EECS 268 Home-Work Quiz 5 (Stacks and Queues) - Fall 2012

Do not write you name on this answer sheet (only your KU-ID). Total: 50 points

Name: **Prasad Kulkarni**

1. What are linear data structures. Give an example. **(4 points)**

   Linear data structures are those where each item has specific first, next and previous relations with other items in the set.
   Examples: arrays, linked lists, vectors, strings.

2. What restrictions are added to the List ADT by: (a) Stacks, and (b) Queues ? **(6 points)**

   Stacks: Items can only be inserted and removed in a LIFO fashion from one end of the list. Thus, the Stack ADT prevents inserting and deleting items anywhere in the list.

   Queues: Items can only be inserted and removed in a FIFO fashion. Items can only be inserted to the front of the list. Items can only be deleted from the end of the list. Thus, the Queue ADT also prevents inserting and deleting items anywhere in the list.

3. What is the output of the following pseudo-code, where `num1`, `num2`, and `num3` are integer variables? **(4 points)**

```
num1 = 4;
num2 = 10;
num3 = 2;
aQueue.enqueue(num2);
aQueue.enqueue(num3);
aQueue.dequeue();
aQueue.enqueue(num1-num2);
aQueue.dequeue(num1);
aQueue.dequeue(num2);
cout << num1 << " " << num2 << " " << num3 << endl;
```

```
2 -6 2
```

4. Assume that you have access to a working implementation of the Stack ADT. Knowing the Stack ADT operations, implement the C++ function `isBalBraces(String str)` that checks for balanced braces in the String `str`. **(6 points)**

```
isBalBraces(String str){
  Stack stk;
  for(int i=0 ; i<str.length() ; i++){
    switch(str[i]){
      case '{': stk.push(str[i]);
        break;
      case '}': if(!stk.isEmpty())
                   stk.pop();
                else{
                   cout << ''Braces are not balanced!'' << endl;
                   return false;
                }
        break;
    }
  }
  if(!stk.isEmpty()){
    cout << ''Braces are not balanced!'' << endl;
    return false;
  }
  return true;
}
```

5. For each of the following situations, which of these ADTs would be most appropriate: (a) Queue; (b) Stack; (c) List. **(4 points)**

   (a) Manage a sorted list of integers. **c**

   (b) A grocery list ordered by the occurrence of the items in the store. **c**

   (c) The list of commands managed by the Linux command-line interface shell that allows the use of 'up' and 'down' arrow keys to traverse past history of commands. **b**

   (d) A program that uses backtracking. **b**

6. Assume that you have access to a working implementation of the Stack ADT. Knowing the Stack ADT operations, implement the C++ function `evalPostfix(String str)` that evaluates a postfix expression provided in the String `str`. Assume that all operands are single-digit numerals (0–9) and the only possible operators are $+, -, *, /$. **(6 points)**

   Will be covered during the lab.

7. What is the problem of *rightward-drift* associated with the array-based implementation of the Queue ADT? Does an array-based implementation of the Stack ADT suffer from the same problem? Why? **(6 points)**

A Queue ADT adds (*enqueues*) items to the back of the queue, while retrieving (*dequeuing*) items from the front of the queue. As a result, the part of an array holding active queue items (between the front and back of the queue) keeps shifting to the right as we add/remove items from the queue. Eventually, the back of the queue may reach the end of the array preventing further enqueues even though there is space available in the array. This is called the problem of rightward drift.
For example, given the array:
|2|4|10|3|9|4| | | | | | | | | | | | |
After three enqueue (inserting 5, 12, and 7 respectively) and dequeue operation each, the array would be:
| | | |3|9|4|5|12|7| | | | | | | | | |

A Stack does not suffer from rightward drift because we can only add (*push*) and remove (*pop*) items from one end of the array.

8. Provide a solution to the *rightward-drift* problem associated with the array-based implementation of the Queue ADT. Explain the idea behind your solution and describe how you would update your Queue ADT variables during *initialization*, *enqueue* and *dequeue*. **(6 points)**

A solution to the rightward drift problem for Queues may use a *circular* array instead of the usual *flat* array. Thus, on reaching the end of the queue, further enqueue operations will add items starting from the front of the queue again if there is room. An integer counter holding the number of active items in the queue can be used to check for array empty/full conditions.
Initialization:
`front=0, back=MAX_QUEUE-1, count=0`

Enqueue:
`back=(back+1)%MAX_QUEUE, items[back]=newItem, count++`

Dequeue:
`front=(front+1)%MAX_QUEUE. count--`

9. List the tradeoffs between using arrays or linked lists to implement the Stack ADT. **(4 points)**

   Tradeoffs include:

   - Arrays provide a fixed-size stack. Linked lists allow the stack to grow and shrink at runtime.
   - An array-based stack implementation can use a compiler-generated copy constructor and destructor. Linked list based implementations typically need to developer to provide explicit definitions of both.
   - Each enqueue and dequeue operation with a linked list based Stack ADT may need to do the additional work of allocating or destroying a node respectively, which is more expensive than similar operations on the array-based stack.

10. Use the stack algorithm shown in class to convert the infix expression "`a+(b-c)*c+d/(e-a*b)+c`" to postfix. **(4 points)**

    The corresponding postfix expression is: `abc-c*+deab*-/+c+`