S.Y. Diploma : Sem. III
[CO/CM/IF/CW]

Object Oriented Programming

Time: 3 Hrs.] Prelim Question Paper Solution [Marks : 70

Q.1 Attempt any FIVE of the following : [10]

Q.1(a) Explain how to define a function outside the class definition. [2]

Ans.: Outside the Class Definition:

Member functions that are declared inside a class have to be defined separately outside the class. Their definitions are very much like the normal functions. They should have a function header and a function body.

```
return-type class-name :: function-name (argument declaration)
  {
             Function body
  }
```

Q.1(b) Explain the concept of Static Data Members (no C++ program needed). [2]

Ans.: Static Data Members

A data member of a class can be qualified as static. The properties of a static member variable are similar to that of a C static variable. A static member variable has certain special characteristics. These are:

- It is initialized to zero when the first object of its class is created. No other initialization is permitted.
- Only one copy of that member is created for the entire class and is shared by all the objects of that class, no matter how many objects are crated.
- It is visible only within the class, but its lifetime is the entire program.

Static variables are normally used to maintain values common to the entire class. For example, a static data member can be used as a counter that records the occurrences of all the objects.

Q.1(c) Explain the concept of destructor (no C++ Program needed) [2]

Ans.: A destructor, as the name implies, is used to destroy the objects that have been created by a constructor. Like a constructor, the destructor is a member function whose name is the same as the class name but is preceded by a tilde. For example, the destructor for a class integer can be defined as shown below:

```
-integer () {}
```

A destructor never takes any argument nor does it return any value. It will be invoked implicitly by the compiler upon exit from the program (or block or function as the case may be) to clean up storage that is no longer accessible. It is a good practice to declare destructors in a program since it releases memory space for future use.

Q.1(d) List the areas of application of OOP. **[2]**

Ans.: The promising areas for application of OOP include:

- Real-time systems
- Simulation and modeling
- Object-oriented databases
- Hypertext, hypermedia and expertext
- AI and expert systems
- Neural networks and parallel programming
- Decision support and office automation systems
- CIM/CAM/CAD systems.

Q.1(e) Describe the structure of a C++ program (with diagram). **[2]**

Ans.: Structure of C++ Program

As it can be seen from the Program 2.3, a typical C++ program would contain four sections as shown in Fig. 2.3. These sections may be placed in separate code files and then compiled independently or jointly.

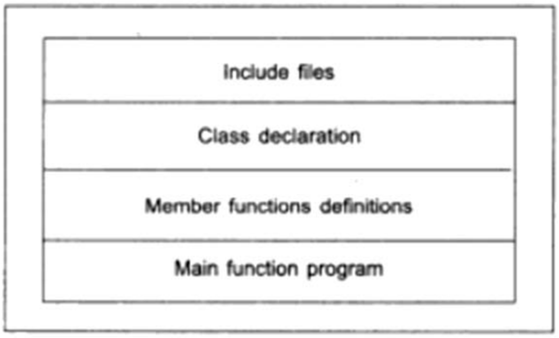


Fig. 2.3 Structure of C++ Program

It is a common practice to organize a program into three separate files.

Q.1(f) Explain the concept of identifiers. **[2]**

Ans.: **Identifiers:**

Identifiers refers to the names of variables, functions, arrays, classes etc. created by the programmer. They are the fundamental requirement of any language. Each language has its own rules for naming these identifiers. The following rules are common to both C and C++:

- Only alphabetic characters, digits and underscores are permitted.

- The name cannot start with a digit.
- Uppercase and lowercase letters are distinct.
- A declared keyword cannot be used as a variable name.

