COP 4530: Summer 2023
Programming Assignment 1 - Lists, Stacks, and Queues)

Total Points: 150
Due: Friday 07/14/2023 11:59:00 PM through Linprog

1 Objective

By completing this assignment you will be able to

- Work with the C++ STL Classes.
- Use iterators intead of positional indices to move among elements in a linear data structure.
- Solve simple abstractions to practical problems using C++ STL classes for linear data structures.
- Understand the STL API for Vectors, Stacks, and Queues.
- Analyze the time complexity of your solutions.

2 Problem 1 - Space Rocks (Vectors) - 50 points

You are simulating the interactions of bodies in space in an alternate dimension. The rocks in this dimension are made of a strange adherent material that clumps together on contact. As a consequence, a smaller rock that collides with a larger rock will be absorbed into the larger rock. If 2 rocks of equal size collide, the will destroy each other. However, we are only concerned about the rocks in a certain asteroid field heading toward one particular rock. termed *Rock0*. A rock smaller than *Rock0* will be absorbed into *Rock0* upon collision. If a larger rock collides with *Rock0*, then *Rock0* gets absorbed into the colliding rock and ceases to exist.

Write a C++ program to simulate the interaction of the rocks in the asteroid field and determine whether *Rock0* will survive the asteroid field or not. Please make sure your solution conforms to the following requirements.

1. Call this program `space.cpp`
2. You may only use the `iostream` and `string` libraries and the STL `vector` class. You may use the standard namespace.
3. Create a Class called Rock with 2 members - a string and a double. The string represents the name of the rock and the double represents the size/weight of the rock. The class would only need a default constructor, a parameterized constructor, and the getter and setter functions. Overloading the insertion and extraction operators is not required but would make your life easier.
4. Create a comparator class. Overload the () operator to create a comparator that will return `true` if the first Rock parameter object were "smaller" than the second Rock parameter object. The "smaller" Rock object is the one with the lower size. If two Rocks were the same size, then use lexicographic order on the name of the rock.

5. Write a function to use insertion sort (given below) that accepts a vector of Rocks and the comparator to sort the values in the vector in ascending order.
6. Write a function called *insert* to accept the sorted vector, another Rock object, and a Comparator class object. Use iterators and the vector's insert function to insert the second parameter in its proper place in the sorted vector.
7. In the main function, create an Object for *Rock0* and a vector of Rocks for the rest of the rocks in the field.
8. Read in the initial values in the asteroid field.
9. sort the vector.
10. Simulate the interaction of rocks using the sorted vector. While simulating, the user will enter one of 3 values - 'A' to add another object in the field, 'P' to progress the simulation by 1 collision, or 'C' to indicate they are done introducing new objects and the simulation may be concluded.
11. The Simulation will end if *Rock0* ceases to exist or if *Rock0* has absorbed all the rocks in the asteroid field.
12. Rocks may only be added to the asteroid field if there is at least one other rock than *Rock0* left.
13. At the end of the simulation, print "Yes" if *Rock0* will survive the simulation and "No" if *Rock0* will be absorbed in the course of the simulation and cease to exist.
14. Analyze the time complexity of the insertion sort and include that in the `analysis.pdf` file.
15. Analyze the time complexity of the insert function and include that in the `analysis.pdf` file.
16. **General Requirements**
    (a) Use the built-in functions of the Vector class when possible
    (b) Conform to standard software engineering practices (data hiding, function/code reuse, const keyword used where required, etc.)

