**Q.2 a. With an example explain how I/O is achieved in C++.**

(9)

2 (a). The standard input device is the Kyboard and output device is the console or screen. In the iostream C++ library, cin is the standard input stream taking input from the Kyboard and cout is the standard output streams: cerr and clog specially designed to show error messages that can be directed to the standard output or to a log file.

Input (Cin)

Input streams use the extraction (>>) operator for standard types. The cin input stream can be used with the extraction operator >>. Extraction operator extracts the values of the identifiers from the input stream cin.

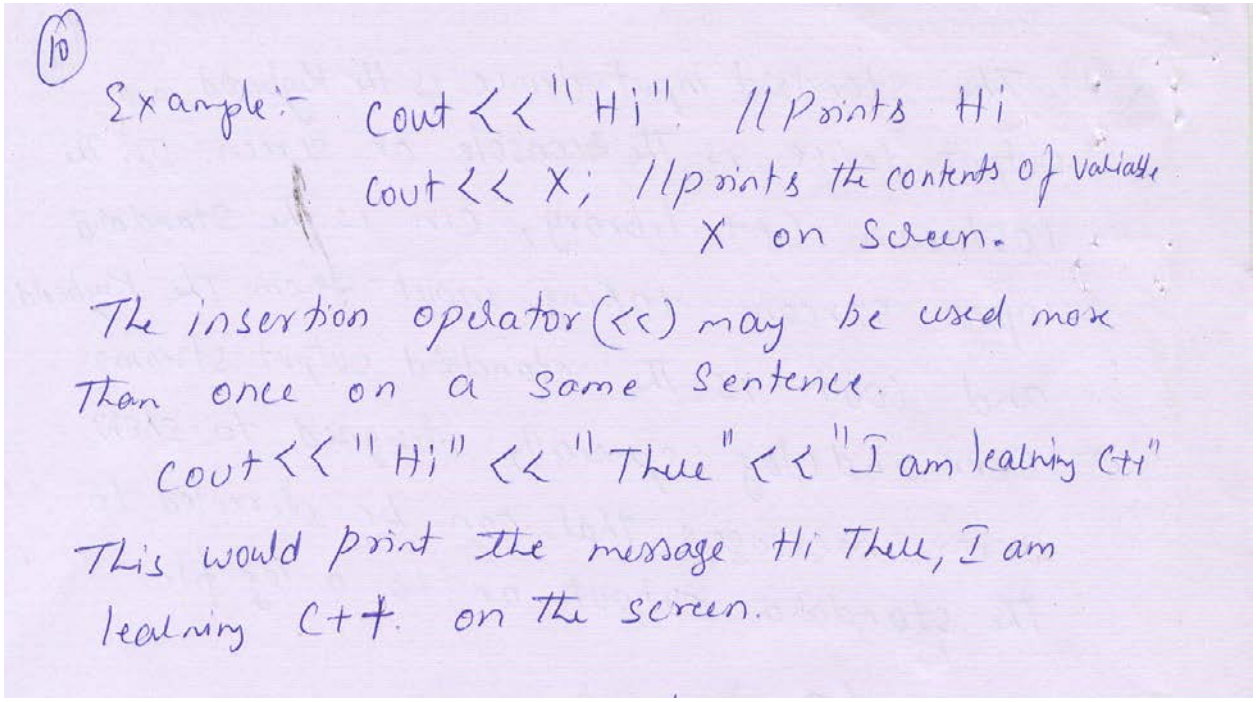
Ex:-

```
int count;
cin>> count;
```

cin can only process any input from the Key board once the newline or return Key has been pressed.

Output (cout)

output streams use the insertion (<<) operator for standard types. The cout stream can be used with the insertion operator <<. Insertion operator inserts the identifiers following the insertion operator to the output stream cout.




**b.Explain the following:**

**(i) Class and Object**
**(ii) Abstraction and Encapsulation**

2 (b) (i) class and object.

