CS251 - Project 2: Stacks/Queues

Out: January 21, 2019 @ 8:00 am
Due: February 4, 2019 @ 8:00 am

1. Overview

Project 2 is split into three parts. In the first part, you will implement a stack class. In the second part, you will use your stack to solve a word-search problem. Finally, in the third part you will write a specialized type of queue called a circular double-ended queue.

2. Detailed Description

Part 1) Implement a stack that stores ordered pairs (i.e (2,3) or (5,8)). **(20 points)**

- Your task in part 1a will be to implement a growable stack using a linked list format. See the lecture slides for details on this data structure.
- **You may not use C++ STL classes or any of Java's built-in classes (linked list, stacks, Pair etc.) for this section. Implement this on your own!**

Part 2) Use your stack to solve a word search problem. **(40 points)**

In this part, you are going to solve a word-search problem, but with a bit of a twist. Normally in these puzzles, you are looking for a word that is written out in a straight line, such as below:

S A B C D
E **T** F G H
I J **A** K L
M N O **C** P
Q R S T **K**

However we want to solve something a bit trickier. In this project, we want to allow the words to be hidden in more interesting patterns, which may not be a line. Such as below:

A B C D E
S G **A** I J
K **T** M **C** O
P Q R **K** T
U V W X Y

Unlike above, we allow ourselves to move any direction we want from any letter. Our objective, in this case, is to print out the location of the letters that make up the specified word. Moreover, the output should be in an ordered pair format. The solution for the above example is:

1 0
2 1
1 2
2 3
3 3

The first number per line is the row number and the second number per line is the column number (both starting at zero from the top or the left, respectively). Thankfully, stacks provide a solution to problems like this! The basic algorithm to solve this problem works as follows:

- Begin by searching for the first letter in the word.
- If you find it, push the location onto your stack, and begin inspecting the immediately surrounding 8 (eight) cells for the next letter. If you are at the boundary of the matrix of letters, do not wrap around to the other side.
- If you find the next letter, push that location onto the stack and repeat until finished.
- If, at any point, you do NOT find the letter you want, pop the last location off the stack and resume the search from the previous stage.
- If the word is found, print out the ordered pairs in the correct order.
- If not, print "not found" (without quotes).

Additional notes for this section:

- Letters can be re-used in the search. In the given example DAD would be acceptable with the answer being:

  0 3
  1 2
  0 3

3. Circular Double-ended Queue. (40 points)

In a standard queue, objects are inserted at the back and pulled from the front in a First In First Out (FIFO) fashion. There is a variation of a standard queue called a circular double-ended queue, often shortened to circular deque. As you may expect from the name, rather than inserting/deleting only on one side of the queue, these deques allow insertion or deletion from both sides on a circular array. The basic methods you will need to implement for this part are:

- enqueue_front: Add an item to the front of the queue
- enqueue_back: Add an item to the back of the queue
- dequeue_front: Take an item off the front of the queue
- dequeue_back: Take an item off the back of the queue
- print_array_size: Print the total size of the allocated array (i.e., how many items could the array store when totally full)
- print_front_index: Print the index of the front pointer
- print_rear_index: Print the index of the rear pointer

Your task is to implement a circular double-ended queue using the array-doubling strategy. As in part 1, you cannot use C++ STL classes or predefined java interfaces like deques. You must write your own. You should start with an initial array size of 2.

Regarding front and rear position,

- If there are no elements, then front and rear index should be reset to -1.
- The first element should be put into index 0 and front, rear should be pointed to index 0.
- While doubling the array you should put the previous contents into the new array and relocate the elements such that the element pointed by front starts from index 0. Therefore, the new front index should be point to 0 and rear should be point to index of the last element.

Example: 'e' or 'd' represents enqueue or dequeue followed by another character 'f' or 'b' that represents the operation on the front or back of the queue followed by an integer to use in the operation ("-" denotes that the cell is empty).

Command	Front Index	Rear Index	Queue Contents
	-1	-1	[-,-] (Initial size of Empty queue : 2)
e f 1	0	0	[1,-]
e b 2	0	1	[1,2]
e f 3	3	1	[1,2,-,3]
e f 4	2	1	[1,2,4,3]
e b 5	0	4	[4,3,1,2,5,-,-,-]
e f 6	7	4	[4,3,1,2,5,-,-,6]
d b	7	3	[4,3,1,2,-,-,-,6]
d f	0	3	[4,3,1,2,-,-,-,-]

4. Input/Output Format

Input and output will be handled using files similar to Project 1. The input and output filepath/filename will be passed as two command line arguments. The provided skeleton code shows how different problems are called based on the first value (Line 1) of the input file. You have to handle remaining portion as per the following description. All your output should be written into a output file passed as an argument.

Line 1: either 1, 2, or 3 declaring which part of the project is being tested.

