National University of Computer and Emerging Sciences, Lahore Campus

Course:	Object Oriented Programming	Course Code:	CS 217
Program:	BS(Computer Science)	Semester:	Spring 2019
Duration:	180 Minutes	Total Marks:	75
Paper Date:	14-May-2019	Page(s):	12
Section:	ALL	Section:	
Exam:	Final	Roll No:	

Instruction/Notes: Use the provided space to solve the questions.
Extra sheets may be used, but may not be attached.
In case of any ambiguity make a reasonable assumption.

Question # 1 (20 points)

We have designed a game called treasure hunt. In this game a 2D board contains different types of treasures. The treasures are dynamically allocated objects of type **Treasure** (see code on the right). So each board square contains a pointer to an allocated Treasure object.

A dice is rolled for each player at the same time and the players move on the board according to the number that turns up on their dice. A player collects the treasure on the square on which he stops, by adding the pointer of that treasure into his bag (see code on the right). The bags are initially empty. Since more than one player can land on the same square, various players may have pointers to the same objects in their bags. Also, pointers to these objects remain in the board. The board and the bags share pointers to the same Treasure objects, without creating new copies of these objects.

Our code contains a class called **TreasureHunt** (see code on the right) which contains the board (containing pointers to different Treasure objects), its dimensions, and a list of players.

Each player contains a bag, which is an array of Treasure pointers, which is updated with each move of the player on the board.

For example, assume that the game has three players.

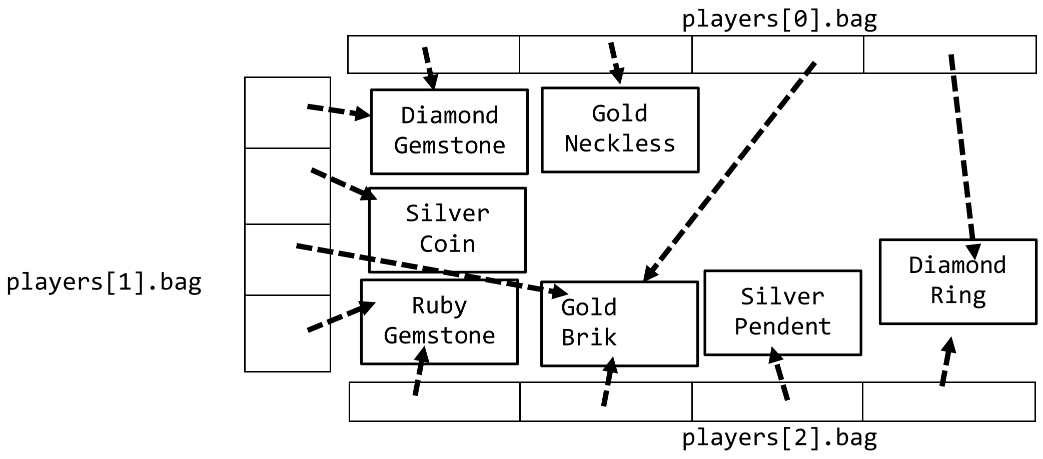
Assume we have the following 4×3 board at the start of game (each square contains a pointer to a treasure object whose description is shown.)

Diamond Gemstone	Bronze Sword	Diamond Ring
Silver Coin	Gold Neckless	Sapphire Gemstone
Gold Brik	Ruby Gemstone	Silver Pendent
Bag of gemstones	Gold Coin	Silver Brick

```
class Player
{
 private:
        Treasure ** bag;
        int bagsize;
        int totalpoints;
        string name;
 public:
      //Player methods
};
class TreaureHunt
{
private:
        Treasure *** board;
        int rows;
        int cols;
        Player ** players;
        int playerCount;
public:
     //treasureHunt methods
};

struct Treasure{
   int value;
   string description;
}
```

At the end of the game, the players' bags may contain the pointers as follows (the actual situation will depend on the numbers that turn up on each dice):



You job is only to write a destructor for the class TreasureHunt to deallocate all the allocated memory of the game. Make sure that no object is deleted more than once and there are no memory leaks. You can add more code utility methods to the class TreasureHunt to divide the work among functions.

Question # 2 (20 points)

In this question you will need to use inheritance, polymorphism, friend class and down casting.

You have designed a new music player application. Let's call it OOPMP, short for OOP Music Player. OOPMP will have a playlist in which songs can be added or removed.
Each song will have lyrics and name. As we cannot keep audio for lyrics, let's assume that the lyrics are in the form of a char*.
Songs will have further types as follow:

1. MP3: this will be an audio song, only containing lyrics, each MP3 song will also have a type (for example pop, rock, classical etc.)
2. MP4: this will be a video song, they will have lyrics as well as video (again, as we cannot keep video we will assume it to be a char* containing a description of the scenes.)
3. sMP4: short for ***secret*** MP4, this type of song will also have lyrics and video. It is called secret MP4 because only OOP music players can play their real lyrics and video. If anyone else tries to play it, they will only hear *bleep* as lyrics and see *glitches* as videos.

