National University of Computer & Emerging Sciences, Karachi
Fall-2021 CS-Department
Midterm I
October 11, 2021 Slot: 11 AM – 12 PM

Course Code: CS 2001	**Course Name: Data Structures**
Instructor Name: Muhammad Rafi / Dr. Ali Raza/Shahbaz/ M Sohail/ Mubashra Fayyaz	
Student Roll No:	**Section No:**

- Return the question paper.
- Read each question completely before answering it. There are **4 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**: 40 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);
     CharBuffer(const CharBuffer & rhs);
     CharBuffer* operator=(const CharBuffer & rhs); //?
     ~CharBuffer(); // ?
     void addBuffer(char * newBuffer); //?
     void removeBuffer(int n_size); // ?
};
```

You are required to provide the implementation details for the functions comments with the “//?” pattern. The function addBuffer add char stream at the right hand of the Buffer and removeBuffer remove n_size character from the left side of the Buffer.

```
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;
}

void addBuffer(char * newBuffer){
    int add_buffer_length = sizeof(newBuffer)/sizeof(char);
    nbuffer= new char(Size+add_buffer_length);
    memcpy(nbuffer,this->Buffer, sizeof(char)*Size);
    memcpy(nbuffer+Size,newBuffer, sizeof(char)*add_buffer_length);
    delete [] Buffer;
    Buffer=nbuffer;
}

void removeBuffer(int n_size){
    nbuffer= new char(Size-n_size);
    memcpy(nbuffer,this->Buffer+n_size, sizeof(char)*Size-n_size);
    delete [] Buffer;
    Buffer=nbuffer;
}
```

Recursion

Question No. 2 **[Time: 15 Min] [Marks: 10]**

a. What are the two important things one has to remember when designing a recursive solution? Is there any advantage of using recursion? [5]

There are two important aspect of a recursive solution (i) Identifying a base case for terminating the recursive calls. (ii) redefining the problem as sub-problems with changing paramters with progression towards base case. 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. [5]

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

Dynamic Safe Arrays & Variants

Question No. 3 **[Time: 15 Min] [Marks: 10]**

A two dimensional array of characters can be considered as a field. Each cell is either water 'W' or a tree 'T'. A forest is a collection of connected trees. Two trees are connected if they share a side i.e. if they are adjacent to each other with respect to any of the four sides. Given the information about the field, write a function which inputs this 2-D array and returns the size of the largest forest, where size of a forest is the number of trees in it. Please see the sample case for clarity:

```
void CountTreesMax(int i, int j, int N)
{
    if(i<0 || i>=N || j<0 || j>=N ||M[i][j] != 'T')
        return;
    count++;
    M[i][j]='*';
    CountTreesMax(i-1,j,N);
    CountTreesMax(i+1,j,N);
    CountTreesMax(i,j-1,N);
    CountTreesMax(i,j+1,N);
}

int main()
{
 for(int i=0;i<N;i++)
    {
        for(int j=0;j<N;j++)
        {
            if(M[i][j] == 'T')
            {
                count=0;
               CountTreesMax(i,j,N);
                if(max<count)
                    max=count;
            }
        }
    }
    cout<< max <<endl;
}
```

Linked List and Variants	
Question No. 4	**[Time: 15 Min] [Marks: 10]**

a. A SinglyLinkedList in its vanilla implementation contains a loop. Write a function which return a Boolean value(True) if the list contains a loop or (False) otherwise. [5]

```
bool LoopDetection(Node<T> * head)
{
  Node<T> * OneStep=head;
  Node<T> * TwoSteps= head;

  while ( OneStep && TwoSteps && TwoSteps->next)
  {
     OneStep = OneStep ->next;
     TwoSteps= TwoSteps ->next->next;
     if (oneStep == TwoSteps){
      return 1; // loop detected
     }
    } //while ends.
  return 0;
} // LoopDetection ends.
```

b. Write an efficient function that decides whether the list contains even number of nodes or odd number of nodes. [5]

```
int countNodes(Node<T> * head)
{
      if head == 0 return 0;
      else
      return (1+ countNodes(head->next));
     // count nodes recursive
}

bool isEvenNodeCount(Node<T> * head)
{
     int count = countNodes(head);
      return (count%2)==0;
     // return true if even numbered of nodes in the list
}
```

Note: from the OneStep and TwoSteps approach we can decide even and odd numbers of nodes in a linkedlist in an efficient manner.

<The End.>