Q.2 a. Explain the three important features of object-oriented programming.

Answer:

The three major features of OOPS are:

(i) *Encapsulation*: The process of data hiding and combining data and methods in a single logical unit is called encapsulation. In C++, encapsulation is implemented through classes. A class in C++ consists of data and methods.

(ii) *Inheritance*: A class can inherit the data and methods from another existing class. The class getting the features is known as derived class and the class from which the features are inherited is known as the base class. Inheritance helps in preserving the investment in the existing code by allowing us to extend the functionality of existing classes.

(iii)*Polymorphism*: The classes belonging to the same family exhibit the same characteristics. To implement the behaviour, we write methods. A method may use the same name across several classes belonging to a single family; however, actual implementation may vary across such classes. This feature of OOPS is known as polymorphism.

b. Write a program to display the multiplication table of the number entered by the user.

Answer:

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

int main() {
        int    i, n;

        cout << "Enter the number whose table is required: ";
        cin >> n;
        cout << "Table of " << n << endl;
        for (i = 1; i <= 10; i++)
                cout << n << " * " << i << " = " << n * i << endl;
        return 0;
}
```

c. Compare *while* and *do…while* loops, with the help of example.

Answer:

do..while loop	while loop
• The format of do.. while loop is: do{ ….} while(condition); • The loop body is executed so long as the condition is true. • It is executed atleast once even if the condition is false before control enters the loop body.	• The format of while loop is: while(condition) { … } • The loop body is executed so long as the condition is true. • It is not executed once even if the condition is false before control enters the loop body.

Q.3 a. Define array. Give the general syntax for declaring an array. How we initialize an array at compile time?

Answer: Page 48, 49 to 51 of Text Book 1

b. Define a structure of Employee with the following fields: empNo, name and salary. Write a program to read and store the data of at most 10 employees in an array. Also display the average salary of the employees.

Answer:

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

struct employee {
            int         empNo;
            char        name[20];
            float       salary;
};

int main() {
            employee            e[10];
            int                 i, n;
            float               sum = 0;
            cout << "How many employees? ";
            cin >> n;
            cout << "Enter the data for " << n << " employees\n";
            for (i = 0; i < n; i++) {
              cout << "\nEnter details of employee : " << i + 1 << "\nEmployee number:
";
              cin >> e[i].empNo;
              fflush(stdin);
              cout << "Name: ";
              gets(e[i].name);
              cout << "Salary: ";
              cin >> e[i].salary;
              sum += e[i].salary;
            }
            cout << "\nThe average salary of the employees is " << sum / n;
            return 0;
}
```

Q.4 Write short notes on the following (any <u>FOUR</u>):
(i) class & objects
(ii) friend functions
(iii) passing parameters to a function by reference
(iv) static data members
(v) access modifiers

Answer:

(i) Class: Class allows us to combine data and the methods that operate on data, under a single logical data unit. The general declaration of a class is shown below:

```
class className {
  private:
          variable / function declarations;
  public:
          variable / function declarations;
  protected:
          variable / function declarations;
};
```

For example, a class for a student can be defined as follows:

```
class student {
  private:
          int     rollNo;
          char    name[30];
          float   marks;
  public:
          void input() { … }
          void output() { … }
};
```

4 (ii) Friend functions: A friend function is a function that is defined outside a class but is still allowed to access the private data of the class of which it is a friend. Friend functions access the private data of a class through an object of the class. To treat an external function as a friend of a class, a declaration of the function must appear in the class with the keyword *friend*. The function implementation is external to the class body and does not constitute a part of the class. For example, the following code shows a friend function *fn* of a class A which is accessing its private member x through the object a.

```
class A {
  private:
          int      x;
  public:
          A() { x = 10 };
          friend void fn();
};
void fn(A  a) {
```

```
                cout << a.x;
        }
```

4 (iii) Passing parameters to a function by reference: If we want that the changed value of a parameter modified within a function to be reflected back in the original value in the calling function, we should use pass the parameter by reference concept. To pass the variable by reference, we declare a pointer to the variable and pass the pointer variable as a parameter to the function. When the function is called, address of operator (&) is used along with this parameter. For example,

```
        void A(int *p) { *p = 10; }
        void B() {
          int x = 5;
          cout << x << endl;
          A(&x);
          cout << x << endl;
        }
```