For project part 1 the input is as follows:

- An integer n that represents the number of lines to follow.
- n lines of the following format.
  The character 'i' (for insert) followed by 2 integers. Push these onto the stack as an ordered pair. The character 'p' (for pop) followed by nothing. Pop an ordered pair and print the integers separated by one space in between them. If the stack is empty print the string "empty". (Lowercase and no punctuation please!)

For project part 2 the input is as follows:

- A line containing 2 integers "n m" (no quotes) that describes the size of the word search puzzle.
- n lines each containing m characters
- One line containing the word you are searching for
- Output is the list of ordered pairs that represents the location of the word. Print "not found" if the word is not found.

For project part 3 the input is as follows:

- An integer n that represents the number of lines to follow
- n lines of the following format
  Either the character 'e' or 'd' representing enqueue or dequeue, followed by space and another character 'f' or 'b' indicating that the operation is on the front or back of the queue, followed by a space and the integer to use in the operation
  The character 's' in which case you print the allocated size of the deque, the front index followed by the rear index delimited by space.
- All dequeued objects should be printed. Each dequeue commands dequeues an item either from front or back. You have to print that item in a new line. Check expected output for details.

For this project, you can assume that all input we give you is well-formatted and valid.

Sample test cases:

Input	Expected output
1 5 i 5 2 i 4 3 p p p	4 3 5 2 empty
2 5 5 A B C D E S G A I J K T M C O P Q R K T U V W X Y STACK	1 0 2 1 1 2 2 3 3 3
3 8 e b 1 e b 2 e f 3 d b d f e b 5 d f s	2 3 1 4 1 1

5. Grading and Programming Environment

Assignments will be tested in a Linux environment. You will be able to work on the assignments using the Linux workstations in HAAS and LAWSON (use your username and password).

C++ Instructions:

Compilation will be done using g++ and makefiles. Create a folder with your username. Put all the source code (should not have any sub-directories in your submission folder) as well as the Makefile that compiles your provided source code into an executable named "**program**".

Ex: g++ -o program main.cpp stack.cpp circular_deque.cpp -std=c++11
./program <input file> <output_file>

Your project must compile using the standard g++ compiler on data.cs.purdue.edu.
For convenience, you are provided with a template Makefile and C++ source file.

Java Instructions:

- Compiler we use: javac (v10.0.2)
- Files to submit: Create a folder with your username. Put all java files into the folder. **There should not be any sub-directories.** Your main program must be named **Main.java**. **We will run that one only**. Your project must compile using the javac compiler (v10.0.2) on data.cs.purdue.edu.

Compiling process:

- To compile Main:
  1. Go to the project root.
  2. Type command in console: javac src/*

- To run Main:
  1. Go to the project root.
  2. Type command in console: java -cp src/ Main inputfiles/test1.txt outputfiles/output-test1.txt

The grading process consists of:

1. Compiling and building your program using your supplied makefile.
2. The name of produced executable program must be "program" (must be lowercase)
3. Running your program automatically with several test input files we have pre-made according to the strict input file format of the project and verifying correct output files. Thus, follow the above instructions precisely – **do not "embellish" the output with additional characters or formatting** – if your program produces different output such as extra prompt and space, points will be deducted.
4. Inspecting your source code.

Input to the programming projects will be via files. The output should be written to the output file passed via command line argument. The output file will be tested for grading.

Important:

1. **If your program does not compile, your grade will be 0 - no exceptions.**
2. **If you don't follow the submissions instructions, your grade will be 0 - no exceptions..**
3. **Plagiarism and any other form of academic dishonesty will be graded with 0 points as definitive score for the project and will be reported at the correspondent office.**

Submission Instructions:

The homework must be turned in by the due date and time using the turnin command. Follow the next steps:

1. Login to data.cs.purdue.edu (you can use the labs or a ssh remote connection).
2. Make a directory named with your username and copy your solution (**makefile, all your source code files including headers and any other required additional file**) there. I.e. **all your source files must be present in a single directory under your username (If you fail to do so, your grade will be 0)**.
3. Go to the upper level directory and execute the following command:
   turnin -c cs251 -p project2 your_username
   (**Important:** previous submissions are overwritten with the new ones. Your last submission will be the official and therefore graded).
4. Verify what you have turned in by typing **turnin -v -c cs251 -p project2**
   (**Important:** Do not forget the **-v** flag, otherwise your submission would be replaced with an empty one)

Overview / additional notes:

- Part 1: Implement a stack using linked list format
- Part 2: Use your stack to solve a word search problem
- Part 3: Implement a deque using the array-doubling strategy

- You may **NOT** use the C++ STL classes or Java built in collections in your stack or queue implementations!

- We provide you with some skeleton code in this project. It's use is completely optional. You can also feel free to edit it as you wish.