Data Structures and Algorithms I
Spring 2021
Homework #2
Jacob Khalili

96/100

(1) For each of the following descriptions of something stored in memory when a program that was coded in C++ is running, state whether the memory is part of static memory, an activation record in the stack, or the heap (i.e., dynamic memory). For those entities that are stored in an activation record, state whether the memory resides in the first, second, or third section of the activation record, according to the breakdown discussed in class. For some of these parts, you may have to speculate about the most likely way that a provided class is implemented.

48/50

(a) A float that is a global variable. – **Static**
(b) An int that is a parameter of a regular function. - **Activation record: Parameters**
(c) A char that is a parameter of a protected member function of a class. - **Activation record: Parameters**
(d) A double that is a private static data member of a class. – **Static**
(e) An int that is a public data member of a dynamically allocated object if the object is pointed to by a local pointer of a regular function. – **Heap**
(f) A bool that is a private data member of an object this is a parameter of a public member function of a class. – **Activation record: Parameters**
(g) A char that is stored in a C++ string object containing many characters, if the string object is a local variable of a public member function of a class. **Heap**
(h) A char that is stored in a C++ string object containing many characters, if the string object is a parameter of a constructor of a class. **Heap**
(i) An int that is a temporary variable created by the compiler that is local to a private member function of a class. **Activation record: Local & Temp Vars**
(j) The value of a register that has been backed up when a private member function has been invoked through a global object (the value will need to be restored when the member function returns). **Activation record: machine status info**
(k) A double that is a private data member of a global object. **Static**
(l) A double that is part of a vector of doubles if the vector is a local variable in a protected member function of a class. **Heap**
(m) An int that is a protected data member of an object that is contained in a vector of objects if the vector is a parameter of a public member function of a class. **Heap**
(n) The memory address stored by a pointer to an object (i.e., the value of the pointer, not the value of the thing that it points to) if the pointer is a private data member of a global object, and the object that it points to is a parameter of a regular function. **Static**
(o) The memory address stored by a pointer to a char (i.e., the value of the pointer, not the value of the thing that it points to) if the pointer is a parameter of a public member function of a class, and the char that it points to is stored of a C++ string object. **Activation record: Parameters**
(p) The value that a reference to a double refers to, if the reference is a parameter of a private member function of a class, and the argument passed to it is a local variable in a regular function. **Activation Record: Parameters** AR local & temp AR correct! section wrong
(q) An int that is a public static data member of an object that is part of a C++ list of objects if the list is a local object of a public member function of a class. **Static**

Data Structures and Algorithms I
Spring 2021
Homework #2
Jacob Khalili

48/50

(2) Answer the following questions concerning lists, stacks, and queues:

(a) Suppose that a printer queue relied on a stack instead of a queue. Why would this be annoying?

Because if you print something while the printer is busy, and someone else prints after you yours would not be the next to print. It would print the newer ones first. The more people print after you the worse it gets, until there are no new requests for a while and the stack empties.

#1	New paper - 1 min
#2	New paper - 2 mins
#3	Your paper from 3 days ago

= order of completion.

(b) Assume you have access to an implementation of a queue class that provides worst-case constant time enqueue (push) and dequeue (pop) operations. Explain how to use two such queues to implement a stack with worst-case constant push and worst-case linear pop.

You could use two queues by loading all the data into one queue, and when it comes time to pop, load all the data into a 2nd queue, until you get to the last element and return that.

```
push(new data):
    stack1.push(new data)

pop():
    while (1 < stack1.size()){
            stack2.push(stack1.pop())
    }

    temp = stack1.pop()
    stack1 = stack2;
    stack2.empty();
    return temp;
```

(-1) pseudo code not needed, although but this is wrong. You need to swap or exchange the two stacks. This would lead to both stacks being the same (& empty)

(c) Assume you have access to an implementation of a queue class that provides worst-case constant time enqueue (push) and dequeue (pop) operations. Explain how to use two such queues to implement a stack with worst-case linear push and worst-case constant pop.

You could use two queues by reversing the order every time you load in. Add the new data to a new queue and copy all the existing from the original queue to it. Then the pops with be constant time.

```
push(new data):
    stack2.push(new data)
    while(stack1.size > 0){
```

Data Structures and Algorithms I
Spring 2021
Homework #2
Jacob Khalili

```
            stack2.push(stack1.pop())
    }
    stack1 = stack2
    stack2.empty()
pop():
    return stack1.pop()
```

(d) Assume you have access to an implementation of a stack class that provides worst-case constant time push and pop operations. Assume the stack has been implemented to support generic programming (e.g., using templates in C++). Explain how to use only one such stack to implement a data structure that supports the same push and pop operations as the provided stack class, and also a getMinValue operation that returns the value of the minimum item in the data structure (but does not alter the data structure), supporting all three types of operations (push, pop, and getMinValue) in worst-case constant time. You can assume that the values of items are comparable to each other using the standard comparison operators (<, <=, ==, >, >=).

```
from queue import LifoQueue

class NewStack:
  stack = LifoQueue(maxsize = 75)
  minElement = 0

  def __init__(self):
    print("")

  def push (self, element):
    if self.stack.qsize() == 0:
      self.minElement = element
    if element <= self.minElement:
      self.stack.put(self.minElement)
      self.minElement = element
    self.stack.put(element)

  def pop (self):
    elementor = self.stack.get()
    if (elementor == self.minElement):
      self.minElement = self.stack.get()
    return elementor

  def getMinValue(self):
    return self.minElement
```

Data Structures and Algorithms I
Spring 2021
Homework #2
Jacob Khalili

This works because whenever an element smaller than the current minimum gets added an extra record containing the previous minimum is also added before the new record. And when we go to retrieve the lowest record we know to delete the one before it and use it as the new minimum.

https://colab.research.google.com/drive/19xpznX5A0jKHsrK2neUuHgmNNxWZsQa2?usp=sharing

I'm judging printout. I assume this is [illegible]

(e) If the recursive routine used to compute Fibonacci numbers (covered in class as part of a previous topic) is run for N = 50, is stack space likely to run out? Explain your answer.

No, because although the program makes well over a million function calls, it has a maximum of about 50 concurrent calls at any given times, thus the stack empties before it has time to completely fill up.