Q.1(g) List some languages that use OOP. [2]

Ans.: List of OOP Languages:

1. C++
2. Java
3. Visual Basic .Net
4. Visual C++ .Net
5. ASIP .Net

Q.2 Attempt any THREE of the following : [12]

Q.2(a) Explain copy constructor with example. [4]

Ans.: As stated earlier, a copy constructor is used to declare and initialize an object from another object. For example, the statement

integer 12(I1);

would define the object I2 and at the same time initialize it to the values of I1. Another form of this statement is

integer I2 = I1;

The process of initializing through a copy constructor is known as copy initialization. Remember, the statement

I2 = I1;

Q.2(b) Write a C++ Program to swap two numbers using pointers. [4]

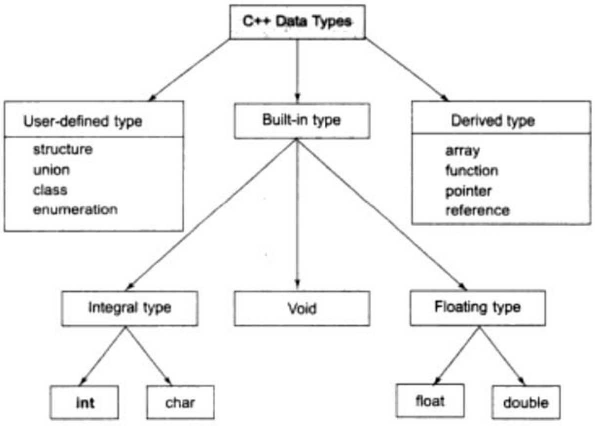
Ans.:

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int *a,*b,*temp;
    cout<<"Enter value of a and b";
    cin>>*a>>*b;
    temp=a;
    a=b;
    b=temp;
    cout<<"After swapping, a=" <<*a<< "b=" <<*b;
    getch();
}
```

Q.2(c) Draw a diagram showing the basic data types in C++ **[4]**

Ans.: **Basic Data Types:**

Data types in C++ can be classified under various categories as shown



Q.2(d) What is a friend function? What are its characteristics? (No C++ program needed) **[4]**

Ans.: **Friendly Functions:**

We have been emphasizing throughout this chapter that the private members cannot be accessed from outside the class. That is, a non-member function cannot have an access to the private data of a class. However, there could be a situation where we would like two classes to share a particular function. For example, consider a case where two classes, manager and scientist, have been defined. We would like to use a function income_tax() to operate on the objects of both these classes. In such situations, C++ allows the common function to be made friendly with both the classes, thereby allowing the function to have access to the private data of these classes. Such a function need not be a member of any of these classes.

To make an outside function "friendly" to a class, we have to simply declare this function as a friend of the class as shown below:

```
class ABC
{
        ……
        ……
    public:
        ……
        ……
        friend void xyz (void):  //declaration
};
```

The function declaration should be preceded by the keyword friend. The function is defined elsewhere in the program like a normal C++ function. The function definition does not use either the keyword friend or the scope operator::. The functions that are declared with the keyword friend are known as friend functions. A function can be declared as a friend in any number of classes. A friend function, although not a member function, has full access rights to the private members of the class.

A friend function possesses certain special characteristics:

- It is not in the scope of the class to which it has been declared as friend.
- Since it is not in the scope of the class, it cannot be called using the object of that class.
- It can be invoked like a normal function without the help of any object.
- Unlike member functions, it cannot access the member names directly and has to use an object name and dot membership operator with each member name. (e.g. A.x.).
- It can be declared either in the public or the private part of a class without affection its meaning.
- Usually, it has the objects as arguments.

Q.3 Attempt any THREE of the following : [12]

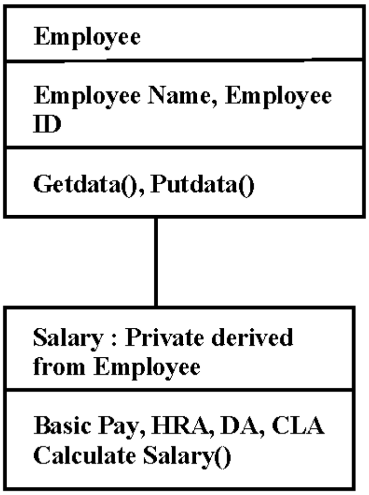
Q.3(a) Write a C++ program that replaces the string "Computer" in the String "Diploma in Computer Engineering" with string "Information Technology". [4]

Ans.:

