2066

Section A $(2 \times 10 = 20)$

1. Explain in detail the following principles of Object-Oriented Programming.

i) Data encapsulation and data hiding.

ii) Inheritance and polymorphism.

iii) Abstraction

Data encapsulation and data hiding:

The mechanism of wrapping up of data and function into a single unit is called encapsulation. Because of encapsulation data and its manipulating function can be kept together. We can assume encapsulation as a protective wrapper that prevents the data being accessed by other code defined outside the wrapper. By making use of encapsulation we can easily achieve abstraction. The purpose of a class is to encapsulate complexity. Each data or function in a class can be marked as private or public. The public interface of a class represents everything that external users of the class may know about the data and function. The private function and data can only be accessed by code that is a member of a class. The code other than member of a class cannot access a private function or data. This insulation of data from direct access by the program is called data hiding. After hiding data by making them private, it is then safe from accidental alteration.

Inheritance and polymorphism:

Inheritance is the process by which objects of one class acquire the characteristics of object of another class. We can use additional features to an existing class without modifying it. This is possible by deriving a new class (derived class) from the existing one (base class).This process of deriving a new class from the existing base class is called inheritance. It provides the concept of hierarchical classification. It allows the extension and reuse of existing code without having to rewrite the existing code. We naturally view the whole world is made up of objects. Many objects are related to each other in a hierarchical way, such as vehicle, four wheeler, and car. If we describe vehicle in an abstract way, the attributes may be such as color, number of seats etc. All vehicles have common behavioral aspect like; they move, accelerate, turn and stop. The more specific class of vehicle is four wheeler that acquires all the features of class vehicle and has more specific attributes like engine number, chassis number etc. The class vehicle is called base class (or super class) and class four wheeler is called derived class (or subclass).

Polymorphism means 'having many forms'. The polymorphism allows different objects to respond to the same operation in different ways, the response being specific to the type of object. The different ways of using same function or operator depending on what they are operating on is called polymorphism. Example of polymorphism in OOP is operator overloading, function overloading. Still another type of polymorphism exist which is achieved at run time also called dynamic binding. For example operator symbol '+' is used for arithmetic operation between two numbers, however by overloading it can be used over Complex Object like currency that has Rs and Paisa as its attributes, complex number that has real part and imaginary

part as attributes. By overloading same operator '+' can be used for different purpose like concatenation of strings. When same function name is used in defining different function to operate on different data (type or number of data) then this feature of polymorphism is function overloading.

Abstraction:
Abstraction is representing essential features of an object without including the background details or explanation. It focuses the outside view of an object, separating its essential behavior from its implementation. We can manage complexity through abstraction. Operating System like Windows, UNIX provides abstraction to the user. The user can view his files and folders without knowing internal detail of Hard disk like the sector number, track number, cylinder number or head number. Operating System hides the truth about the disk hardware and presents a simple file-oriented interface. The class is a construct in object oriented programming for creating user-defined data for abstraction. When data and it operation are presented together, the construct is call ADT (Abstract Data Type). In OOP classes are used in creating ADT. For example, a student class can be made and can be available to be used in programs. The programmer can implement the class in creating objects and its manipulation without knowing its implementation. The program can use the function Sort_name() to sort the names in alphabetical order without knowing whether the implementation uses bubble sort, merge sort, quick sort algorithms.

2. Why constructor and destructor are required on Object Oriented Programming? Explain with suitable example.

A constructor is a special member function that is executed automatically whenever an object is created. It is used for automatic initialization. Automatic initialization is the process of initializing object's data members when it is first created, without making a separate call to a member function. The name of the constructor is same as the class name. For example:

```
class rectangle
{       private:
        int length;
        int breadth;
        public:
        rectangle()
        {       //constructor
                length = 0;
                breadth = 0;
        }
……
……
};
```

A destructor is a special member function that is executed automatically just before lifetime of an object is finished. A destructor has the same name as the constructor (which is the same as the class name) but is preceded by a tilde (~). Like constructors, destructors do not have a return

value. They also take no arguments. Hence, we can use only one destructor in a class. The most common use of destructors is to deallocate memory that was allocated for the object by the constructor. For example:

```
class Test
{       private:
        int x,y;
        public:
        Test()
        {       cout<<"Memory is allocated"<<endl;
        }
        ~Test()
        {       cout<<"Memory is deallocated"<<endl;
        }
}
void main()
{       clrscr();
        {       Test p;
        }       // life time of p finishes here, and destructor is called
        getch();
}
```

Output:
Memory is allocated
Memory is de-allocated

3. Define a *student* class (with necessary constructors and member functions) in Object Oriented Programming (abstract necessary attributes and their types). Write a complete code in C++ programming language.