Algorithm for Insertion Sort

This is pseudocode for insertion sort. Please use the C++ Vector STL class API to perform the required operations.

```
loop i from 1 to Length(A)
        x = A[i]
        j = i - 1
        loop as long as j >= 0 and A[j] > x
                A[j+1] = A[j]
                j = j - 1
        end loop
        A[j+1] = x
end loop
```

Sample Run 1

```
Enter the values for Rock 0: rock0 5
Enter the number of other rocks in the asteroid field: 5
Enter the values for the other rocks: rock1 20
rock2 3
rock3 3
rock4 10
rock5 6

P - Progress the simulation
A - Add another rock
C - Complete the simulation
Enter your choice: P
rock2:3 was absorbed

P - Progress the simulation
A - Add another rock
C - Complete the simulation
Enter your choice: P
rock3:3 was absorbed

P - Progress the simulation
A - Add another rock
C - Complete the simulation
Enter your choice: A
Enter the values for the new rock: rock6 1.5

P - Progress the simulation
A - Add another rock
C - Complete the simulation
Enter your choice: P
rock6:1.5 was absorbed

P - Progress the simulation
A - Add another rock
C - Complete the simulation
Enter your choice: A
Enter the values for the new rock: rock10 25

P - Progress the simulation
A - Add another rock
C - Complete the simulation
Enter your choice: P
rock5:6 was absorbed

P - Progress the simulation
A - Add another rock
C - Complete the simulation
Enter your choice: C
rock4:10 was absorbed
rock1:20 was absorbed
rock10:25 was absorbed
```

```
rock0:73.5 has survived
```

Sample Run 2

```
Enter the values for Rock 0: rock0 5
Enter the number of other rocks in the asteroid field: 3
Enter the values for the other rocks: rock1 2
rock2 1
rock3 2
P - Progress the simulation
A - Add another rock
C - Complete the simulation
Enter your choice: P
rock2:1 was absorbed

P - Progress the simulation
A - Add another rock
C - Complete the simulation
Enter your choice: A
Enter the values for the new rock: rock4 10

P - Progress the simulation
A - Add another rock
C - Complete the simulation
Enter your choice: A
Enter the values for the new rock: rock5 15

P - Progress the simulation
A - Add another rock
C - Complete the simulation
Enter your choice: C
rock1:2 was absorbed
rock3:2 was absorbed
rock0:10 was destroyed by colliding with an equal-sized object

rock0:10 did not survive
```

3 Problem 2 - PostFix Expressions (Stacks) - 50 points

A PostFix expression is a mathematical expression where the operator is placed after the operands, eliminating the need for the parentheses to show precedence. For this problem, you will be reading in an infix expression represented by a space separated string of symbols and (a) convert that to PostFix and (b) evaluate the expression using the Stack data structure.

Please make sure your program conforms to the following requirements

1. Call this program `solver.cpp`
2. You may only use the `iostream` and `string` libraries and the STL `stack` and `vector` classes. You may use the standard namespace.
3. Create a class called `Identifier` with 2 data members - a string for the name, and a double for the value. The class would only need a default constructor, a parameterized constructor,

and the getter and setter functions. Overloading the insertion and extraction operators is not required but would make your life easier.

4. You wouldn't need any specific functions here, just the main function. But you could separate the process into functions to convert and solve if you prefer that.
5. Read in the infix expression as a newline terminated string. Use this input to populate a vector of strings. Each element of the vector would contain one token - either an operand or an operator.
6. Use a stack and another vector of strings to convert this expression to postfix. Print the postfix expression.
7. Use the postfix expression vector, to populate a vector of the `Identifier` class. This vector will have an element for each identifier symbol in the postfix vector, along with its value.
8. Use the postfix vector and a stack to evaluate the value for the expression. Print the final answer.
9. The input expression will only contain valid C++ identifiers, numeric literals, parentheses, and one of the 4 binary arithmetic operators (+, -, *, /).
10. Explain in *analysis.pdf* why this process is linear.
11. **General Requirements**
    (a) Use the built-in functions of the Vector and stack classes when possible
    (b) Conform to standard software engineering practices (data hiding, function/code reuse, const keyword used where required, etc.)

