National University of Computer & Emerging Sciences, Karachi

Fall-2019 CS-Department

Midterm I

September 23, 2019 Slot: 1 PM – 2 PM

Course Code: CS 218	**Course Name: Data Structures**
Instructor Name:	**Muhammad Rafi / Nida Pervaiz**
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: 15 Min] [Marks: 15]**

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=rhs.buffer;
}
CharBuffer*
operator=(constCharBuffer & 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 fillArray()
{
     for (int i=0; i<cols;
i++){
          int val; cin>>val;
          buffer[i] = val;
               }
}};
int main(){
   CharBuffer ob1(5);
   ob1.fillArray;
   CharBuffer ob2(ob1);
   CharBuffer ob3(7);
   ob3.fillArray;
   ob2=ob3
}
```

a) Analyze each function and point-out any issue you find in it? [5]

- Copy Constructor – this copy constructor is using shallow copy and this could lead to memory leaks and program crash.
- Assignment Operator – this implementation is not checking self-reference
- Destructor – In the destructor the memory should be release in the reverse order of memory acquires in constructor, in the constructor an array based memory is acquired and destructor is using non array delete.

b) What will exactly happen when after filling array of object ob3 we call out ob2=ob3? [5]

-obj2 is created by using object obj1 as a copy. The copy constructor is not doing any deep copy hence a shared character buffer is used by the two objects obj1 and obj2. When the instruction obj2=obj3 is executed the errors in assignment operator delete the memory of obj2. Hence obj1 memory would also be deleted.

c) Write the code after resolving all issues that you find in part (a) of this question. [5]

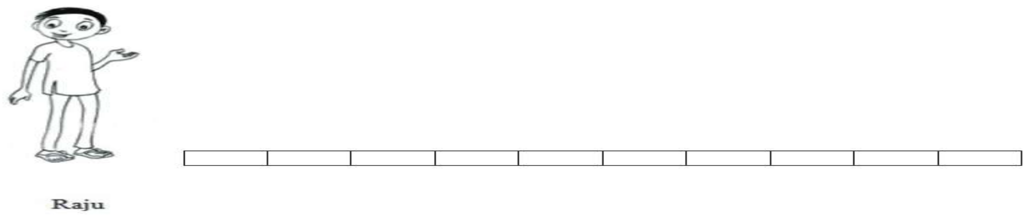
```
//Copy Constructor
CharBuffer(constCharBuffer & rhs){
      size=rhs.size;
      buffer = new char[size];
      memcpy(buffer, rhs.buffer, sizeof(char)*size);
}
```

```
//Assignment Operator
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;
}
```

```
//Destructor
~CharBuffer(){
      if(buffer)
          delete[]  buffer;
}
```

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

Raju is in wonderland and he has got a problem. He needs to estimate the number of possible ways in which he can escape a linear (1D) maze in front of him. He can move one step, two steps or four steps at a time(cells). The size of maze is Max; given this information you need to write a recursive routine to check in how many different ways he can escape this maze. [10]



You need to identify the base case(s), recurring case(s) and required to provide code for the recursive program for it.

```
int CountPaths(int n, int TotalPaths){
      if ((n==0) || (n==1)) {
      return TotalPaths + 1;}
      else if (n==2){
      return (TotalPaths+2);}
      else if (n==3){
      return CountPaths(n-2,TotalPaths)+CountPaths(n-1,TotalPaths);
      }
      else {
            return CountPaths(n-4, TotalPaths)+
            CountPaths(n-2,TotalPaths)+CountPaths(n-1, TotalPaths);
      }
}
```

Dynamic Safe Arrays	
Question No. 3	**[Time: 30 Min] [Marks: 25]**

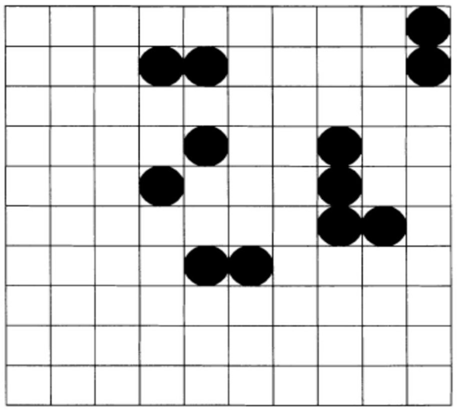
A. Given a 1d dynamic safe array, code the function to circularly shift the array contents backward to a k value which will be given by the user as input together with the array contents. [10]

Example: Input → K=3 and Array={1,2,3,4,5,6} Output→ {4,5,6,1,2,3}

```
// function
void leftRotateK(int arr[], int n, int k)
{   if (k > n) k= k % n;
    for (int i=0, j = 0; j < k; j++)
    {
        int temp = arr[0];
        for ( i = 0; i < n - 1; i++)
        {
          arr[i] = arr[i + 1];
        }
        arr[i] = temp;

    }
}
```

B. Consider a 2D grid of size N x N (as an 2D Arrays of Boolean values 1 mean square is black, 0 means white) given below:



In this grid two squares belong to the same group if they share a common edge. In the given grid. There is one group of four occupied squares, three groups of two occupied squares, and two individual occupied squares.

a. Computes the size of a group when a square in the group is given.

b. Computes number of groups of all sizes. [15]

```
//a. Computes the size of a group when a square in the group is given.

      int countNumberOfGroups(int* groupToGroupCount){
          int count = 0;
          for (int i = 1; i <= SIZE * SIZE; ++i) {
              if ( *(groupToGroupCount+i) > 0 )
                  count++;
          }
          return count;
      }

      int connectedCells(int **grid, int i, int j, int size) {
          if (i < 0 || j < 0 || i >= size || j >= size || grid[i][j] != 1)
      return 0;

          int count = 0;
          if (grid[i][j] == 1) count++;

          grid[i][j] = 2;     //mark that its visited no its not be counted
      multiple times

          count += connectedCells(grid, i+1, j, size);    //move down and
      count the connected pairs
          count += connectedCells(grid, i, j+1, size);    //move right and
      count the connected pairs

          return count;
      }

      int* findAllTheConnectedCells(int** grid, int gridSize){
          int* groupToGroupCount= new int[gridSize*gridSize + 1];     //max
      possible group size

          for (int i = 0; i <= gridSize * gridSize; ++i)
              groupToGroupCount[i] = 0;

          for (int row = 0; row < gridSize; ++row) {
              for (int col = 0; col < gridSize; ++col) {

      groupToGroupCount[connectedCells(grid,row,col,gridSize)]++;
              }
          }
          return groupToGroupCount;
      }
      int groupCountOfSize(int* groupToGroupCount, int groupSize){
          return groupSize <= SIZE*SIZE ? *(groupToGroupCount+groupSize) :
      0;
      }

      void printAllGroupsAndSizes(int* groupToGroupCount){
          cout << "All Group Size and Quantity are as following: " << endl;
          for (int i = 1; i <= SIZE * SIZE; ++i) {
              if ( *(groupToGroupCount+i) > 0 )
                  cout << "Group Size: " << i << " Number of Groups: " <<
      *(groupToGroupCount+i) << endl;
          }
          cout << "----------------------------------------------" << endl;
}



// Code is developed by Mehdi K163904
```