Tutorial Letter 202/1/2012
Programming: Data Structures

COS2611

Semester 1

School of Computing
This tutorial letter contains solutions to assignment 02

Bar code

Learn without limits.

UNISA university of south africa

Tutorial matter received so far	
COS2611/101/3/2012	General information and assignments
COS2611/201/1/2012	Solutions to Assignment 1
COS2611/202/1/2012	This letter: Solutions to Assignment 2

Dear student

The purpose of this tutorial letter is to supply the solutions for Assignment 2.

Please feel free to contact any of the lecturers with any problems, suggestions and/or queries. For this purpose, we appeal to you to make use of the *myUnisa* forum where solutions to problems/questions and answers to queries posted on the forum are available to the benefit of all students. We also encourage you to actively participate in this forum by providing support (not complete solutions!) to your fellow students.

Question 1: Linked Lists [5]

Show what is produced by the following C++ code. Assume the node is in the usual info-link form with info of type int. (list and ptr are pointers of type nodeType.)

```
list = new nodeType;
list->info = 20;
ptr = new nodeType;
ptr->info = 28;
ptr->link = NULL;
list->link = ptr;
ptr = new nodeType;
ptr->info = 30;
ptr->link = list;
list = ptr;
ptr = new nodeType;
ptr->info = 42;
ptr->link = list->link;
list->link = ptr;
ptr = list;
while (ptr != NULL)
{
cout << ptr->info << endl;
ptr = ptr->link;
}
```

Solution:

30
42
20
28

Question 2: Linked Lists [20]

Solution:

```
template <class Type>
void orderedLinkedList<Type>::mergeLists(orderedLinkedList<Type> &list1,
                                                  orderedLinkedList<Type>
&list2)
{
        nodeType<Type> *lastSmall;     //pointer to the last node of
                                         // the merged list.
        nodeType<Type> *first1 = list1.first;
        nodeType<Type> *first2 = list2.first;

        count = list1.count + list2.count;

        if (list1.first == NULL)   //first sublist is empty
        {
           first = list2.first;
           list2.first = NULL;
       count = list2.count;
        }
        else if (list2.first == NULL)   // second sublist is empty
    {
       first = list1.first;
       list1.first = NULL;
       count = list1.count;
    }
    else
    {
       if (first1->info < first2->info) //Compare first nodes
       {
           first = first1;
           first1 = first1->link;
           lastSmall = first;
       }
       else
       {
           first = first2;
           first2 = first2->link;
           lastSmall = first;
       }

       while (first1 != NULL && first2 != NULL)
       {
           if (first1->info < first2->info)
           {
               lastSmall->link = first1;
               lastSmall = lastSmall->link;
               first1 = first1->link;
           }
           else
           {
               lastSmall->link = first2;
               lastSmall = lastSmall->link;
               first2 = first2->link;
           }
       } //end while
```

```
        if (first1 == NULL)                  //first sublist exhausted first
            lastSmall->link = first2;
        else                                 //second sublist exhausted first
            lastSmall->link = first1;

        list1.first = NULL;
        list1.last = NULL;
        list2.first = NULL;
        list2.last = NULL;
        count = list1.count + list2.count;
         }
} //end mergeList
```

Test program:

```
//22 34 56 2 89 90 0 -999
//14 56 11 43 55 -999

#include <iostream>
#include "orderedLinkedList.h"

using namespace std;

int main()
{
        orderedLinkedList<int> newList, list1, list2;

        int num;

        cout << "Enter numbers, for list1, ending with -999" << endl;

        cin >> num;

        while(num != -999)
        {
          list1.insert(num);
          cin >> num;
        }

        cout << "list1: ";
        list1.print();
        cout << endl;
        cout << "Length of list1: " << list1.length() << endl;


        cout << "Enter numbers, for list2, ending with -999" << endl;

        cin >> num;

        while(num != -999)
        {
          list2.insert(num);
          cin >> num;
        }
```

```
        cout << endl;

        cout << "list2: ";
        list2.print();
        cout << endl;
        cout << "Length of list2: " << list2.length() << endl;

        newList.mergeLists(list1, list2);

        cout << "newList after merging lis1 and list2" << endl;
        newList.print();
        cout << endl;
        cout << "Length of the newList: " << newList.length() << endl;


        return 0;
}
```

Question 3: Recursion [6]

Write a recursive function that finds and returns the sum of elements of an `int` array. Also write a program to test your function.

