National University of Computer & Emerging Sciences, Karachi

Fall-2018 CS-Department

Midterm I

October 01, 2018 Slot: 9AM – 10AM

Course Code: CS201	**Course Name: Data Structures**
Instructor Name:	**Muhammad Rafi / Nida Pervaiz / Teerath Das**
Student Roll No:	**Section No:**

- Return the question paper.
- Read each question completely before answering it. There are **3 questions and 2 pages.**
- In case of any ambiguity, you may make assumption. But your assumption should not contradict with any statement in the question paper.
- All the answers must be solved according to the sequence given in the question paper.
- Be specific, to the point while coding, logic should be properly commented, and illustrate with diagram where necessary.

Time: 60 minutes. **Max Marks**: 50 points

Object Oriented Programming

Question No. 1 **[Time: 10 Min] [Marks: 10]**

Consider the class definition provide below:

```
class CharBuffer{
   private:
     char *Buffer;
     int size;
   public:
     CharBuffer();
     CharBuffer(int s){
          Size=s;
          Buffer=new char[size];
     }
     CharBuffer(constCharBuffer & rhs){
          Size=rhs.size;
          Buffer=new char[size];
          memcpy(Buffer,rhs.Buffer, sizeof(char)*Size);
     }
     CharBuffer* operator=(constCharBuffer & rhs)
     {
          if (this != &rhs){
               delete[] Buffer;
               Size=rhs.size;
               Buffer=new char[size];
               memcpy(Buffer,rhs.Buffer, sizeof(char)*Size);
          }
          return *this;
     }
     ~CharBuffer(){
          if(Buffer) delete [] Buffer;
     }
 };
```

You are required to provide the implementation details for the functions comments with the question mark (?).

Recursion & DynamicSafeArrays	
Question No. 2	**[Time: 25 Min] [Marks: 20]**

a. Recursion has a number of disadvantages if we compare it with iterative approach. Give at least two of these. Is there any advantage of using recursion? [5]

Recursion is quite resource hungry as it may consume large stack and processor cycles for a deep recursive call. At the same time, it is also difficult to debug and trace for code maintenance, change and upgrade. With all these disadvantages, still recursion logically convincible for a complex problem which can easily be defined through a recursive definition.

b. Write a recursive function that returns the sum of the digits of a positive integer. For example: SumOfDigits(int x) when x=123 will return 1+2+3=6. [10]

```
int SumOfDigits(int x)
{
    if (x == 0)
       return 0;
    return (x % 10 + SumOfDigits(n / 10));
}
```

c. Write a function that takes two instances of DynamicSafeArrays<int> – (one dimension- with a single integer pointer) and decide whether the two arrays are disjoint or not. The disjoint means there is no element common in the two arrays. An efficient approach is required for full marks. [5]

```
bool Disjoint(DynamicSafeArray<int> &A, DynamicSafeArray<int> &B)
{
    int SizeA, SizeB;
    SizeA= A.getSize();
    SizeB= B.getSize();

     for (int i=0; i<SizeA; i++)
      for (int j=0; j<SizeB; j++)
         if (A[i] == B[j]) // any common
            return false;

    // If no element of A is present in B
    return true;
}
```

// there are other efficient approaches as well…..

SinglyLinkedList, DoublyLinkedList

Question No. 3 **[Time: 25 Min] [Marks: 20]**

a. A palindrome is a word, number, or other sequence of characters which reads the same backward as forward, such as madam or racecar. Given an instance of SinglyLinkedList<char>, for example List<char> as below:

Head -- > $\boxed{\text{M}}$ => $\boxed{\text{A}}$ => $\boxed{\text{D}}$ => $\boxed{\text{A}}$ => $\boxed{\text{M}}$ => $\boxed{\textbf{\textit{null}}}$

You need to write a function that decide whether the given instance of the singly linked list of char forms a palindrome or not. [10]

Assuming there are three functions available as we discussed during the lecture:

SinglyLinkedList<T> * SplitHalf (SinglyLinkedList<T> & rhs);

void Reverse(SinglyLinkedList<T> & rhs);

bool CompareTwoLists(SinglyLinkedList<T> &FirstHalf, SinglyLinkedList<T> &SecondHalfReverse);

The approach is very simple, split the list in two halves, reverse the second half and run a compare on the two halves portions of the lists, if they are identical the original list is a palindrome.

There are other approaches as well.

b. Write a function for DoublyLinkedList<T> which decide if the two pointers to the nodes of the given list are adjacent or not? [5]

```
Head -> 1000
Tail -> 4000
```

1000	2000	3000	4000
A	B	C	D
2000	3000	4000	nil
nil	1000	2000	3000

```
bool IsAdjacent(DNode<T> * left, DNode<T> * right){

return (left->nextPtr == right && right->prevPtr == left ) ||
( right->nextPtr == left && left->prevPtr == right );

}
```

c. Given the DNode<T> pointers to two adjacent nodes of a doubly linked list say, left and right, which are not from the two extremes of the list. Write a function that swaps the two given nodes. [5]

```
Head -> 1000
Tail -> 4000
```

1000	2000	3000	4000
A	B	C	D
2000	3000	4000	nil
nil	1000	2000	3000

```
void SwapAdjacentMiddleDNodes(DNode<T> * left, DNode<T> *
right){

DNode<T> *tempLeft,*tempRight;

tempLeft = left->prevPtr;
tempRight = right->nextPtr;
tempLeft->nextPtr = right;
right->prevPtr=tempLeft;
left->nextPtr=tempRight;
tempRight->prevPtr=left;
right->nextPtr=left;
left->prevPtr=right;

}
// code works for any pair of non-extreme nodes.
```