A class is a generalization of a structure in C. It is a construct for implementing a user defined type. once defined, such types may be conveniently used as the languages primitive types. The instances of a class is called objects. A class specifies the representation of objects and a set of operations that are applicable to such objects. The implementation details of the class are private to the class. The public interface of a class comprises of two categories of operations. The first category consists of accessor functions that return meaningful abstractions about the object's state. The second category consists of transformer functions that move an object from one valid state to another: For Example.

```
class Person
{ Private: char *name;
           short age;
  Public: void SetName(const char *);
          short GetAge();
};
```

Objects

```
Person P1, P2;
```

(ii)

(ii) Abstraction and Encapsulation

The object's data attributes define the current state of an object. The current state of the object can be manipulated only by calling the publicly accessible interfaces or methods. An object can only be manipulated through an interface that responds to a limited number of different kinds of messages or calling of methods. The internal structure of objects is hidden from the client. There remain some methods, which provide an interface between the service provider object and other service taker objects. The way of packaging an object's data members within the protective custody of its functions members or methods is called data encapsulation. With encapsulated data, the components of an object should not be accessible using the dot notation as it is possible in structs in C.

Encapsulation of data within objects offer several advantages. two of which are.

(i) modularity (ii) Information hiding.

Data abstraction: is a way of representation of abstract data types to expose the interface, not the implementation, where data is hidden inside the type by exposing the operations applicable.

**Q.3a. With an example explain how local transfer of control is achieved in C++.**

(12)

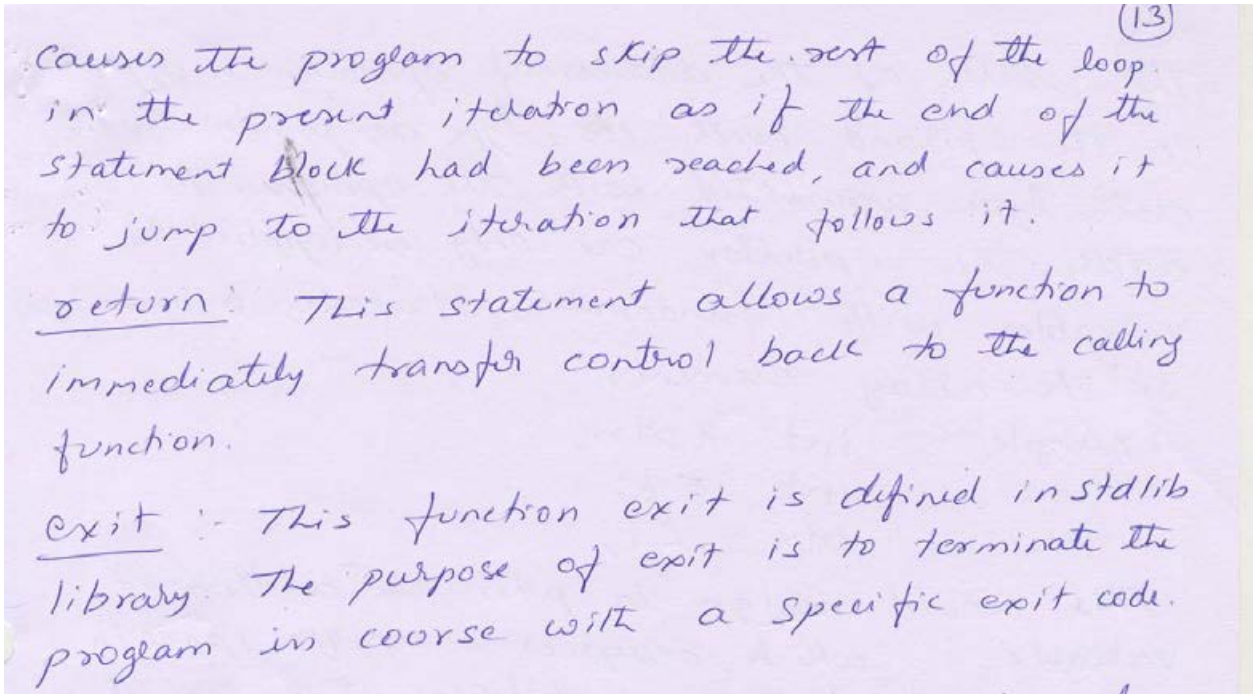
3 (a) Jump statements perform an immediate local transfer of control. They are four types:

* goto
* break.
* continue
* return.

goto: This statement performs an unconditional transfer of control to the named label as given in a labeled statement. The label must be in the current function body. This is not a recommended way of use, as it affects the structuredness of a program.

break: This statement is used to exit a looping or switch statement. It transfers control to the statement immediately following the looping statement or switch statement. The break statement terminates only the innermost loop or switch statement. In loops, break is used to terminate before the termination criteria evaluates to a zero value. In the switch statement, break is used to terminate sections of code usually before a case label.

continue: This statement forces immediate transfer of control to the loop-continuation statement of the smallest enclosing loop. The "loop-continuation" is the statement that contains the controlling expression for the loop. In a for loop, execution of a continue statement causes evaluation of iteration-part and then condition-part. The statement thus



(13)

Causes the program to skip the rest of the loop in the present iteration as if the end of the statement Block had been reached, and causes it to jump to the iteration that follows it.

return: This statement allows a function to immediately transfer control back to the calling function.

exit :- This function exit is defined in stdlib library. The purpose of exit is to terminate the program in coorse with a specific exit code.

