Universität Zürich[UZH]

VISUALIZATION AND MULTIMEDIA LAB

Algorithms and Data Structures

Assignment 2
Deadline: Sunday, 19th of March 2012, 23:59
Hand-in: in OLAT, https://www.olat.uzh.ch

Task A): Theory Questions

1) Which of these real world examples correspond to a stack?
a) Waiting at the check-in line at the airport.
b) Undo a change in a text editor.
c) Taking a cookie from a box full of snacks.
d) Putting items into a shopping bag one above another.

2) What is the main reason to have a runtime stack?
a) Keep track of the point to which each active subroutine should return control, when it finishes executing.
b) Keep track of the used memory by the program.
c) To calculate the CPU usage.
d) To communicate with the operating system.

3) What happens when the function dequeue() is called?
a) An object at the rear of a queue is inserted.
b) An object at the front of a queue is inserted.
c) The element at the front of a queue is removed.
d) The element at the rear of a queue is removed.

4) Which function is not an accessor function?
a) size()
b) isEmpty()
c) front()
d) dequeue()

5) Which of the following statements are true for an array based stack?
a) Arrays are always dynamic and the stack could grow to infinity.
b) An initial index is necessary to build an array based stack.
c) Each time we use push() the index has to be incremented.
d) While using top() the index has to be decremented.

6)Which of these complexity orders is wrong?
a) O(log n) ε $O(n)\varepsilon$ $O(n^3)$
b) $O(nlogn)\varepsilon$ $O(n)\varepsilon$ $O(n!)$
c) $O(1)\varepsilon$ $O(n^3)\varepsilon$ $O(3^n)$
d) $O(loglogn)\varepsilon$ $O(logn)\varepsilon$ $O(n)$

7) On which position stands the active element of an array-based stack with the length n?
a) 0
b) n
c) n+1
d) n-1

8) Which of the following denotes the way elements in the stack are accessed?
a) LIFO
b) FIFO
c) FILO

Task B): Array Based Stack

We have a football card game and the deck of the game is implemented as a stack. We are not interested in the mechanics of the game but only in the functionality of the deck. A stack provides those functions; constructor(), destructor(), pop(), push(), top(), empty(), size(). The corresponding files for this task are in Exercise 1 folder.

1) Implementation

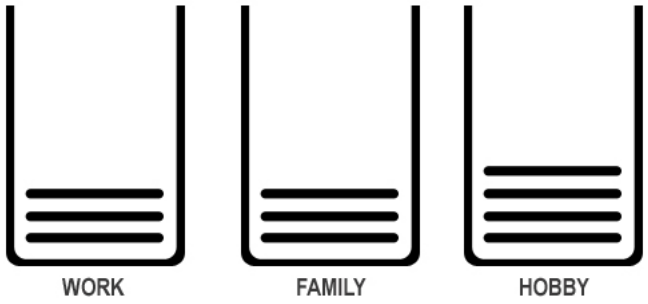
The header file is already provided and the **main.cpp** is filled with code to test the card deck with FC Barcelona. Your exercise is to implement all the functions in the **stack_ab.cpp** .

Hint: Be aware of stack over/underflow, try to get the knowledge of exceptions (for later usage).

Task C): ADT Stack

The aim of this task is to get an overview of the ADT Stack. You were in the holidays and your assistant made a mess with your mails. You have three trays, each for its

own specific purpose. So all the letters have a purpose and therefore have to be in the corresponding right tray. The respective files for this task are in Exercise 2 folder.



1) printTrays()

In the file *main.cpp* you have the skeleton of a method named **printTrays()** which should print an simple overview of the trays with the letters in it on the standard output.

2) cleanUp - Algorithm

We provide you a function skeleton named **cleanUp()** which should clean up the mess your assistant made. After this method has been called, every letter should be in the right tray.

Hint: Use a temporary tray to order all the letters.

3) Order priority for experts

Order the letters in priority so the letter with the highest priority is on top of the stack.

Task D): Array-based Queue

Imagine you want to develop your own simple operating system. You have reached the point where you are coding the process management system of your OS. You have decided to use a simple array-based queue to manage the processes waiting to get access to the CPU. In this exercise, you will have to program some of the queue operations. The corresponding files for this task are in Exercise 3 folder.

1) Complete the methods

In the file process_queue.cpp you have two function skeletons: **get_size()**, which should return the number of elements contained in the queue and **is_empty()** which should return a boolean value indicating if the queue is empty or not. Complete these two functions accordingly.

2) Adding element in the queue

The skeleton of the method **add(process* my_process)** is given. The method is supposed to add the process pointed by the pointer **my_process** to the queue. Complete the function correctly, and do not forget to update class variables if necessary.

3) Removing element from the queue

The function **remove()** is given. It is supposed to remove the first process of the queue and return a pointer of that removed process.

Hint: Be careful with special cases and consider them too (for eg. removing from an empty queue etc.). Also remember that you have to shift the remaining elements in the queue by one position after removing an element.

Task E): More Queues

In this exercise we learn how to work with queues in C++. The corresponding files for this task are in Exercise 4 folder. You will find the necessary information about C++ queues here: *http://www.cplusplus.com/reference/stl/queue/*

1) Traffic Light

Imagine a traffic light in front of a crossroad. The arriving cars will pass the crossroad if the traffic light is green or they will wait if the light is red. Your goal is to implement the line of cars with a C++ queue. The street in our example has two lanes, so you have to use two queues. An arriving car will always wait in the shortest line.
After every new arriving car the user can switch the traffic light. If the traffic light turns green all waiting cars will leave the queue.