The output of the code segment will be

```
        5
        10
```

4 (iv) Static data members: There is only one copy of a static data member and it is shared by all the objects of the class. Any change made by any object is reflected in all the objects. The static data members are visible only within the class but their lifetime is the entire program. Its declaration is prefixed with the keyword static. A static data member has to be defined outside the class using the scope resolution operator. For example,

```
        class A {
          private:
                static int i;
                …
        };
        int A::i = 10;
```

4 (v) Access modifiers: There are three access modifiers available in C++:

a. public: A public member of a class is accessible to any program code defined inside the class, defined in an inherited class or defined outside the class.
b. private: These members are available only to the functions defined within the class.
c. protected: A protected member of a class is accessible to any program code defined inside the class or defined in an inherited class but not to the code defined outside the class.

Q.5 a. Define a class *Complex* having a real part and an imaginary part. Include the following functions in the class:

- **A constructor to initialize the values of the members to 0**
- **A function to initialize the data members of the class**
- **Overload + operator to add two complex numbers**

A function to display a complex number

Answer:

```
class complex {
              int r, i;
              public:
                complex() {
                        r = 0;
                        i = 0;
                }

                void initialize() {
                        cout << "Enter the real part: ";
                        cin >> r;
                        cout << "Enter the imaginary part: ";
                        cin >> i;
                }

                complex operator + (complex b) {
                        complex c;
                        c.r = r + b.r;
                        c.i = i + b.i;
                        return c;
                }

                void display() {
                        cout << "\nThe complex number is " << r << "+" << i << "i";
                }
              };
```

b. What is a destructor? What are its properties?

Answer:

Destructors are special methods of a class that are automatically invoked when the object is destroyed and removed from the system. These are generally required when memory is dynamically allocated in the constructor of the class and it is required to release it before the object is destroyed.

The properties of a destructor are as follows:

- A destructor is a class method having the same name as the class name and is prefixed with a tilde (~) character
- Destructor does not take any arguments

- Destructor does not return anything, not even void data type
- Destructors cannot be overloaded

Q.6 a. What is multiple level inheritance? What will be the calling sequence for constructors and destructors for the following class definitions :

```
class A{ … };
class B : public A { … };
class C: protected B { … };
```

Answer:

Multiple level inheritance: When a class say B inherits the features from a base class say A and some other class say C inherits the features from B, it is known as multiple level inheritance. This can be further extended to any levels of inheritance. At any level of inheritance, a class has features of all the classes that lie in the path from the base of the hierarchy to that level as per the access modifiers that are used in inheritance in the path.

This can be shown through an example as follows:

```
class A{ … };
class B : public A { … };
class C: protected B { … };
```

Here, class C will have the features from A as well as B as per the access modifiers (public and protected in this case) that lie in the path from A to C.

Calling hierarchy for constructors and destructors:

Constructor of A will be called followed by constructor of B and then finally of C.

Destructor of C will be called followed by destructor of B and then finally of A.

b. What are virtual inheritance? Why is it required?

Answer:

Virtual inheritance: Virtual inheritance is used to overcome the problem faced when multiple level inheritance and multiple inheritance are used together. For example, consider the case below:

```
class A {
    public:
        int  a;
    …
};
class B : public A { … }
class C : public A { … }
class D : public B, public C { … }
```

In this case, D will have two copies of grandparent variable ‘a’. Hence, if there is a call to display the value of ‘a’ in D then which ‘a’ will it refer to. Solution to this problem is virtual inheritance. It is to use *virtual* keyword when inheritance is being carried out

```
class A {
        public:
            int  a;
        …
    };
    class B : public virtual A { … }
    class C : public virtual A { … }
    class D : public B, public C { … }
```

This ensures that only one copy of variables of the grandparent is inherited by the grandchild.

Q.7 a. What are abstract classes? How can a class be made abstract? Give example.

Answer:

Abstract classes: These are the classes that cannot be instantiated ie., we cannot create the objects of these classes. Generally abstract classes are used as a template on which a class hierarchy may be created. The base level classes in the hierarchy are typically too generalized to have any implementation and that is why they created as abstract classes.

A class becomes an abstract class if it contains one or more pure virtual functions. Pure virtual functions do not provide the body of the function.

All classes that are derived from an abstract class must provide the implementation for each of the pure virtual function declared in the base class. If not, the derived class would itself become abstract.