Solution:

```
int sumArray(const int list[], int first, int length)
{
if (length-first == 1)
return list[first];
else
return list[first] + sumArray(list, first + 1, length);
}
```

Test Program:

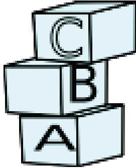
```
int main()
{
int list[10]={1,2,3,4,5,6,7,8,9,10};
int listB[1] = {5};
cout << "Sum list = " << sumArray(list,0,10) << endl;
cout << "Sum listB = " << sumArray(listB,0,1) << endl;
return 0;
}
```

Question 4: Stacks[6]

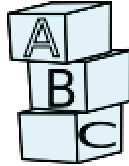
Write a program to transfer elements from stack S1 to stack S2. The order of the elements must remain as they were on the original stack S1. You may use one additional stack. Use the Stack class from the Standard Template Library.

Solution:

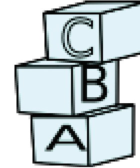
```
template < class Type >
void reverseStack( stack<Type> &S1 )
{
      stack<Type> temp ;

      while (!S1.empty())
      {
            temp.push(S1.top());
            S1.pop();
      }
      while (!temp.empty())
      {
            S2.push(temp.top());
            temp.pop();
      }
}
```



The original stack S1.



The temp stack after the first while loop. Notice that the elements are in reverse.



The contents of stack S2 after the second while loop. Notice that the elements are in required order.

Note:

- Drawing a diagram of what the question requires may help in understanding the problem.
- When elements are moved from one stack into another, their order is reversed. Observe how the helper (temp) stack is used. How would the helper stack be used if you were supposed to reverse stack S1? Look, if you return elements in S2 to S1, they will be reversed.

Question 5: Queues [6]

Write a function `reverseQ` that uses a local stack to reverse the contents of a queue. Use the following header:

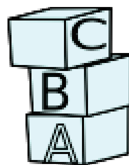
```
template < class object >
void reverseQ ( queueType < Object > &q )
```

You can make use of any member function of `class queueType`. Note that this function is not a member of `class queueType`.

Solution:

```
template < class Type >
void reverseQ ( queueType < Type > &q )
{
      stack<Type> s
      while (!q.isEmptyQueue())
      {
            s.push(q.front);
            q.deleteQueue();
      }
      while (!s.empty())
      {
            q.addQueue(s.top);
            s.pop();
      }
}
```

Original queue



The first while loop removes the front element (FIFO) from queue and places it into the stack. Finally, the last element of the queue is at the top (LIFO) of the stack.

Top element inserted in front of the queue reversing them in the process.

Note:

- Reversing elements in the queue requires a helper stack. No matter how many queues you use, the order of element never changes (unless you use one queue for to hold one element). Unlike the stack, the elements can be changed by using other stacks. It requires one only one queue to change the order of a stack, but it requires two additional stack to change its order.

Question 6: Searching [6]

Describe how the binary search algorithm searches for **27** in the following list:
5, 6, 8, 12, 15, 21, 25, 31.

Solution:

Position	0	1	2	3	4	5	6	7
Values	5	6	8	12	15	21	25	31

Binary search algorithm searches for elements in a sorted list. It compares the search element with the middle element in the list, if the search element is equal to the middle element, the position of this element is returned. Otherwise, the lower left or the upper right of the middle element is searched.

Position	0	1	2	3	4	5	6	7
Values	5	6	8	12	15	21	25	31

First = 0 Mid=3 last=7

In the above list, 27 is compared to 12. Surely, 27 should be on the upper right of the list. The new search list is shaded below.

Position	0	1	2	3	4	5	6	7
Values	5	6	8	12	15	21	25	31

First = 4 Mid= 5 last=7

Again, compare 27 with 21. 27 must be on upper right.

Position	0	1	2	3	4	5	6	7
Values	5	6	8	12	15	21	25	31

First = 6
Mid= 6 last=7

27 should be on the right of 25.

Position	0	1	2	3	4	5	6	7
Values	5	6	8	12	15	21	25	31

First = 7
Mid=7
last=7

Lastly, compare 27 and 31, but now 27 should on the left of 31. Our new search list is:

Position	0	1	2	3	4	5	6	7
Values	5	6	8	12	15	21	25	31

Last= 6 First = 7

This end the search because **first** if greater than **last**. 27 is not in the list.