**b.Briefly explain the following:**

**(i) Void Pointer**
**(ii) Address-of Operator**
**(iii) Indirection Operator**
**(iv) Invalidate a Pointer**

3 (b). (i) Void Pointer :- The void data type always represents an empty set values. one cannot declared a variable of type void. The only item that can be declared with the type Specifier void is a pointer. If a pointer's type is void *, the pointer can indicate any variable that is not declared with the const Keyword. A void pointer cannot be dereferenced. It can be converted into any other type of data pointer. and vice versa.



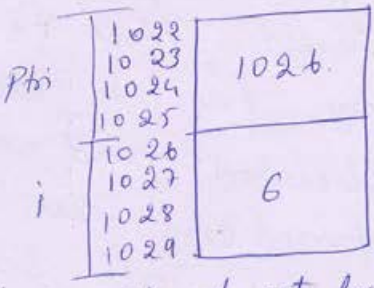
(ii) Address-of operator :- whenever a variable is declared, it occupies space in memory. The compiler decides where in memory the variable will be placed. The unary address-of operator (&) gives the address of its operand.

**Invalidate a pointer:-** There are some most common conditions that invalidate a pointer value or memory location of a valid item. Such as the pointer.

1. is a null pointer.
2. Specifies the address of some item, which no longer exists.
3. Specifies an address that is inappropriately aligned for the type of the item pointed to.
4. Specifies an address not used by the executing program.

Example:-

```
int *ptri;
int i = 6;
ptri = &i;
```




Here, the pointer variable ptri is stored at location 1022, integer variable i is stored at location 1026 having the value 6, and by virtue of assigning address of the variable i to ptri, that contains the value 1026, which is the memory location or address of the variable i. Now, if we write

```
i = *ptri;
```

it means, we would like to assign the indirection or dereferencing of the pointer ptri, i.e. the integer variable i is assigned the value of the item the pointer ptri refers to, i.e. the contents of the memory location 1026., i.e. 6. Null pointer can not be properly dereferenced.



236/2

**Q.4a. With an example explain inline function in C++.**

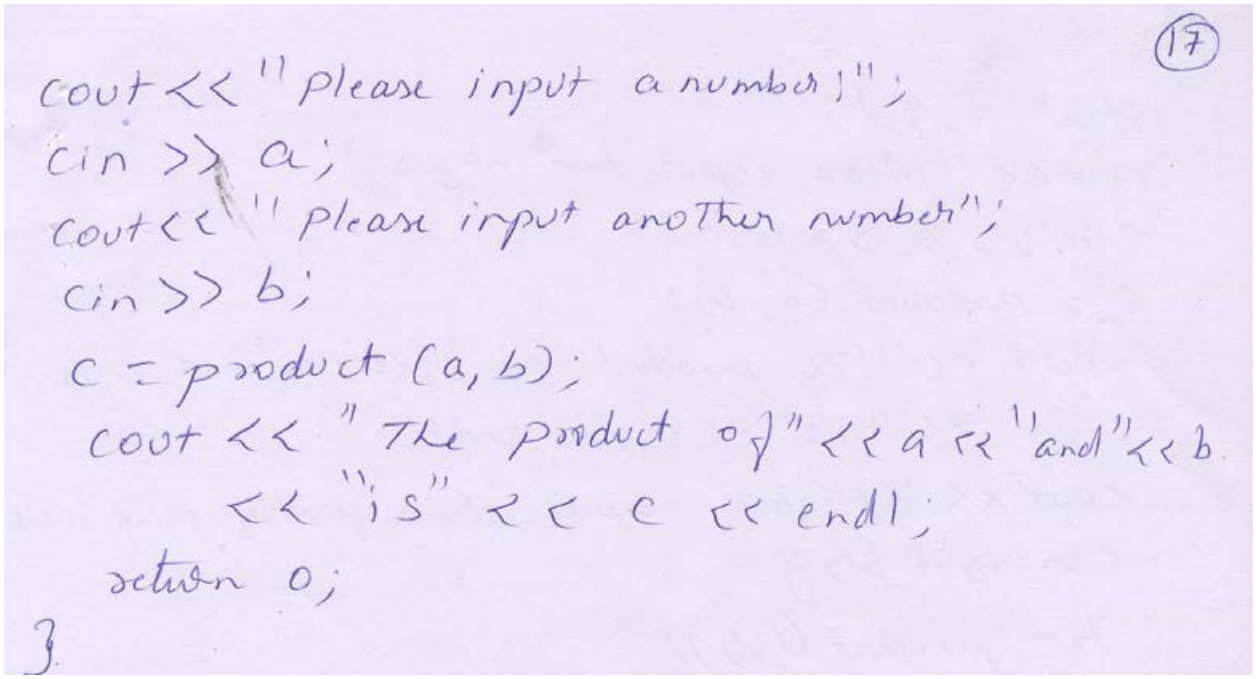
4 (a). Inline functions are used to reduced the overhead of a normal function call. The inline specifier is a suggestion to the C++ compiler that inline substitution of the function body is to be preferred to the usual function call implementation.