For example, following is the code for an abstract class A having a pure virtual function *fn*.

```
class A {
        public:
                void fn(int a) = 0;      // pure virtual function
                …
};
```

b. Explain *try..catch* and *throw* constructs with the help of an example.

Answer:

Exception handling is managed using the keywords try, catch and throw. Program statements that are to be monitored for exceptions are contained within the try block. If an exception occurs within the try block, it is thrown. The catch block specifies the code that is to be executed when an exception occurs. To throw an exception the throw keyword is used.

The general form of exception handling block is:

```
try{
        // code to be monitored
        throw exception
}
```

```
catch (datatype arg) { // exception handing code }
```

For example, the following code shows how to throw an exception if division by zero is attempted

```
cout << "\nEnter the numerator: ";
cin >> a;
cout << "\nEnter the denominator: ";
cin >> b;
try {
        if (b == 0)
                throw 0;
        cout << "a/b = " << a/b;
}
catch (int i) { cout << "Exception: Division by zero not allowed"; }
```

Q.8 a. What are templates and what is their use? Explain.

Answer:

A template is a footprint on which the functions and class definitions are based on. Templates work on generic data types and they are instantiated as per the requirement of the function or class.

The need to have templates is explained with the help of following example. Consider a function that finds the minimum of two integers having the following form:

```
int minimum(int a, int b) { … }
```

This may be overloaded to find the minimum of two floating point numbers. In this case, it will look like

```
float minimum(float a, float b) { … }
```

This may be extended for doubles also. All these implementations will have the same logic, except for the fact that they will operate on different data types. To help provide only one implementation, C++ provides the concept of templates. A template takes a generic data type and compiler generates the code based on this template depending on the current context that is the desired input data type. A template can be defined for the same having the syntax

```
template <class T>
T minimum(T a, T b) { … }
```

When a call will be made to this function as say minimum(x, y) then if the data type of x and y will be integer then system will invoke it as if it is defined for integers. Similarly, if the data type of x and y will be float then system will invoke it as if it is defined for floating point variables.

The benefit of templates lies in the fact that it helps in creating concise code avoiding definition of several overloaded functions.

b. Write a program to find the minimum of two values using templates. In main(), write calls to display its use on different data types.

Answer:

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

        template <class T>
        T minimum(T a, T b) {
                if (a < b)
                     return a;
                else
                     return b;
        }

        int main()          {
                int x = 10, y = 20;
                cout << "\nMin of " << x << ", " << y << " is " << minimum(x, y);
                float m = 1.23, n = 2.10;
                cout << "\nMin of " << m << ", " << n << " is " << minimum(m, n);
                return 0;
        }
```

Q.9 a. Write a program to display the contents of file on the screen.

Answer:

```
          #include <iostream>
        #include <conio.h>
        #include <fstream>
        using namespace std;
        int main() {
                     char                filename[30], c;
                     ifstream            inFile(filename);

                     cout << "\nEnter the name of the file: ";
                     cin >> filename;
                     if (!inFile) {
                                cout << "\nNot able to open the file";
                                getch();
                                exit(1);
                     }
                     while (!inFile.eof()){
                                inFile.get(c);
                                cout << c;
                     }
                     inFile.close();
                     return 0;
        }
```

b. With the help of examples, explain the use of the following flags in the *ios* class:

(i) scientific **(ii) boolalpha**
(iii) right **(iv) showpos**

Answer:

(i) scientific: used to display floating point numbers in exponential form.
Example:
```
cout.setf(ios::scientific);
cout << 100.0;
```
code will produce the output: 1.000000e+002

(ii) boolalpha: used to convert a bool variable to string such as true or false.
Example:
```
bool  b = true;
cout.setf(ios::boolalpha);
cout << b;
```
code will produce the output: *true* in place of 1 for true

(iii)right: used to display data right aligned. Effect can be seen when width is also set. Example:
```
cout.setf(ios::right);
cout.width(10);
cout << "abc";
```
code will produce the output: abc
Here 7 blanks appear before the text abc.

(iv)showpos: used to show leading positive sign in numeric output. Example:
```
cout.setf(ios::showpos);
cout << 100;
```
code will produce the output: +100

<u>Text Book</u>

Object Oriented Programming with C++, Poornachandra Sarang, PHI, 2004