- **Derive a C*omputer Science and Mathematics* class from *student* class adding necessary attributes (at least three subjects).**
- **Use these classes in a main function and display the average marks of computer science and mathematics students.**

```
#include<iostream.h>

#include<conio.h>

class student

{       protected:

        float english, sum, avg;
```

```
        public:
        void getstudentdata()
        {       cout<<"Enter english marks:"<<endl;
                cin>>english;
        }
};
class computer:public student
{       float IT, cprog, networks;
        public:
        void getcomputerdata()
        {       cout<<"Enter marks in IT:"<<endl;
                cin>>IT;
                cout<<"Enter marks in cprog:"<<endl;
                cin>>Cprog;
                cout<<"Enter marks in networks:"<<endl;
                cin>>Networks;
        }
        void average()
        {       sum=english+IT+cprog+networks;
                avg=sum/4;
                cout<<"Average marks is"<<avg;
        }
};
class mathematics:public student
{       float calculus, stat, algebra;
```

```
        public:
        void getmathdata()
        {       cout<<"Enter marks in calculus:"<<endl;
                cin>>calculus;
                cout<<"Enter marks in statistics:"<<endl;
                cin>>stat;
                cout<<"Enter marks in Linear Algebra:"<<endl;
                cin>>algebra;
        }
        void average()
        {       sum=english+calculus+stat+algebra;
                avg=sum/4;
                cout<<"Average marks is"<<avg;
        }
};
int main()
{       computer C;
        mathematics M;
        cout<<"Enter marks of computer students:"<<endl;
        C.getstudentdata();
        C.getcomputerdata();
        cout<<"Enter marks of mathematics student:"<<endl;
        M.getstudentdata();
        M.getmathdata();
        C.average();
```

```
        M.average();
        return 1;
}
```

Section B ($8 \times 5 = 40$)

4. What is type casting? Explain with suitable example.

Sometimes, a programmer needs to convert a value from one type to another in a situation where the compiler will not do it automatically. For this C++ permits explicit type conversion of variables or expressions as follows:

```
(type-name) expression      //C notation
type-name (expression)      //C++ notation
```

For example:

```
int a = 10000;
int b = long(a) * 5 / 2;        //correct
int b = a * 5/2;                //incorrect
```

5. Write a program to compute subtraction of two complex numbers using operator overloading.

```
#include<iostream.h>
 #include<conio.h>
 #include<string.h>
 #include<stdio.h>
 class complex
 {      int i,r;
        public:
        void read()
        {       cout<<"\nEnter Real Part:";
                cin>>r;
                cout<<"Enter Imaginary Part:";
                cin>>i;
```

```
        }
        void display()
        {       cout<<"\n= "<<r<<"+"<<i<<"i";
        }
        complex operator-(complex a2)
        {       complex a;
                a.r=r-a2.r;
                a.i=i-a2.i;
                return a;
        }
};
void main()
{       clrscr();
        complex a,b,c;
        cout<<"\nEnter the first complex number:";
        a.read();
        a.display();
        cout<<"\nEnter the second complex number:";
        b.read();
        b.display();
        c=b-a;
        c.display();
        getch();
}
```

6. Why exception handling is required? Explain with suitable example.

Exception handling mechanism in C++ is basically built upon three keywords: try, throw, and catch. The keyword try is used to surround a block of statements, which may generate exceptions. This block of statements is known as try block. When an exception is detected, it is thrown using the throw statement situated either in the try block or in functions that are invoked from within the try block. This is called throwing an exception and the point at which the throw is executed is called the throw point. A catch block defined by the keyword catch catches the exception thrown by the throw statement and handles it appropriately. This block is also called exception handler. The catch block that catches an exception must immediately follow the try block that throws an exception.
For example:

```
#include<iostream.h>
#include<conio.h>
void main()
{       clrscr();
        int a, b;
        cout<<"Enter values of a & b:\n";
        cin>>a>>b;
        try
        {       if(b == 0)
                throw b;
                else
                cout<<"Result = "<<(float)a/b;
        }
        catch(int i)
        {       cout<<"Divide by zero exception: b = "<<i;
        }
        cout<<"\nEND";
        getch();
}
```