By inlining we mean that instead of transfering control to and from the function code segment, one may directly embed a copy of the function body whenever the function is called. That is function call will be replaced by function code. An inline function can be declared and defined simultaneously. Inline functions are best used for small functions which are sensitive to the function call overheads.

The inline keyword tells the compiler to substitute the code at its discretion within the function defination for every instance of a function call.

Example:-

```
#include<iostream.h>
inline int product(int x, int y)
{
    return x*y;
}
int main()
{
    int a, b, c;
```



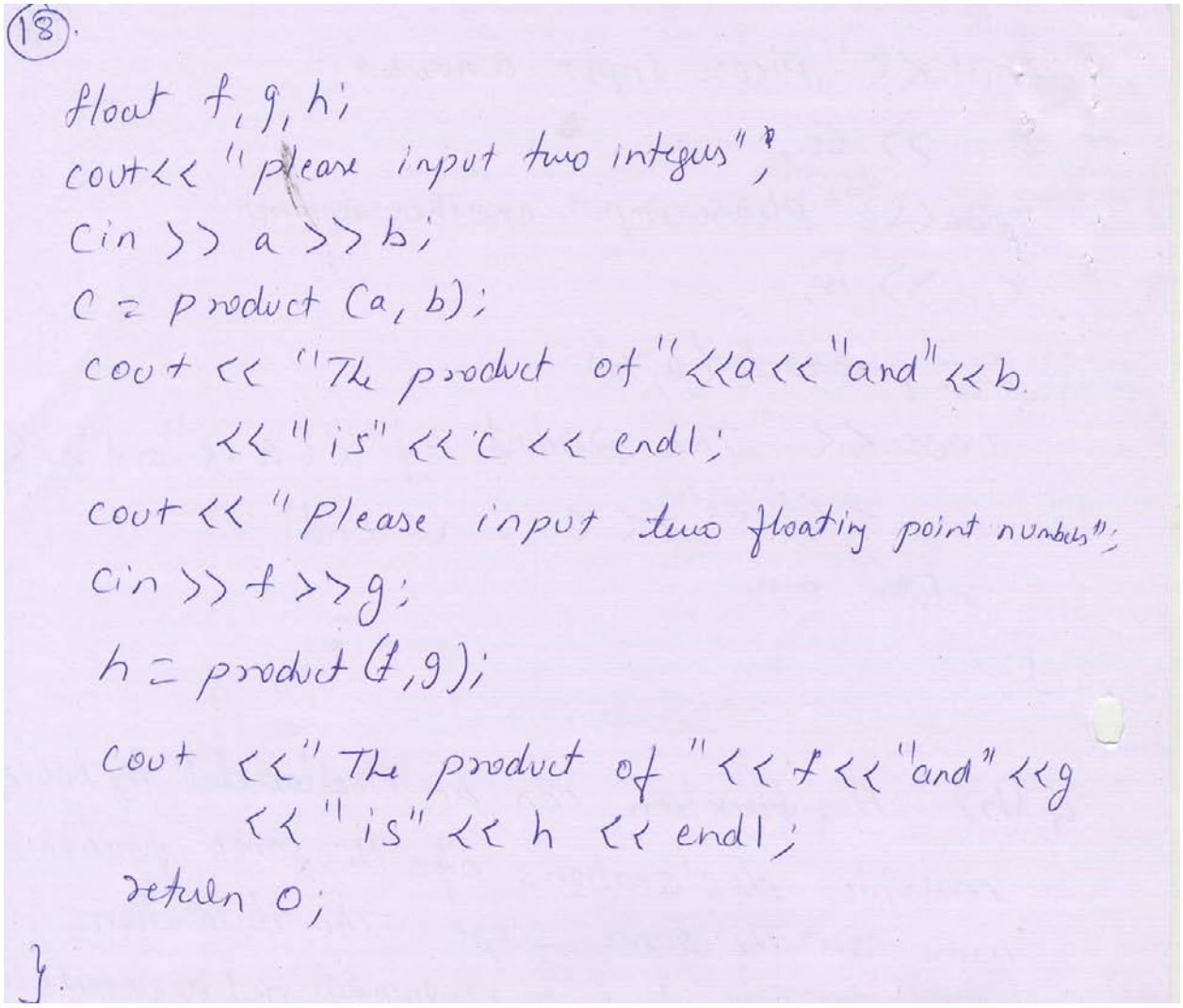
```
(17)
cout << "Please input a number:";
cin >> a;
cout << "Please input another number";
cin >> b;
c = product (a, b);
cout << "The product of" << a << "and" << b
     << "is" << c << endl;
return 0;
}
```