Question 7: Sorting[20]

a) Selection sort.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
11	8	9	4	2	5	3	12	6	10	7
2	8	9	4	11	5	3	12	6	10	7
2	3	9	4	11	5	8	12	6	10	7
2	3	4	9	11	5	8	12	6	10	7
2	3	4	5	11	9	8	12	6	10	7
2	3	4	5	6	9	8	12	11	10	7
2	3	4	5	6	7	8	12	11	10	9
2	3	4	5	6	7	8	12	11	10	9
2	3	4	5	6	7	8	9	11	10	12
2	3	4	5	6	7	8	9	10	11	12
2	3	4	5	6	7	8	9	10	11	12
2	3	4	5	6	7	8	9	10	11	12

The shaded area is **sorted**

- Find the **smallest** element in the unsorted list.
- Move it to the **beginning** of unsorted list.

b) Insertion sort

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
11	8	9	4	2	5	3	12	6	10	7
8	11	9	4	2	5	3	12	6	10	7
8	9	11	4	2	5	3	12	6	10	7
4	8	9	11	2	5	3	12	6	10	7
2	4	8	9	11	5	3	12	6	10	7
2	4	5	8	9	11	3	12	6	10	7
2	3	4	5	8	9	11	12	6	10	7
2	3	4	5	6	8	9	11	12	10	7
2	3	4	5	6	8	9	10	11	12	7
2	3	4	5	6	7	8	9	10	11	12

The shaded area is **sorted**

- Find the first **unsorted** element in the unsorted list.
- Move it to the proper place.

c) Quick sort

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
11	8	9	4	2	**5**	3	12	6	10	7
3	**4**	2	5	9	11	8	12	6	10	7
2	3	4	5	9	11	8	12	6	10	7
2	3	4	5	9	11	8	**12**	6	10	7
2	3	4	5	7	11	**8**	9	6	10	12
2	3	4	5	**6**	7	8	9	11	10	12
2	3	4	5	6	7	8	9	**11**	10	12
2	3	4	5	6	7	8	**10**	9	11	12
2	3	4	5	6	7	8	9	10	11	12

The shaded area is **sorted**

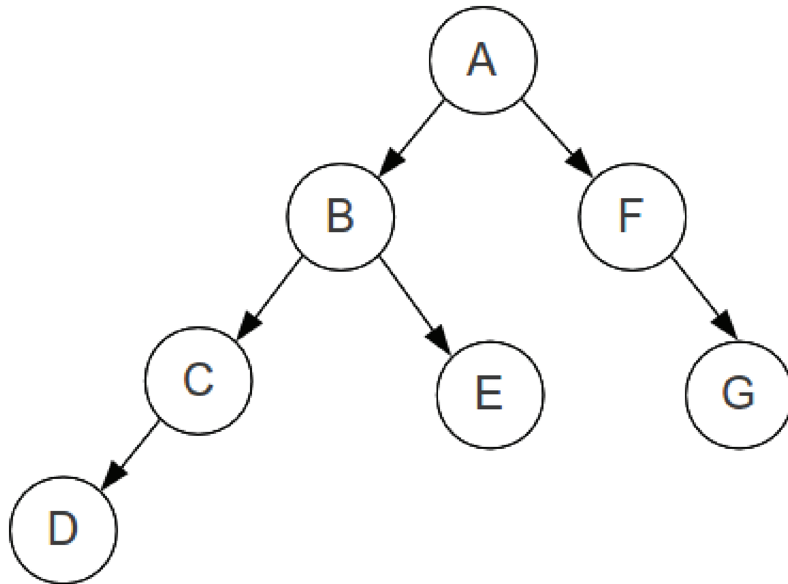
- Quick sort partition the list into two.
- Sort the lower sublist, then the upper sublist.

In the first row, the pivot selected is 5, using this pivot the list is sorted with elements smaller than the pivot on the left and elements greater than the pivot on the right after the pivot has been swapped to its final position 3. Note that this position (of the pivot) marks the final sorted position of the element. Also note that all elements on the left are

less than the pivot and on right are greater than pivot in all iterations.

The list on the left of the previous pivot (position 0-2) is sorted next. The pivot is 4 and the list is sorted as previously. Repeat until all elements on the left are sorted, then sort the elements on the right.

Question 8: Tree [6]



Inorder: DCBEAFG
Inorder traversal:- Traverse the left subtree
Visit the node
Traverse r the right subtree

Preorder: ABCDEFG
Preorder traversal:- Visit the node
Traverse the left subtree
Traverse r the right subtree

Postorder: DCEBGFA
Postorder traversal:- Traverse the left subtree
Traverse r the right subtree
Visit the node

UNISA©2012