```
#include <iostream.h>
#include <string.h>
void findAndReplaceAll(std : : string & data, std : : string toSearch, std : :
string replaceStr)
{
    size_t pos = data.find(toSearch);
    while( pos != std : : string : : npos)
    {
        // Replace this occurrence of Sub String
        data.replace(pos, toSearch.size(), replaceStr);
        // Get the next occurrence from the current position
        pos =data.find(toSearch, pos + toSearch.size());
    }
}

void main()
```

```
{
        clrscr( );
    std : : string data = "Diploma in Computer Engineering";
    std : : cout<<data<<std : : endl;
    findAndReplaceAll(data, "Computer", "Information Technology");
    std : : cout<<data<<std : : endl;
    getch();
}
```

Q.3(b) **[4]**



Define classes to appropriately represent class hierarchy as shown in above figure. Use constructors for both classes and display Salary for a particular employee.

Ans.:

```
#include<iostream.h>
#include<conio.h>
class employee
{
    public:
    int empid; char empname[20];
    employee()
    {
        cout<<"New employee record details:"<<endl;
        cout<<"Enter Employee ID:";
        cin>>empid;
        cout<<"Enter Employee Name:";
        cin>>empname;
    }
    void showdata()
```

```
     {
          cout<<"Employee ID: "<<empid<<" has name: "<<empname<<endl;
     }
};

class salary:private employee
{
     public:
          float basic,hra,da,cca,total;
          salary():employee()
          {
               cout<<"Enter basic salary (in Rupees):";
               cin>>basic;
               cout<<"Enter house rent allowance (in Rupees):";
               cin>>hra;
               cout<<"Enter dearness allowance (in Rupees):";
               cin>>da;
               cout<<"Enter city compensatory allowance (in Rupees):";
               cin>>cca;
               total = basic+hra+da+cca;
               cout<<"The total salary of the employee: "<<empname<<" having
               ID: "<<empid<<" is Rs. "<<total<<endl;
     }
};

void main()
{
     clrscr();
     salary s1;
     s1.getdata();
     getch();
}
```

Q.3(c) Define a class named 'Train' representing following members: [4]
Data members :
- **Train Number**
- **Train Name**
- **Source**
- **Destination**
- **Journey Date**
- **Capacity**

Member functions :

- **Initialise members**
- **Input Train data**
- **Display data**

Write a C++ program to test the train class.

Ans.:
```
#include<iostream.h>
#include<conio.h>

class train
{
     public:
          int num, capacity;
          char name[20], source[20], dest[20], date[20];
          void getdata()
          {
               cout<<"Enter Train Number:";
               cin>>num;
               cout<<"Enter Train Name:";
               cin>>name;
               cout<<"Enter Train Source Station:";
               cin>>source;
               cout<<"Enter Train Destination Station:";
               cin>>dest;
               cout<<"Enter date of journey in DD-MM-YYYY format:";
               cin>>date;
               cout<<"Enter total number of people that the train can carry:";
               cin>>capacity;
          }
     void showdata()
     {
          cout<<"Train Details:\n\nTrain Number:"<<num<<"\nTrain
Name:"<<name<<"\nTrain Source Station:"<<source<<"\nTrain Destination
Station:"<<dest<<"\nData of Journey:"<<date<<"\nCapacity: "<<capacity<<"
people."<<endl;
     }
};

void main()
{
     train t1;
     clrscr();
     t1.getdata();
     t1.showdata();
     getch();
}
```

Q.3(d) Write a C++ Program to copy a file abc.txt into another file xyz.txt [4]

Ans.:

```
#include<iostream.h>
#include<conio.h>
#include<fstream.h>
#include<stdio.h>
#include<stdlib.h>
void main()
{
     clrscr();
     ifstream fs;
     ofstream ft;
     char ch, fname1[20], fname2[20];
     cout<<"Enter source file name with extension (like abc.txt) : ";
     gets(fname1);
     fs.open(fname1);
     if(!fs)
     {
          cout<<"Error in opening source file..!!";
          getch();
          exit(1);
     }
     cout<<"Enter target file name with extension (like xyz.txt) : ";
     gets(fname2);
     ft.open(fname2);
     if(!ft)
     {
          cout<<"Error in opening target file..!!";
          fs.close();
          getch();
          exit(2);
     }
     while(fs.eof()==0)
     {
          fs>>ch;
          ft<<ch;
     }
```