**b. Explain function overloading. Write a C++ program to demonstrate function overloading.**

4 (b) A function can be overloaded by having multiple declarations of the same function name in the same scope. The declarations differ in the type and number of arguments in the argument list. When an overloaded function is called, the types of the actual arguments are compared with the types of the formal arguments. Thus, it selects the correct function

Example :-

```
#include <iostream.h>
int product (int x, int y)
{
    return x*y;
}
float product (float x, float y)
{
    return x*y;
}
int main()
{
    int a, b, c;
```



```
(18).
float f, g, h;
cout<< "please input two integers";
cin >> a >> b;
c = product (a, b);
cout << "The product of "<<a<< "and" << b.
     << "is" << c << endl;
cout << "Please input two floating point numbers";
cin >> f >> g;
h = product (f, g);
cout <<" The product of " << f << "and" << g
     << "is" << h << endl;
return 0;
}.
```

**c. What is the difference between Return-by-value and Return-by-reference?**

4 (c). When many arguments may be sent to a function, only one may be returned from it. This is a limitation when you need to return more information. There are other approaches to returning multiple variables from functions. One is to pass arguments by reference and the other is to return a structure instead of a single data.

A return statement specifies the return value. The following code snippets show a function declaration, including the return statement.

```
int product (int i, int j)
{
    return i*j;
}
```

The function product() can be called

```
int a = 4;
b = 5
int answer = product(a, b);
```

In this example, the return statement initializes a variable of the returned type with the int value 20. The compiler checks the type of the returned expression against the returned type. It performs all standard and user-defined conversions as necessary.

References can also be used as return types for functions. The reference returns the lvalue of the item to which it refers. This allows one to place function calls on the left side of assignment statements.

Example:

```
#include <iostream.h>
int &max(int &x, int &y)
{
    return (x>y)? x: y;
}
int main()
{
    int a=5, b=6;
    int c = max(a, b);
    cout << "a=" << a << "b=" << b << "c=" << c;
    return 0;
```

**Q.5 a. What is a class in C++? Explain with an example.**

5 (a) CLASS

A class in C++ is a mechanism for creating user-defined data types. A class is also a mechanism to organize data and functions together in a same structure.

class means, state + Behaviour.

class Specification has two parts:

1) class declaration
2) Member Function definition.

class declaration indicates the types and scope of its members (data & functions).

Member function definition tells how the member functions within the class are implemented.

Polymorphism is supported through classes with virtual functions.

An object has the same relationship to a class that a variable has to a data type. An instance of a class is an object, rather than a variable. An object occupies space in memory, whereas a class does not.

A class is declared using the Keyword class, whose functionality is similar to one of the C Key word struct, but with the possibility of including functions as well.

(21)

Syntax:

```
class classname
{
    access_right 1:
        member 1;
    access_right 2:
        member 2;
    ...
} objectname;
```

Where, classname is the name for the class (user defined type) and the field objectname is optional containing one or more valid object identifiers. The declaration body (within the braces) may contain members, which may either be data or function declarations. The data members are called instance data and function members are called member functions. There can be also optional access right labels which, may be any of these three keywords.

Private, public, & protected.

Private members of a class are accessible only from other members of its same class or from its "friend".