Sample Run

```
Enter the infix expression:
a1 + ( b2 - 6.2 ) * ( 4 + c3 ) / variable
The postfix expression is
a1 b2 6.2 - 4 c3 + * variable / +
Enter the value of a1: 5.1
Enter the value of b2: -2
Enter the value of c3: 3.4
Enter the value of variable: 10
The expression evaluates to -0.968
```

4 Problem 3 - Printer Queues (Queues) - 50 points

A printer queue is used in a shared printer to print documents on a first come first serve basis. For this program, you will simulate a shared printer. Please make sure your program conforms to the following requirements

1. Call this program `printer.cpp`
2. You may only use the `iostream` and `string` libraries and the STL `deque` and `vector` classes. You may use the standard namespace.
3. Create a class called `Document`. This class contains a string for the name of the document, a string for the user printing the document, an integer for the number of pages, and a boolean to hold the document status (canceled or not). The class would only need a default constructor, a parameterized constructor, and the getter and setter functions. Overloading the insertion and extraction operators is not required but would make your life easier.

4. Create a class called `User` that contains a string for the user's name, and an int for the total number of pages that that user printed. The class would only need a default constructor, a parameterized constructor, and the getter and setter functions. Overloading the insertion and extraction operators is not required but would make your life easier.

5. In the main function, create a vector of Users and a deque of documents.

6. Start off the simulation by reading in the current jobs in the queue. Start a "timer" of sorts by setting it to 0. Every minute would print 8 pages.

7. Present a menu to the user. The user may choose between 'A' - add a new print job or 'C' - cancel a print job once a minute. That is, the user may add jobs or cancel jobs every 8 pages. The user may also enter 'N' for next. In this case, the printer would print the next 8 pages without altering the job queue.

8. If an unprinted job is canceled, do not remove it from the queue. Just mark it as canceled. You may assume document names in the deque will be unique. You may write a function to find an element in the deque to help.

9. The menu will be presented for the first time after the first 8 pages have been printed. The program will end on its own at this point if the jobs originally in the deque totaled to fewer than 8 pages.

10. The simulation ends when all jobs have been printed or canceled. To clarify, stop presenting the menu to the user if the deque is empty or if all jobs in the deque are canceled jobs.

11. While simulating, also aggregate the total pages printed by every user. Canceled print jobs do not get accumulated into this total.

12. At the end, print the total time taken for the simulation, and the number of pages printed by every user.

13. **General Requirements**
    (a) Use the built-in functions of the Vector and stack classes when possible
    (b) Conform to standard software engineering practices (data hiding, function/code reuse, const keyword used where required, etc.)

Sample Run

```
Enter the number of jobs in the Queue: 4
Enter the jobs:
Doc1.txt whalley 4
Syllabus.pdf liux 7
test1.docx myers 13
receipt.jpg gorham 2

A - Add a new job
C - Cancel a job
Enter your choice: A
Enter the job:
assignment1.cpp jayarama 12

A - Add a new job
C - Cancel a job
Enter your choice: C
Enter the document name: Doc1.txt
Error: Document already printed

A - Add a new job
C - Cancel a job
Enter your choice: A
Enter the job:
Schedule.pdf liux 20

A - Add a new job
C - Cancel a job
Enter your choice: A
Enter the job:
program.py roy 3

A - Add a new job
C - Cancel a job
Enter your choice: N

A - Add a new job
C - Cancel a job
Enter your choice: C
Enter the document name: program.py
Job Canceled

A - Add a new job
C - Cancel a job
Enter your choice: N

Queue cleared.
Total time: 8
User Totals:
whalley: 4
liux: 27
myers: 13
gorham: 2
jayarama: 12
```

5 Submission Instructions

You will be submitting the file directly through the programming server. Please follow these instructions:

1. Please create a tarball called `FSUID_HW1.tar`, where FSUID is your Canvas login, with the following files:
   - space.cpp
   - solver.cpp
   - printer.cpp
   - analysis.pdf
2. When you're ready to submit, first change the file permissions to give us read permissions:
   `chmod 704 FSUID_HW1.tar`