```
        cout<<"File copied successfully..!!";
        fs.close();
        ft.close();
        getch();
    }
```

Q.4 Attempt any THREE of the following : [12]

Q.4(a) Write a program to implement single inheritance. [4]

Ans.: Single Inheritance : Public

```
#include <iostream?

using namespace std;

class B
{
        int a;              //  private;    not inheritable
    public:
        int b;              //  public;     ready for inheritance
        void get_ab();
        int get_a(void);
        void show_a(void);
};
class D : public B          //  public derivation
{
        int c;
    public:
        void mul (void);
        void display (void);
};
//――――――――――――――――――――――――――――――――――――――――――――――――――――――――
void B : : get_ab(void)
{
        a = 5; b = 10;
}
int B : : get_a()
        return a;
}
void B : : show_a()

    count << "a = " << a << "\n";
}
```

```
void D : : mul()
{
     c = b * get_a();
}
void D : : display()
{
     cout << "a = " < get_a() << "\n";
     cout << "b = " << b << "\n";
     cout << "c = " << c << "\n\n";
}
//————————————————————————————————————————————————
int main()
{
     D d;
     d.get_ab();
     d.mul();
     d.show_a();
     d.display();

     d.b = 20;
     d.mul();
     d.display();

     return 0;
}
```

Q.4(b)Write a program to implement multilevel inheritance. **[4]**

Ans.:

```
#include <iostream>
using namespace std;
class student
{
     protected:
          int roll_number;
     public:
          void get_number(int);
          void put_number(void);
};
void student : : get_number(int a)
{
          roll_number = a;
}
```

```
void student : : put_number()
{
          cout <<"Roll Number: "<< Roll_number << "\n";
}
class test : public student      //  First level derivation
{
     protected:
          float sub1;
          float sub2;
     public:
               void get_marks(float, float);
               void put_marks(void);
};
void test : : get_marks(float x, float y)
{
     sub1 = x;
     sub = y;
}
void test : : put_marks()
{
     cout << "Marks in SUB1 = "<< sub1 << "\n";
     cout << "Marks in SUB2 = "<< sub2 << "\n";
}
class result : public test       // Second level derivation
{
          float total;                 // private by default
     public:
          void display(void):
};
void result : : display(void)
{
          total = sub1 + sub2;
          put_number();
          put_marks();
          cout << "Total = " << total << "\n";
}
int main()
{
          result student 1;            // student1 created

          student1.get_number(111);
```

```
            student1.get_marks(75.0, 59.5);

            student1.display();
        return 0;
    }
```

Q.4(c) Write a C++ program to implement multiple inheritance. **[4]**

Ans.: Multiple Inheritance

```
#include <iostream>

using namespace std;

class M
{
    protected:
            int m;
    public:
            void get_m(int);
};
class N
{
    protected:
        int n;
    public:
        void get_m(int);
};

class P : public M, public N
{
    public:
        void display(void);
};

void M : : get_m(intx)
{
        m = x;
}

void N : : get_n(int y)
{
        n = y;
}

void P : : display(void)
{
        cout << "m + " << m << "\n";
        cout << "n = " << n << "\n";
        cout << "m*n = " << m*n << "\n";
}
int main ()
{
        P p;

        p.get_m(10);
        p.get_n(20);
        p.display();

        return 0;
}
```

Q.4(d) Explain the benefits of OOP. **[4]**

Ans.: Benefits of OOP:

- Through inheritance, we can eliminate redundant code and extend the use of existing classes.
- We can build programs from the standard working modules that communicate with one another, rather than having so start writing the code from scratch. This leads to saving of development time and higher productivity.
- The principle of data hiding helps the programmer to build secure programs that cannot be invaded by code in other parts of the program.
- It is possible to have multiple instances of an object to co-exist without any interference.
- It is possible to map objects in the problem domain to those in the program.
- It is easy to partition the work in a project based on objects.
- The data-centered design the approach enables us to capture more details of a model in implementable form.
- Object-oriented systems can be easily upgraded from small to large systems.
- Message passing techniques for communication between objects makes the interface descriptions with external systems much simpler.
- Software complexity can be easily managed.

Q.4(e)Write a C++ program to show an example of implementation of Destructors. **[4]**

Ans.: Implementation of destructors