Protected members are accessible, in addition to from members of its same class and friends, and also from members of its derived classes.

Public members are accessible from anywhere where the class is visible. This holds true till another access right label appears in the block.

(22) Example :-

```
class SimpleClass
{
    private:
        int IntegerData;
    public:
        void SetData(int d);
        {
            IntegerData = d;
        };
        void ShowData()
        {
            cout << "\n Data is" << IntegerData;
        };
};
```

**b. Differentiate between constructor and destructor.**

5 (b) Constructors and destructors are special member functions of classes that are used to construct and destroy objects. Construction of an object involves memory allocation and initialization for the object. Destruction of an object involves clean up and deallocation of memory for the object. Like all other members functions, constructors and destructors are declared within a class declaration. They can be defined inline or external to the class. Constructors can have default declarations. arguments.



There are some restrictions to constructors and destructors, which are as follows:

1. Constructors and destructors do not have return types nor can they return value.

2. Although constructors and distructor are also functions, one cannot use references and pointers on constructors and distructors, because one cannot take their addresses.

3. One cannot declare constructors with the Keyword virtual.

4. One cannot declare constructors and destructors as static, const, or volatile.

5. Unions cannot contain class objects that have constructors and destructors.

The constructor member function has, the same name as the corresponding class. The C++ run time system ensures that the constructor in a class, ~~which con~~ is called when an object instance is created. If no constructor is declared and defined for a class, compiler provided default constructor is used instead.

Destructor.

MODERATION-I

A destructor is a special member function with the same name as its class that is prefixed by a ~ (tilde). A destructor takes no arguments and has no return type (not even void). Destructor can be declared as virtual or pure virtual. Destructors are used to deallocate memory and do other cleanup for an object and its members when the object is destroyed.

Destructor is automatically called when an object goes out of scope, or when a pointer to object is deallocated using specific call to delete operators.

**Q.6a. Describe overloading of unary and binary operators.**

6 (a). Overloading of Unary Operators.

A unary operator can be overloaded by declaring a non member function that takes one argument, or a nonstatic member function of a class that takes no arguments. If an object is prefixed with an overloaded unary operator, its interpreted as the function call., eg. $-x$ is interpreted as : x.operator $-()$ or operator $-(x)$.

Example :-

```
class Fraction.
{
    Fraction operator -();
};
int main()
{
    Fraction x, y;
    y = -x;
    return 0;
}
Fraction Fraction :: operator -()
{
    Fraction tmp;
    tmp.num = -this -> num;
    tmp.denom = this -> denom;
    return tmp;
```

25

Overloading Binary Operators:

One can overload a binary operator by declaring a non member function that takes two arguments, or a non static member function that takes one argument. When one uses an object with an overloaded binary operators you can interpret the operator function call x*y as:

```
x.operator(y);
```

or

```
operator*(x,y);
```

depending on the declarations of the operator function.

```
class Fraction
{
    ...
    Fraction operator*(const Fraction &);
};
int main()
{
    Fraction x(3,4);
    Fraction y=10;
    x*y;
    return 0;
}
```

**b. With the help of an example explain the use of cast operator in C++.**

6 (b) cast operator

The cast operator is used for explicit type conversions.

```
class Fraction
{
    ...
    int num;
    int dnom;
  Public:
    operator float ();
};
Fraction :: operator float ()
{
    return ((float) this -> num) / ((float)
                                    this -> dnom);
}
```

A class can have multiple cast operators defined for a class, so the objects belonging to the class can be typecast to similar or equivalent classes or data types.

**Q.7 a. Explain (i) Inheritance (ii) Multiple inheritance**

7(a). Inheritance is one of the most important notions of OOP. Inheritance allows creating classes from existing classes through an IsA relationship. When members are inherited, they can be used in the derived class as if they are members of the derived class that inherits them, provided, the base class permits the derived class to do so. If class B inherits from class A, then B is called the Specialization of generalized class A. class A is called the base or Superclass and class B is called the derived or Sub class. A derived class inherits the properties that include data and function members of its Base class.

(27)

Thus, all the code, which is written to work with objects of the base class, will work with the objects of derived classes. This makes class definitions for subclasses much smaller and neater and makes it possible to write code, which works with objects of different types, on the right level of generality. One can also add extra data members as well as member functions to the derived class. One can also modify the implementation of existing member functions or data by overriding base class member functions or data in the newly derived class. A class can be derived from one or more base classes.

