SEAT NO. CT-23025

NED UNIVERSITY OF ENGINEERING & TECHNOLOGY

FIRST YEAR(Computer Science)

SPRING SEMESTER EXAMINATIONS 2024

Time : 3 Hours | **Batch 2023** | **Dated : 19-JUL-24** | **Max Marks : 60**

Object Oriented Programming - CT-260

Q1 [Marks 12, CLO1]. **Identify** any errors in the given program. If there are no errors, **determine** the output. Assume all necessary libraries and namespaces are imported for the programs to run.

a.
```
int main() {
vector<int> v; vector<int> w(11);
v.push_back(1); v.push_back(2);
v.push_back(3);
w.push_back(1); w.push_back(2);
w.push_back(3);
cout << "Size of v: " << v.size() << endl;
cout << "Size of w: " << w.size() << endl; }
```

b.
```
class MathUtility {
public:
static int divide(int a, int b) {
if (b == 0) {
   cout<< "Division by zero error"; exit(1)}
return a / b;   } };
int main() { int result;
cout << MathUtility::divide(10, 2) << endl;
cout << MathUtility::divide(8, 0) << endl; }
```

c.
```
int main() {
string line;
ofstream objectFile("text.txt");
if (!objectFile) {
   cerr << "Error opening file!" << endl;
   return 1;   }
objectFile << "Good Luck!" << endl;
objectFile >> line;
objectFile.close(); }
```

d.
```
string processString(string msg) {
cout << "Original Message: " << msg;
int index = 0;
while (msg[index]!='\0') {
     msg[index]+=2;   index++;   }
  return msg;}
int main() {
string line = processString("OOPEXAM");
cout<<"Encrypted Message: "line<<endl;   }
```

QQRGZCO

e.
```
class Vehicle {
public:
Vehicle() {
  cout << "Vehicle constructor called." << endl; }
virtual ~Vehicle() {
  cout << "Vehicle destructor called." << endl;  }
virtual void display() const {
   cout << "Displaying Vehicle." << endl; } };
class Car : public Vehicle {
public:
Car() {cout << "Car constructor called." << endl; }
~Car(){cout << "Car destructor called." << endl;  }
 void display() const {
     cout << "Displaying Car." << endl; } };
int main() {
   Vehicle* vehiclePtr = new Car();
   vehiclePtr->display();
   delete vehiclePtr;  }
```

f.
```
class SquareArea{
float sideLength;
public:
SquareArea(float sideLength=0) :
sideLength(sideLength) { }
SquareArea() : sideLength(0){ }
void setLength(float f){
        sideLength=f; }
float calculateArea(){
return sideLength*sideLength; } };
int main() {
SquareArea ob;
ob.setLength(12);
cout<<"Area is "<<ob.calculateArea();}
```

Q2 [Marks 18, CLO1]. Answer the following short questions. Please be specific.

a. **Describe** how can we read and write any data record (object of a class) in a file.
b. **Differentiate** between abstract and concrete classes by giving suitable examples.
c. Does friend function violate the rule of encapsulation? **Explain** your answer with a suitable example.
d. A program can face any unexpected error at runtime due to any unforeseen circumstances. **Explain** how such unexpected exceptions can be handled in C++.
e. **Explain** what the default assignment operator = does when applied to objects.
f. In composition, can the functions of a class directly access the data members of the object it contains as a data member? **Explain** your answer.

P.T.O

Q3 [Marks 10, CLO2]. **Construct** a **template function** that calculates and returns median of an array passed as a parameter to it. **Median calculation formulae:** For a sorted array: If the total number of terms (n) is an odd number, then the formula is given below:

$$Median = \left(\frac{n+1}{2}\right) th\ term$$

If the total number of terms (n) is an even number, then the formula is given below:

$$Median = \left(\frac{\frac{n}{2} th\ term + \frac{n+1}{2} th\ term}{2}\right)$$

Note: The array must be sorted in ascending order first to apply the formula. Write a test program which calls the template function by passing first passing an int array and then a float array.

Q4 [Marks 10, CLO2]. You are required to **design** a UML class diagram and **construct** a solution for the following scenario in C++. In this scenario, we have the following classes:

1. **Shape**: A abstract base class with basic properties.
2. **ColoredShape**: A derived class from Shape that adds color information.
3. **TexturedShape**: Another derived class from Shape that adds texture information.
4. **ColoredTexturedShape**: A derived class from both ColoredShape and TexturedShape, representing a shape that has both color and texture.

Requirements:

1. **Shape**: It contains a method draw() that outputs "Drawing a shape", an attribute id to uniquely identify the shape.
2. **ColoredShape**: It inherits from Shape, adds an attribute color, overrides the draw() method to output "Drawing a colored shape".
3. **TexturedShape**: It Inherits from Shape, adds an attribute texture, overrides the draw() method to output "Drawing a textured shape".
4. **ColoredTexturedShape**: It inherits from both ColoredShape and TexturedShape, overrides the draw() method to output "Drawing a colored and textured shape".

Create objects of each class and demonstrate the correct usage of the draw() method. Ensure the id attribute from the Shape class is indirectly accessible and automatically initialized for each object.

Q5 [Marks 10, CLO2] : **Construct** a String class in C++ that manages a dynamic character array (char*). The class should include essential functionalities such as string initialization, copying, concatenation using + operator, character access with bounds checking, comparison using == operator, and display. Additionally, incorporate exception handling to manage out-of-bounds access errors. Use the following UML class for constructing the class.