```
#include <iostream>

using namespace std;

int count = 0;

class alpha
{
    public:
        alpha()
        {
            count++;
            cout << “\nNo. of object created” << count;
        }

        ~alpha()
        {
            cout << “\nNo. of object destroyed” << count;
            count––;
        }
};
int main()
{
        cout << “\n\nENTER MAIN\n”;

        alpha A1, A2, A3, A4;
        {
            cout << “\n\nENTER BLOCK1\n”;
            alpha A5;
        }
        {
            cout << “\n\nENTER BLOCK2\n”;
            alpha A6;
        }
        cout << “n\nRE-ENTER MAIN\n”;

        return 0;
}
```

Q.5 Attempt any TWO of the following : [12]

Q.5(a) Explain the characteristics of constructors. [6]

Ans.: The constructor functions have some special characteristics. These are:

- They should be declared in the public section.
- They are invoked automatically when the objects are created.
- They do not have return types, not even void and therefore, and they cannot return values.
- They cannot be inherited, though a derived class can call the base class constructor.
- Like other C++ functions, they can have default arguments.
- Constructors cannot be virtual. (Meaning of virtual will be discussed later in Chapter 9.)
- We cannot refer to their addresses.
- An object with a constructor (or destructor) cannot be used as a member of a union.
- They make 'implicit calls' to the operators new and delete when memory allocation is required.

Q.5(b) Write a C++ program to display number of objects created using static member. [6]

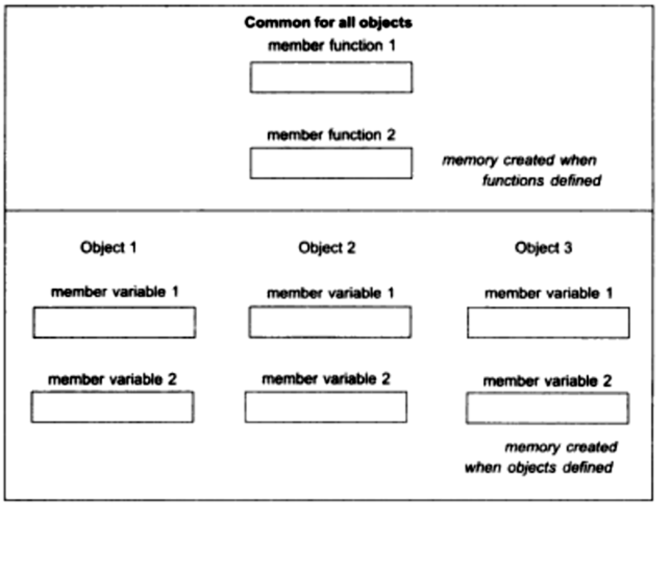
Ans.: **Static Class Member**

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

class item
{
            static int count;
            int number;
    public:
            void getdata (int a)
            {
                number = a;
                count ++;
            }
            void getcount (void)
            {
                cout. << "count";
                cout << count << "\n";
            }
};
int item : : count;

int main ( )
```

```
{
            item a, b, c;               // count is initialized to zero
            a.getcount ( );             // display count
            b.getcount ( );
            c.getcount ( );

            a.getcount (100);           // getting data into object a
            b.getcount (200);           // getting data into object b
            c.getcount (300);               // getting data into object c

            count << "After reading data" << "\n";

            a.getcount ( );             // display count
            b.getcount ( );
            c.getcount ( );
            return 0;
}
```

Q.5(c) Explain the concept of memory allocation for objects. [6]

Ans.: Memory space for objects is allocated when they are declared and not when the class is specified. This statements is only partly true. Actually, the member functions are created and placed in the memory space only once when they are defied as a part of a class specification. Since all the objects belonging to that class use the same member functions, no separate space is allocated for member functions when the objects are created. Only space for member variables is allocated separately for each object. Separate memory locations for the objects are essential, because the member variables will hold different data values for different objects. This is shown in Fig. 5.3.



