NIT Agartala

End-term Examination

Subject : Object Oriented Programming
Paper Code : PCA02C11
Name : Tribhuwan Singh Bisht
Enrollment No. : 20MCA006
Branch : Computer Science & Engineering
Semester : IInd
Date : 09-07-21

1. (a) Is it mandatory to declare a constructor in a class?

(a) It can be used to initialize the objects to desired values or default values at the time of creation of an object.

It is not mandatory for the coder to write a constructor for a class. If no user-defined constructor is provided for a class, compiler initializes member variables to its default values.

1.(b) Write a C++ program to swap the values of two integers using user defined function with return type & with argument.

(b)

```
#include <iostream>
using namespace std;
int swap(int &x, int &y)
  { int temp = x;
    x = y;
    y = temp;
    return 0;
  }
int main()
  {
    int a=100
    int b=200
  cout << "Before swap, value of a:" <<a<<endl;
  cout << "Before swap, value of b:" <<b<<endl;
   swap (a, b);
  cout<< "After swap, value of a:" <<a <<endl;
  cout<< "After swap, value of b:" <<b<<endl;
  return 0;
}
```

Output :

```
Before swap, value of a :100
Before swap , value of b :200
After swap , value of a :200
After swap , value of b :100
```

1.(c) Mention four benefits of object oriented Programming.

(c) Following are the benefits of object oriented programming :

(i) We can build the programs from standard working modules that communicate with one-another, rather than having to start writing the code from scratch which leads to saving of development time & higher productivity.

(ii) OOP language allows to break the program into the bit-sized problems that can be solved easily (one object at a time).

(iii) OOP systems can be easily upgraded from small to large systems.

(iv) It is possible that multiple instances of objects co-exist without any interference.

(v) The principle of data hiding helps the programmer to build secure programs which cannot be invaded by the code in other parts of the program.

3. (a) Define manipulator with suitable example.

(a) Manipulators are used to changing formatting parameters on streams & to insert or extract certain special characters.

Following are some of the most widely used C++ manipulators:

(i) endl:

This manipulator has the same functionality as '\n' (newline character). But this also flushes the output stream.

(ii) showpoint / noshowpoint:

This manipulator controls whether decimal point is always included in the floating-point representation.

(iii) setw:

This manipulator changes the width of the next input/output field. When used in an expression out << setw(n) or in >> setw(n), sets the width perimeter of the stream out or in to exactly n.

Ex:

```
int main()
{
  cout << "setw(6):" << setw(6) << 42 << '\n';
  return 0;
}
```

Output:

```
setw(6):    42
```

2.(b) What is copy constructor & what is its use?

(b) Copy constructor is used to declare & initialize an object from some already existing object of the class.

Copy constructor takes a reference to the object of same class as an argument. Copy constructor cannot accept argument by value.

Syntax:

```
class_name ( class_name & object_name)
 {
    ---- body of constructor ----
 }
```

Ex:

```
customer_id ( customer_id &c);
```

Copy constructor is used to:

(i) To Initialize an object using another object.

(ii) To copy an object to pass it as an argument by value to a function.

(iii) Also to copy the object to return it from a function by value.

2.(c) Create a class area to calculate & display the area of a rectangle, circle & square based on constructor overloading.

(c)

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

class area()
 {
   private:
    int l, b, s;
    float r;
```

```
public:
    area(int l, int b)
    {
        cout << "Area of rectangle is:" << l*b << endl;
    }
    area(int s)
    {
        cout << "Area of square is:" << s*s << end;
    }
    area(float r)
    {
        cout << "Area of circle is:" << π*r*r << end;
    }
};
int main()
{
    area a1(10,5), a2(5), a3(10.0);
    return 0;
}
```

Output :

```
Area of rectange is : 50
Area of square is : 25
Area of circle is : 314
```

3. (a) What is a destructor? Can it be overloaded?

(a) → Destructor functions are inverse of constructor functions. They are invoked automatically when objects are destroyed. An object gets destroyed when its scope of existence finishes or when the delete operator is used.

→ Destructors are functions with same name as class & they do not accept any parameters. Destructor is preceded by '~' (tilde).

→ Destructors do not have any return type & they cannot be inherited as well.

Syntax:

```
~ name_of_the_class()
{

}
```

→ Destructor cannot be overloaded. You can have only one destructor for a class.

3. (b) Distinguish between the following two statements:

time T2(T1);
time T2 = T1;

where T_1 & T_2 are the objects of time class.

(b) time T2(T1) → Here copy constructor is called

Copy constructor is called when a new object is created from an existing object, as a copy of the existing object.

time T2 = T1 → Here assignment operator is called

Assignment operator is called when an already initialized object is assigned a new value from another existing object.

3. (c) Create a class bank_account with the data members: name of account holder, account type & account balance. Also create a member function: deposit() to deposit money in the account & display all details.

(c)

```
#include <iostream>
using namespace std;
class bank_account
 {
   public:
    string name, account_type;
    float account_balance;
    void deposit(float);
    void display();
 };
void bank_account :: deposit(float x)
  {
    account_balance = account_balance + x;
  }
void bank_account :: display()
  {
    cout << "Name: " << name << end;
    cout << "Account Type: " << account_type <<
                                        endl;
    cout << "Account Balance: "
                << account_balance << endl;
  }
```

```
int main( )
{
    bank_account b1;
    b1.name = "Tanmay";
    b1.account_type = "Savings";
    b1.account_balance = 27000;
    b1.display( );
    b1.deposit (1000);
    cout << "Before Deposit" << endl;
    b1.display( );
    b1.deposit (1000);
    cout << "After Deposit" << endl;
    b1.display( );
    return 0;
}
```

Output :