3. Run the submission script:
   `~jayarama/homework/submitHW1 FSUID_HW1.tar`

Please note the following:

1. Do not copy-paste the submission script. PDF's use Unicode characters that won't copy correctly on your terminal. Type it in.
2. THE FILE HAS TO BE NAMED CORRECTLY. You run the risk of overwriting other files
3. WE ONLY NEED YOUR TAR FILE. This is the file that contains your solution. Please don't submit your object file or the executable.
4. IF YOU GET THE "File Submitted" MESSAGE, YOU HAVE SUCCESSFULLY SUBMITTED THE FILE.
5. IF YOU MAKE MULTIPLE SUBMISSIONS, THE OLDER SUBMISSIONS WILL BE DELETED. The script will only keep the latest version of your file, and that is the submission we'll grade.

6 General Guidelines

1. Please include a line with your name and your FSUID in each file you turn in.
2. **This is individual work. You may NOT collaborate with other students in the course, former students, hire tutors to "help", copy solutions off the internet, or use pay-for solution websites, including but not limited to Chegg, CourseHero, WiseAnt, Bartelby, WizePrep, tutor.com, assorted Social Media groups, Discord servers where people offer "100% plagiarism-free solutions", any GroupMe groups students might have created, etc.) This includes posting the problem statements on these websites even if you do not use the answer obtained. Doing so is a violation of the Academic Honor Code. Violation of the Honor Code will result in a 0 grade on the program, a reduced letter grade in the course and potentially more serious consequences.**
3. **While using assistive technologies to generate your code might seem like a smart thing to do right now, you will eventually find yourself in a difficult position of being unable to pass the course due to failing the test average requirement**. At this level, it is recommended you write the program yourself to truly understand how programming works. Also, this would violate the Honor Policy.
4. Please make sure you're only using the concepts already discussed in class.
5. If we have listed a specification and allocated point for it, you will lose points if that particular item is missing from your code, even if it is trivial.
6. No global variables (variables outside of main()). constants and symbolic constants are allowed.
7. No `goto` statements.
8. All input and output must be done with streams, using the library `iostream`.
9. Please make sure that you're conforming to specifications (program name, function names, parameters and return types, print statements, expected inputs and outputs etc.). Not doing so will result in a loss of 10 points. Please note that names are case sensitive.

10. Try not to use `break` or `continue` statements in loops. break is allowed only for preventing the fall-though in a `switch` statement. Unnecessary and frivolous use of these would result in a loss of 10 points per statement.
11. NO C style printing is permitted. (Aka, don't use printf). Use cout if you need to print to the screen.
12. When you write source code, it should be readable and well-documented (comments).
13. Make sure you either develop with or test with linprog (to be sure it reports no compile errors or warnings) before you submit the program.
14. Testing your program thoroughly is a part of writing good code. We give you sample runs to make sure you match our output requirements and to get a general idea of how we would test your code. Matching your outputs for JUST the sample runs is not a guarantee of a 100. We have several extensive test cases.
15. Please make sure you've compiled and run your program before you turn it in. Compilation errors can be quite costly. We take 5 points off per compiler error for the first 9 errors. The 10th compiler error will result in a grade of 0.
16. Only a file turned in through the submission script counts as a submission. A file on your computer or linprog, even if it hasn't been edited after the deadline, does not count.
17. The student is responsible for making sure they have turned in the right file(s). We will not accept any excuses about inadvertently modifying or deleting files, or turning in the wrong files.
18. **Program submissions** should be done through the linprog submission script (if it's not there yet I'll create it soon.) Do not send program submissions through e-mail – e-mail attachments will not be accepted as valid submissions.
19. The ONLY files you will submit via linprog is `HW1.tar`
20. **General Advice** - always keep an untouched copy of your finished homework files in your email. These files will have a time-stamp which will show when they were last worked on and will serve as a backup in case you ever have legitimate problems with submitting files through linprog. Do this for ALL programs.