LAB-III EXAM

MCA
(Semester - II)

Name: Arzoo Khan

—

Roll No: 20MCA011

—

Paper Code: CSC26

—

Paper Title: Lab-III (OOP)

—

Date: 04-08-2021

—

Time: 10:00 AM – 12:00 PM

1) **5. Check whether a given number is prime or not by overloading '!' operator.**

SOURCE CODE:

```
#include <iostream>
#include <stdbool.h>

class Integer
{
  int num;

public:
  // Member function to get input from the user for a Integer class object
  void input()
  {
    std ::cout << "\nEnter a number: ";
    std ::cin >> num;
  }

  //Member function to display the Integer Class object
  void display()
  {
    std ::cout << num;
  }

  // Member function ('!' operator overloading) to check
  // whether a number is prime or not
  bool operator!()
  {
    for (int i = 2; i <= num / 2; i++)
    {
      if (num % i == 0)
      {
        return false;
      }
    }
    return true;
  }
};

//Driver Code
int main(void)
```

```
{
  Integer x;
  char ex;

  do
  {
    x.input();
    x.display();

    if (!x)
      std ::cout << " is prime";
    else
      std ::cout << " is not prime";

    std ::cout << "\n\nExit? ";
    std ::cin >> ex;
  } while (ex != 'y' && ex != 'Y');

  return 0;
}
```

OUTPUT:

```
OOP_Lab_III_Exam.cpp - Arzoo
OUTPUT   DEBUG CONSOLE   PROBLEMS   TERMINAL                    Code

Enter a number: 31
31 is prime

Exit? n

Enter a number: 45
45 is not prime

Exit? n

Enter a number: 61
61 is prime

Exit? n

Enter a number: 97
97 is prime

Exit? y
```

2) 20. Consider the following class definition

```
class father
{
   protected :
    int age;

   public:
    father (int x) {age = x;}
    virtual void iam ( )
    {
        cout < < “I AM THE FATHER, my age is : ”<< age<< end1;
    }
};
```

Derive the two classes son and daughter from the above class and for each, define iam( ) to write our similar but appropriate messages. You should also define suitable constructors for these classes. Now, write a main ( ) that creates objects of the three classes and then calls iam( ) for them. Declare a pointer to class father. Successively, assign addresses of objects of the two derived classes to this pointer and in each case, call iam ( ) through the pointer to demonstrate the run-time polymorphism.

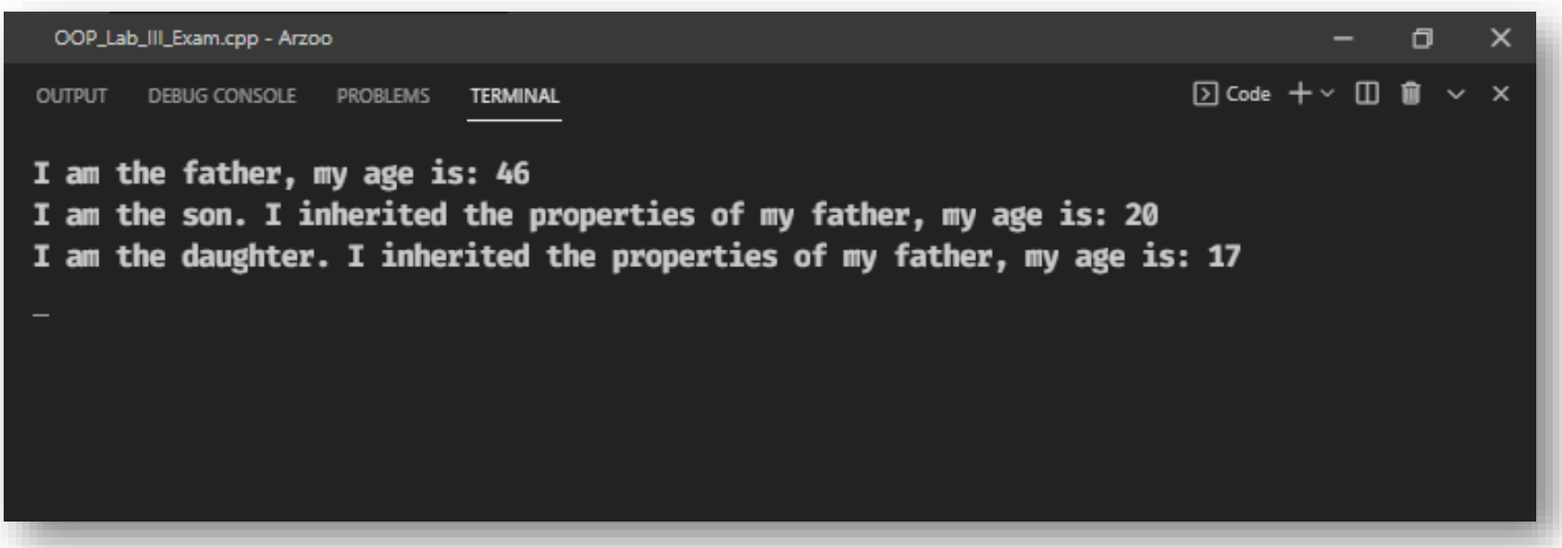
SOURCE CODE:

```
#include <iostream>
using namespace std;

// Base Class
class Father
{
protected:
  int age;

public:
  Father() {} // Default Constructor

  Father(int x) // Parameterized Constructor
  {
    age = x;
  }
```

```
  virtual void iam()
  {
    cout << "I am the father, my age is : " << age << endl;
  }
};

// Derived Class
class Son : public Father
{
public:
  Son() {} // Default Constructor

  Son(int x) // Parameterized Constructor
  {
    age = x;
  }

  void iam()
  {
    cout << "I am the son. I inherited the properties of my father, "
         << "my age is : " << age << endl;
  }
};

// Derived Class
class Daughter : public Father
{
public:
  Daughter() {} // Default Constructor

  Daughter(int x) // Parameterized Constructor
  {
    age = x;
  }

  void iam()
  {
    cout << "I am the daughter. I inherited the properties of my father, "
         << "my age is : " << age << endl;

  }
```

```
};

// Driver Code
int main()
{
  Father *father_ptr[3]; // Array of pointers of base class (Father class)

  // Base class (Father) pointer pointing to same class object
  father_ptr[0] = new Father(46);

  // base class (Father) pointer pointing to a derived class (Son) object
  father_ptr[1] = new Son(20);

  // base class (Father) pointer pointing to a derived class (Daughter) object
  father_ptr[2] = new Daughter(17);

  father_ptr[0]->iam();  // Late Binding (run-time polymorphism)
  father_ptr[1]->iam();  // Late Binding (run-time polymorphism)
  father_ptr[2]->iam();  // Late Binding (run-time polymorphism)

  return 0;
}
```

OUTPUT:



*****