```
Before Deposit
Name : Tanmay
Account Type : Savings
Account Balance : 27000

After Deposit
Name : Tanmay
Account Type : Savings
Account Balance : 28000
```

4. (a) What do you mean by abstract class?

(a) Abstract class is a class which contains atleast one pure virtual function in it. Abstract classes are used to provide an interface for its sub class. Class inheriting an abstract class must provide definition to the pure virtual function, otherwise they will also become abstract class.

The following is an example of abstract class:

```
class AB{
public:
virtual void f()=0;
};
```

→ Abstract class cannot be instantiated, but pointers & references of abstract class type can be created.

→ Abstract classes are mainly used for upcasting, so that its derived classes can use its interface.

(b) Write a simple program to show that how ambiguity in multipath inheritance can be avoided using scope resolution operator?

(b)
```
#include <iostream>
class ClassA
{
public:
int a;
};
class ClassB : public ClassA {
public:
int b;
};
```

```
class ClassC : public ClassA
 {
   public :
    int c;
 };

Class ClassD : public ClassB, public ClassC
 {  public :
    int d;
 };

void main( )
 {
    ClassD obj;
    obj.ClassB::a=10 ;
    obj.ClassC::a=100;
    obj.b=20 ;
    obj.c =30;
    obj.d=40;

    cout<< "\n A from ClassB :" << obj.ClassB::a;
    Cout << "\n A from ClassC :" << obj.ClassC::a;
    Cout<<"\n B : " << obj.b;
    Cout<< "\n C : " << obj.c;
    Cout << "\n D : " << obj.d;
 }
```

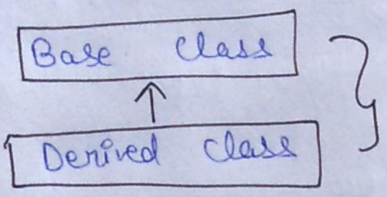
Output :

```
A from ClassB : 10
A from ClassC : 100
B : 20
C : 30
D : 40
```

4. (c) Demonstrate single inheritance with suitable diagram & program.

(c) If a single class is derived from one base class, then it is called single inheritance.

Single inheritance block diagram



→ Based on the visibility mode used or access specifier used while deriving, the properties of base class are derived. Access specifier can be private, protected or public.

Syntax:

```
class A                    // base class
  {-----
  };
Class B : access-specifier A // derived class
  {-----
  };
```

Program:

```
#include <iostream>
using namespace std;
class base
 {
  public:
  int x;
  void getdata()
  {
   cout << "Enter the value of x=";
   cin >> x;
  }
};
```

```
class derived : public base
{
  private :
  int y;
  public :
  void readdata()
    {
      cout << "Enter the value of y=";
      cin >> y;
    }
  void product()
    {
      cout << "Product =" << x*y;
    }
};
int main()
  {
    derived a; //object of derived class
    a.getdata();
    a.readdata();
    a.product();
    return 0
  }
```

Output :

```
Enter the value of x=3
Enter the value of y=4
Product = 12
```

5. (a) Differentiate between virtual function & pure virtual function.

(a) virtual function	Pure virtual function
(i) A virtual function is a member function of base class which can be redefined by derived class	(i) A pure virtual function is a member function of base class whose only declaration is provided in base class & should be defined in derived class otherwise derived class also becomes abstract.
(ii) Classes having virtual functions are not abstract.	(ii) Base class containing pure virtual function becomes abstract.
(iii) Definition is given in base class.	(iii) No definition is given in base class.
(iv) Base class having virtual function can be instantiated i.e. its object can be made.	(iv) Base class having pure virtual function becomes abstract i.e. it cannot be instantiated.
(v) If derived class do not redefine virtual function of base class, then it does not affect compilation.	(v) If derived class do not redefine virtual function of base class, then no compilation error but derived class also becomes abstract just like the base class.
(vi) All derived class may or may not redefine virtual function of base class.	(vi) All derived class must redefine pure virtual function of base class otherwise derived class also becomes abstract just like base class.

5. (b) Explain why friend function cannot overload '=' operator.

(b) We need to understand the way assignment operator works, the right-hand operand is source & lefthand operator is the target of assignment, the left-hand operand is typically called LValue & right-hand operand is called RValue.

Now, LValue must be a variable & cannot be constant, RValue can be a variable as well as constant.

When you overload assignment operator as member fn, it is the LValue object for which that assignment operator is called & the RValue object is passed as parameter & if the RValue is a constant & you have provided a constructor to convert that constant to obj then using that constructor that particular constant (which was a RValue) would be converted to object & will passed as parameter to assignment operator.

Now, if you provide a constant as LValue & say you are allowed to overload assignment operator as friend fn, then both the operands will be passed to the friend fn as parameter, now if there is a constructor the receives a constant & creates object, then compiler will use that constructor & convert that LValue (which is a constant for the given scenario) to object & will make a call to the assignment operator & you will be able to modify a constant in that way, which will be completely illogical & illegal.

Hence, to avoid modifying constant as LValue, assignment operator was restricted to be overloaded as friend function.

5. (c) Name 5 operators which cannot be overloaded.

(c) Following are the C++ operators that cannot be overloaded :

(i) :: → Scope resolution operator.

(ii) ?: → Ternary operator.

(iii) . → member selector.

(iv) Sizeof operator.

(v) * → Member pointer selector.