Design and code your classes keeping in mind the above mentioned instructions and the following main function and its desired output (on next page).

```
int main()
{
       /*Creating MP3 song, arguments are
       name of song, lyrics and type*/
       MP3 *s1= new MP3("Happy",
              "Because I'm happy...",
              "pop") ;
       //play s1, it will just print the lyrics
       s1->play();

       /*Creating MP4 song,
       arguments are name of song,
       lyrics and description of video*/
       MP4 *s2= new MP4("Happy",
              "Because I'm happy...",
              "A man walking..");
       s2->play();

       /*Creating sMP4 song, arguments are
       name of song, lyrics and type*/
       sMP4 *s3= new sMP4("Born Free",
              "Born free...",
              "Different pictures\nappearing on screen...");
       s3->play();   /*note that playing song of sMP4
                     does not show real lyrics and video*/

       //Creating a music player of type OOPMP
       OOPMP myPlayer(3);

       //Adding songs to OOPMP
       myPlayer.addToPlayList(s1);
       myPlayer.addToPlayList(s2);
       myPlayer.addToPlayList(s3);

       /*invoking memeber function to play all songs in playlist
       note that now sMP4 song shows the real lyrics and video*/
       myPlayer.playAll();
       }
```

```
TYPE:
pop
LYRICS:
Because I'm happy...

LYRICS:
Because I'm happy...
VIDEO:
A man walking..

LYRICS:
bleeeeeeep
VIDEO:
glitchhhh

Currently Playing: Happy
TYPE:
pop
LYRICS:
Because I'm happy...

Currently Playing: Happy
LYRICS:
Because I'm happy...
VIDEO:
A man walking..

Currently   Playing:   Born
Free
LYRICS:
Born free...
VIDEO:
Different pictures
appearing on screen...
```

Question # 3 (20 points)

The Sieve of Eratosthenes is an algorithm to find all the prime numbers between 2 and an input number n. For example, if n=23, the algorithm finds 2, 3, 5, 7, 11, 13, 17, 19 and 23. The algorithm works as follows:

(1) Create an array, A, containing all numbers between 2 and n.
(2) Start with p=2
(3) Remove all multiples of p from A (excluding p itself), since these multiples cannot be prime.
(4) Set p = the next number in A after all the removals in step(3)
(5) If p<n/2, repeat from (3).

Write a function called Eratosthenes which returns a dynamic, int*, array, containing all the prime numbers between two given numbers *m* and *n*. The size of the array should be exactly one more than the total number of prime numbers. The function should place a **-1** at the end of the array to mark the end of the output.

Question # 4 (3*5 = 15 points)

(a) Write the output of code segment given below for following inputs.

```
void main()
{
      double ** arr;
      int size;
      try
      {
            cout << "Enter Size of array: ";
            cin >> size;
            if (size <= 0)
            throw exception("Array size must be greater than zero \n");
            else if (size == 0)
                  throw size;
            arr = new double*[size];
            for (int i = 0; i < size;i++){
                  arr[i] = new double[size];
            }
      }
      catch (bad_alloc ba){
            cout << ba.what()<< endl;
      }
      catch (int s){
            cout << "Wrong size: " << s << endl;
      }
      catch (out_of_range o){
            cout << o.what() << endl;
      }
      catch (exception e){
            cout << e.what() << endl;
      }
}
```

Size = -1
Size = 0
Size = 50000

(b) Write the output of code segment below.

```
class strwrap{
            string s;
   public:
            strwrap(const char * str){
                  s=str;
            }
            void whatprint1(){
                        (this+2)->whatprint2();
            }
            void whatprint2(){
                        (this-1)->print();
            }
            void print(){
                        cout<<s<<endl;
            }

      };

     int main() {
         strwrap arr[3]={"What will", "be printed", "on the screen?"};
         arr[0].whatprint1();
     }
```

Output

(c) Consider the following function template ***sort*** that takes an array of elements as input, sorts and prints it. ***main1*** function gives an example of how it is used for integer array and its output. Identify if this function will work for an array of char* type, if not write a code such that the ***main2*** function works and produces desired output as shown below, i.e. sorts the strings in given array in ascending order of their length and prints them.

```
void sort(T* arr, int n)
{
      int i, j;
      for (i = 0; i < n; ++i)
      {
            for (j = 0; j < n-i-1; ++j)
            {
             /*Comparing consecutive data
            switching values if value
            at j > j+1*/
                  if (arr[j] > arr[j+1])
                  {
                        T temp= arr[j];
                        arr[j]= arr[j+1];
                        arr[j+1]= temp;
                  }
            }
      }
      for (int i=0; i<n; i++)
            cout<<arr[i]<<endl;
}
```

```
//main1
void main ()
{
   int arr[5]={89,10154,687,435,8054};
   sort(arr, 5);

}
```

<u>**output**</u>

```
89
435
687
8054
10154
```

```
//main2
void main2()
{

   char **cArray= new char*[3];
   cArray[0]= new char[3];
   strcpy(cArray[0],"xa");
   cArray[1]= new char[4];
   strcpy(cArray[1],"abc");
   cArray[2]= new char[5];
   strcpy(cArray[2],"mnop");
   sort(cArray, 3);
}
```

<u>**Desired output**</u>

```
xa
abc
mnop
```

Space for part (c)