Example:-

```
class super
{   public:
        int a;
        int b;
};

class sub : public super
{   public:
        int b;
        int c;
};

void f()
{
    sub d;
    d.a = 1;
    d.b = 2;
    d.super::b = 3;
    d.c = 4;
    super *ps = &d;
}
```

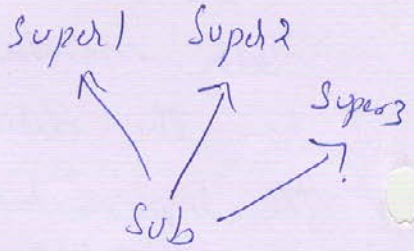
(28).

(ii) multiple Inheritance.

multiple Inheritance allows declaring a derived class or subclass that inherits properties from more than one base class or super class.

Ex: super1, super2, & super3 are direct base classes of class sub. The order of derivation is not significant

```
class super1 { };
class super2 { };
class super3 { };
class sub : public super1,
            public super2,
            public super3
{
```




**b. Explain friend function, with example**

7 (b) Friend.

If a class X declares a friend in order to grant the access privileges same as that of members of class X, it means a friend of class X can access all the members of the class X which has granted friendship. Friendship is granted by a class X to another class Y, or to a member function of another class Y, or to a single isolated non-member function. However, neither a constructor nor a destructor of a class can be a friend function to another class. A class X grants friendship, by explicitly declaring

the friend inside the structure declaration of class X. A class Y must be defined before any of the member function of class Y can be granted friendship by another class X. This in effect, opens a small hole in the protective shield of data abstraction of the class, and so it should be used sparingly with extreme caution.

Example:-

```
#include<iostream.h>
class base
{
    int val1, val2;
    Public:
        void get()
        {
            cout<<"Enter two values:";
            cin>>val1>>val2;
        }
        friend float mean(base ob);
};
float mean(base ob)
{
    return float(ob.val1+ob.val2)/2;
}
Void main()
{
    base obj;
    obj.get();
    cout<<"In mean value is:"<<mean(obj);
}
```

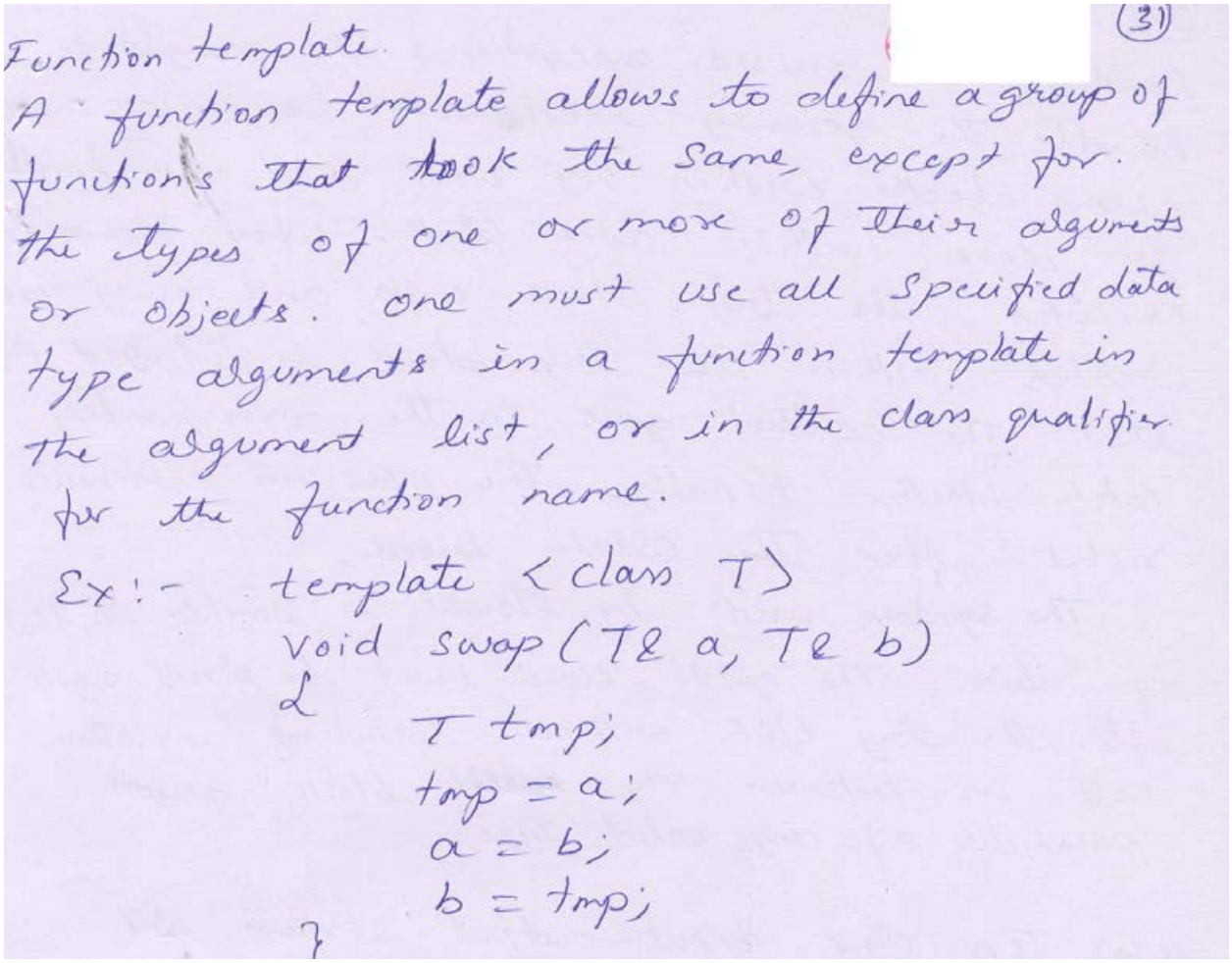
**Q.8 a. Differentiate between a template and a marco. Explain class template and function template.**