7. Differentiate between super class and sub class with suitable examples.

Inheritance is the process of deriving new class from the existing class. Newly created class is called sub class (also called as child class or derived class) and existing old class is called as super class (also called as parent class or base class). If we derive a class, all the features of parent class are inherited into child class and also we can add extra new features to the class. This inheritance provides us with the mechanism of adding new features to the class without directly modifying it. For example:

```
class person

{       char name[20];
```

```
        int age;
        Public:
        void getperson()
        {       cout<<"Enter name and age"<<endl;
                cin>>name>>age;
        }
        void displayperson()
        {       cout<<"Name:"<<name<<endl;
                cout<<"Age:"<<age<<endl;
        }
};
class employee: Public person          // employee is the sub class of person which is a super class
{       Public:
        int eid, sal;
        void getemployee()
        {       cout<<"Enter ID and salary:"<<endl;
                cin>>eid>>sal;
        }
        void displayemployee()
        {       cout<<"EID:"<<eid<<endl;
                cout<<"Salary:"<<sal<<endl;
        }
};
void main()
{       clrscr();
```

```
        employee e;
        e.getperson();
        e.getemployee();
        cout<<"Employee details:"<<endl;
        e.displayperson();
        e.displayemployee();
        getch();
}
```

8. Write a program in C++ to count a number of words in a line of text.

```
#include<iostream.h>
#include<stdio.h>
#include<conio.h>
void main()
{       char line[80], c;
        int i=0, j=1;
        clrscr();
        cout<<"Enter the sentence:"<<endl;
        gets(line);
        cout<<"You have entered:"<<endl;
        puts(line);
        while((c=(line[i++]))!='.')
        {       if(c==' ')
                j=j+1;
        }
        cout<<j<<endl;
```

```
    getch();
}
```

9. Differentiate between function overriding and function overloading. Explain with suitable examples.

Overloading means having functions with the same name but with different signature. Signature includes method name and Parameters. These functions can be part of base class or derived class. Whereas Overriding means changing the functionality of a method without changing the signature. We can override a function in base class by creating a similar function in derived class and by use virtual/override keywords.

Example for function overriding:

```
class A
{       public:
        void show()
        {       cout<<"This is class A";
        }
};
class B : public A
{       public:
        void show()
        {       cout<<"This is class B"<<endl;
        }
};
void main()
{       clrscr();
        B b;
        b.show(); //invokes the member function from class B
```

```
        b.A :: show(); //invokes the member function from class A
        getch();
}
```

Example for function overloading:

```
#include <iostream>
int mul (int a, int b)
{       return (a*b);
}
float mul (float a, float b)
{       return (a*b);
}
int main ()
{       int x=5,y=2;
        float n=5.0,m=2.0;
        cout << mul (x,y);
        cout << "\n";
        cout << mul(n,m);
        cout << "\n";
        return 0;
}
```

10. Explain the role of polymorphism in Object Oriented Programming.

Polymorphism means state of having many forms. We know that polymorphism is implemented using the concept of overloaded functions and operators. In this case, polymorphism is called early binding or static binding or static linking. This is also called compile time polymorphism because the compiler knows the information needed to call the function at the compile time and, therefore; compiler is able to select the appropriate function for a particular call at the compile

time itself. There is also another kind of polymorphism called run time polymorphism. In this type, the selection of appropriate function is done dynamically at run time. So, this is also called late binding or dynamic binding. C++ supports a mechanism known as virtual functions to achieve run time polymorphism. Run time polymorphism also requires the use of pointers to objects.

11. Explain the different type of class access specifiers.

The specification starts with the keyword **class** followed by the class name. Like structure, its body is delimited by braces terminated by a semicolon. The body of the class contains the keywords private, public, and protected. Private data and functions can only be accessed from within the member functions of that class. Public data or functions, on the other hand are accessible from outside the class. Usually the data within a class is private and functions are public. The data is hidden so it will be safe from accidental manipulation, while the functions that operated on the data are public so they can be accessed from outside the class.

Access Specifier	Accessible from Own Class	Accessible from Derived Class	Accessible from Objects Outside the Class
Public	Yes	yes	yes
protected	Yes	Yes	no
Private	Yes	No	no

12. Write a program to find the cube of given integer using inline function.

```
#include<iostream.h>

#include<conio.h>

inline void cube(int a)

{       int cube;

        cube=a*a*a;

        cout<<"Cube="<<cube<<endl;

}

void main()

{       clrscr();
```

```
        int x;
        cout<<"Enter a number:"<<endl;
        cin>>x;
        cube(x);
        getch();
}
```

13. Write a program to convert centigrade into Fahrenheit temperature.

```
#include<iostream.h>
#include<conio.h>
void main()
{       float ftemp, ctemp;
        cout<<"Enter the temperature in centigrade:"<<endl;
        cin>>ctemp;
        ftemp=(ctemp*9/5)+32;
        cout<<"Equivalent temperature in fahrenheit is:"<<ftemp<<endl;
        getch();
}
```