EE 453/717: Advanced Computing for Electrical Engineers

Instructors: Saravanan Vijayakumaran, Sachin Patkar

Indian Institute of Technology Bombay

Autumn 2011

Solutions to Quiz 1

---

1. Consider two singly linked lists of integers whose elements are sorted in descending order. Their lengths can be different and they can also be empty. If a list is nonempty, the last node in it points to `NULL`. Write a C++ function which takes the pointers to the first nodes of these two sorted linked lists as inputs and returns a pointer to the first node of a linked list which contains the integers from the both the lists in descending order. The signature of the function should be the following: `node* MergeLists(node* list1, node* list2);` where the definition of `node` is the following

```
struct node{
    int data;
    node* next;
}
```

Solution:

```
node* MergeLists(node* list1, node* list2)
{
    node* mergedListHead, mergedListTail;
    if(list1 == null && list2 ==null)
    {//if both lists are null, return null
        return null;
    }
    if(list1 == null)
    {//if list1 is null, simply return list2
        return list2;
    }
    if(list2 == null)
    {//if list2 is null, simply return list1
        return list1;
    }
    if(list1.data > list2.data)
    { //initialize mergedListHead pointer to list1 if list1's data is greater
        mergedListHead = list1;
        mergedListTail = list1;
        list1 = list1->next;
    }
    else
```

```
    { //initialize mergedListHead pointer to list2 if list2's data is greater
        mergedListHead = list2;
        mergedListTail = list2;
        list2 = list2->next;
    }

    while(list1!=null && list2!=null)
    {
        if(list1.data > list2.data)
        {
            mergedListTail->next = list1;
            mergedListTail = mergedListTail->next;
            list1 = list1->next;
        }
        else
        {
            mergedListTail->next = list2;
            mergedListTail = mergedListTail->next;
            list2 = list2->next;
        }
    }

    if(list1 == null)
    { //remaining nodes of list2 appended to mergedListTail when list1
      //has reached its end.
        mergedListTail->next = list2;
    }
    else
    { //remaining nodes of list1 appended to mergedListTail when list2
      //has reached its end.
        mergedListTail->next = list1;
    }
    return mergedListHead;
}
```

2. Implement a queue using two stacks. Consider the `Queue` class below which contains two stacks as private data. Assume that the `Stack` class contains the methods `IsEmpty()`, `Push(T x)` and `Pop()`. Fill in the implementations of the `public` methods shown.

```
template<class T>
class Queue {
   private:
      Stack<T> stack1;
      Stack<T> stack2;
```

```
   public:
      bool IsEmpty();
      void Add(T x);
      T Remove(); // remove element at front of queue and return it
};
```

Solution:

```
void Queue::Add(T x)
{
    stack1.Push(x);
}

T Queue::Remove()
{
    while(!stack1.IsEmpty())
    {
        stack2.Push(stack1.Pop());
    }

    T temp = stack2.Pop();
    // This will throw an exception of stack2 is empty
    // which will happen when the Queue is empty

    while(!stack2.IsEmpty())
    {
        stack1.Push(stack2.Pop());
    }
    return temp;
}

bool Queue::IsEmpty()
{
    return stack1.IsEmpty();
}
```

3. Implement a stack using a queue. Consider the `Stack` class below which contains a queue as private data. Assume that the `Queue` class contains the methods `Size()`, `Add(T x)` and `T Remove()`. Fill in the implementations of the `public` methods shown.

```
template<class T>
class Stack {
   private:
      Queue<T> q;

   public:
```

```
      void Push(T x);
      T Pop();
};
```

Solution:

```
void Stack::Push(T x)
{
    q.Add(x);
}

T Stack::Pop()
{
    for (int i = 0; i < q.Size()-1; i++)
    {
        q.Add(q.Remove());
    }
    return q.Remove();
}
```