Q.6 Attempt any TWO of the following : [12]

Q.6(a) Define Operator Overloading. Write a C++ program to implement Unary Operator Overloading. [6]

Ans.: Defining Operator Overloading:

To define an additional task to an operator, we must specify what it means in relation to the class to which the operator is applied. This is done with the help of a special function, called operator function, which describes the task. The general form of an operator function is:

```
    return type classname : : operator op(arglist)
    {
            Function body           // task defined
    }
```

where return type is the type of value returned by the specified operation and op is the operator being overloaded. The op is preceded by the keyword operator. operator op is the function name.

```
#include <iostream>
using namespace std;
class space
{
        int x;
        int y;
        int z;
public:
        void getdata(int a, int b, int c);
        void display(void);
        void operator-();       // overload unary minus
};
void space : : getdata(int a, int b, int c)
{
        x = a;
        y = b;
        z = c;
}
void space : : display(void)
{
        cout << x << " ";
        cout << y << " ";
        cout << z << "\n";
}
void space : : operator –()
{
        x = -x;
        y = -y;
        z = -z;
}
int main()
{
        space S;
        S.getdata(10, -20, 30);
        cout << "S : ";
        S. display();
        - S;
        cout << "S : ";
        S.display();
        return 0;
}
```

Q.6(b) Write a C++ Program to implement Constructor in Derived Class. [6]

Ans.:

```
#include <iostream>
using namespace std;
class alpha
{
    int x;
  public:
    alpha(int i)
    {
        x = i;
        cout << "alpha initialized \n";
    }
    void show x(void)
    {   cout << "x =" << x << "\n";   }
};
class beta
{
    float y;
 public:
    beta(float j)
    {
        y = j;
```

```
            cout << "beta initialized \n";
        }
        void show_y(void)
        {  cout << "y = " << y << "\n";}
    };
    class gamma: public beta, public alpha
    {
        int m, n;
      public:
        gamma(int a, float b, int c, int d):
                alph(a), beta(b)
        {
                m = c;
                n = d;
                cout << "gama initialized \n";
        }
            void show_mn(void)
            {
                cout << "m = " << m << "\n"
                    << "n = " << n << "\n";
            }
    };
    int main()
    {
        gamma g (, 10, 75, 20, 30);
        cout << "\n";
        g.show_x();
        g.show_y();
        g.show_mn();
        return 0;
    }
```

Q.6(c) Explain arithmetic operations on pointers. Write a C++ Program to show these operations. [6]

Ans.: There are a substantial number of arithmetic operations that can be performed with pointers. C++ allows pointers to perform the following arithmetic operations:

- A pointer can be incremented (++) (or) decremented (– –)
- Any integer can be added to or subtracted from a pointer
- One pointer can be subtracted from another

```
#include <iostream.h>
#include <conio.h>
void main()
{
     int num □ = (56, 75, 22, 18, 90);
     int *ptr;
     int i;
     clrscr();
     cout << "The array values are:\n";
     for(i – 0; i < 5; i++)
               cout << num [i] << "\n";
     /* Initializing the base address of str to ptr*/ ptr – num;
     /* Printing the value in the array */
     cout << "\nValue of ptr       :     "<< *ptr;
     cout << "\n";
     ptr++;
     cout << "\nValue of ptr++     :     :<<*ptr;
     cout << "\n";
     ptr – –;
     cout << "\nValue of ptr – –   :     "<<*ptr;
     cout << "\n";
     ptr = ptr +2;
     cout << "\nValue of ptr+2     :     "<<*ptr;
     cout << "\n";
     ptr = ptr – 1;
     cout << "\nValue of ptr – 1   :     "<<*ptr;
     cout << "\n";
     ptr+ = 3;
     cout << "\nValue of ptr+ - 3:        "<<*ptr;
     ptr - = 2;
     cout << "\n";
     cout <<"\nValue of ptr - = 2:        "<<*ptr;
     cout << "\n";
     getch();
```

❑ ❑ ❑ ❑ ❑