Second Year – Term II
Systems & Biomedical Eng. Department
Cairo University

Time Allowed: 75 minutes
Date: April 2013

SBE 201 - Data Structure
Midterm Exam
Model Answer

1.

a. **_______ form of access is used to add and remove nodes from a queue.**

   i. LIFO, Last In First Out
   ii. FIFO , First In First Out
   iii. Both 1 and 2
   iv. None of these

b. What happens when you push a new node onto a stack?

   i. **the new node is placed at the front of the linked list.**
   ii. the new node is placed at the back of the linked list.
   iii. the new node is placed at the middle of the linked list.
   iv. No Changes happens

c. What is a pointer? Is it possible to have two pointers point to the same data? Explain your answer by an example.

A pointer is a variable that stores address of other variable.

Yes, we can have two pointers to the same variable.

Example:

```
int *x, *y, z =10;

x= &z;
y= &z;
```

d. What does it mean when a pointer is used in an if statement?

Any time a pointer is used as a condition, it means "Is this a non-null pointer?" A pointer can be used in an if, while, for, or do/while statement, or in a conditional expression.

```
if ( p ){
        /* do something */
}
else {
        /* do something else */
}
```

e. Can you subtract pointers from each other? Why would you?

If you have two pointers into the same array, you can subtract them. The answer is the number of elements between the two elements.

f. Can you add pointers together? Why would you?

No, you can't add pointers together. If you live at 1332 Tahrir Street, and your neighbor lives at 1364 Tahrir Street, what's 1332+1364? It's a number, but it doesn't mean anything. If you try to perform this type of calculation with pointers in a C program, your compiler will complain.

The only time the addition of pointers might come up is if you try to add a pointer and the difference of two pointers:

p = p + (p2 - p1);   OR   p += p2 - p1;

2. Assume you construct a single linked list with patient records as nodes. Implement an algorithm to delete a patient record (a node) in the middle of that single linked list, given only access to that patient record (node). For example: Node d should be removed from the linked list a → b → c → d → e. Your algorithm should take one input, i.e. pointer to node d. Your algorithm should return true if the node is successfully removed, i.e. the linked list will look like a → b → c → e. Your algorithm should return false if the input node is not in the middle.

Answer:

The solution to this is to simply copy the data from the next node into this node and then delete the next node.

NOTE: This problem can't be solved if the node to be deleted is the last node in the linked list.

```
boolean deleteNode(LinkedListNode *n) {
        if (n == null || n->next == null) {
                return false;                       // Failure
        }
        LinkedListNode *next = n->next;
        n->data = next->data;
        n->next = next->next;
        free (next);
        return true;
}
```

3. Assume that you have two stacks S1 and S2 with their complete interface.

a. How would implement a queue named MyQueue by using S1 and S2 only.

b. Write the "dequeue" function of MyQueue

Answer

Since the major difference between a queue and a stack is the order (first-in-first-out vs. last-in-first-out), we know that we need to modify peek() and pop() to go in reverse order. We can use our second stack S2 to reverse the order of the elements (by popping S1 and pushing the elements on to S2). In such an implementation, on each peek() and pop() operation, we would pop everything from S1 onto S2, perform the peek / pop operation, and then push everything back.

This will work, but if two pop / peeks are performed back-to-back, we're needlessly moving elements. We can implement a "lazy" approach where we let the elements sit in S2.
S1 will thus be ordered with the newest elements on the top, while S2 will have the oldest elements on the top. We push the new elements onto S1, and peek and pop from S2.When S2 is empty, we'll transfer all the elements from S1 onto S2, in reverse order.

```
void add(T value) {
        s1.push(value);
}

T peek() {
        if (!s2.empty()) return   s2.peek();
        while (!s1.empty())   s2.push(s1.pop());
        return   s2.peek();
}

T remove() {
        if (!s2.empty()) return   s2.pop();
        while (!s1.empty())   s2.push(s1.pop());
        return s2.pop();
}
```

SBE 201 - Data Structure
Midterm Exam
Model Answer

4. Describe how you could use a single array to implement three stacks. Write a push function that can be used to insert an item into a selected stack of the three stacks.

Answer:

Divide the array in three equal parts and allow the individual stack to grow in that limited space (note: "[" means inclusive, while "(" means exclusive of the end point).
»»for stack 1, we will use [0, n/3)
»»for stack 2, we will use [n/3, 2n/3)
»»for stack 3, we will use [2n/3, n)

This solution is based on the assumption that we do not have any extra information about the usage of space by individual stacks and that we can't either modify or use any extra space. With these constraints, we are left with no other choice but to divide equally.

```
int stackSize = 300;
int buffer[stackSize * 3];
int stackPointer[3]= {0, 0, 0};                    /* stack pointers to track top item */

void push(int stackNum, int value) {
            /* Find the index of the top element in the array + 1, and
            * increment the stack pointer */
    int index = stackNum * stackSize + stackPointer[stackNum] + 1;
    stackPointer[stackNum]++;
    buffer[index] = value;
}

int pop(int stackNum) {
    int index = stackNum * stackSize + stackPointer[stackNum];
    stackPointer[stackNum]--;
    int value = buffer[index];
    buffer[index]=0;
    return value;
}

int peek(int stackNum) {
    int index = stackNum * stackSize + stackPointer[stackNum];
    return buffer[index];
}

boolean isEmpty(int stackNum) {
    return stackPointer[stackNum] == stackNum*stackSize;
}
```