8 (a) Template :- C++ templates provide a way to reuse source code as opposed to inheritance and composition which provide a way to reuse object code. A template definition uses one or more data types as arguments and defines a class or function based on the argument. which are data types

class template :- The relationship between a class template and an individual class is like the relationship between a class and an individual object. An individual class defines how group of object instances can be constructed from the individual class, while a class template defines how a group of classes can be generated from the template classes.

Ex :-

```
template < class T >
class Array.
{
    protected:
        int length;
        T * arr;
    public:
        Array (int size);
        virtual ~Array();
        T& operator [] (int index);
};
```



(31)

Function template.

A function template allows to define a group of functions that took the same except for the types of one or more of their arguments or objects. one must use all specified data type arguments in a function template in the argument list, or in the class qualifier for the function name.

Ex:-

```
template <class T>
void swap(T& a, T& b)
{
    T tmp;
    tmp = a;
    a = b;
    b = tmp;
}
```

**b. Illustrate the usage of nested try block. Is it necessary that number of catch blocks should be equal to the number of try blocks? Justify.**

Ans Page 335 of Text book

**Q.9 a. What is STL? Why should they be used? Explain the different components of STL.**

Ans Page 361 of Text book

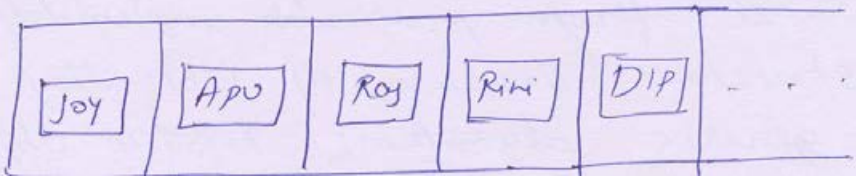
**b. What are the different forms of get () function of istream class? Illustrate the uses by citing proper examples.**

9 (b) STL components are:

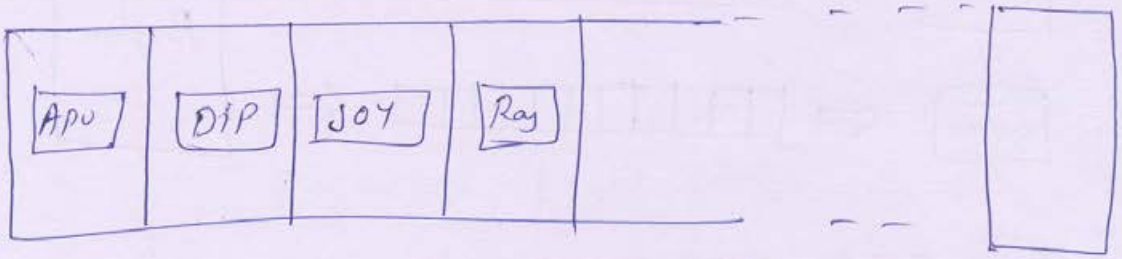
- