
  - Break out of loop, exit program normally.

Expected prompt/input with example inputs:

```
Specify what data type to store in priority queue:
      1) int
      2) float
      3) double
      4) string
      5) bool
> 1

Now accepting commands (quit to exit program):
> enqueue 1 10
> enqueue 2 5
> enqueue 3 15
> enqueue 4 10
> empty
0
> first
3
> dequeue
3
> dequeue
1
> dequeue
4
> dequeue
2
> empty
1
> quit

Exiting Program.
```

```
Note: You must use the above format for accepting the commands. Points will be
taken off for deviating from this format.
```

Template Program Files

There are no template files for this assignment. The include for *stack.h* will be published on Canvas, and will be available for copying (cp) on the Sunlab computers from the following directory:

`/home/cmpsc122/s18/hw4/`

Reference files for the two queue implementations will be available on Canvas.

# Homework 4

Compiling the Program with g++

Use the following command to compile your classes. This is the command I personally use on the Sunlab machines to compile and grade your programs:

```
g++ -Wall -o <output_name> <program_name.cpp>
```

Example:

```
g++ -Wall -o rpn rpn.cpp
```

Remember: Your programs must successfully compile without any errors, or a zero will be given for that program.

Following Instructions

You are expected to follow the directives laid out in this assignment and the course syllabus. Points will be deducted for incorrectly named files, missing/incorrectly filed out banner comments, not supplying hardcopy submissions in a pocket folder with the appropriate information, and failing to implement features/functionality specified in the assignment. It is expected that you will utilize the provided template files for this assignment, and that you do not modify the names of class members/functions that are provided in the template.

If you have questions about any portions of the assignment or what is expected, contact me for clarification.

Submission

- Electronic Submission (Due: One minute before midnight, 11:59 PM, Sunday, April 15, 2018)

  - Your four (4) source code files (*stack.h*, *rpn.cpp*, *pq.h*, and *pq.cpp*)
  - Submitted using the mail122 command on the Sunlab machines

- Hardcopy Submission (Due: Beginning of class, Monday, April 16, 2018)

  - Printed hardcopies of the four (4) source files
  - The pages of each program should be stapled together
  - Submitted in a pocket folder with your name, the course, and the section number on the front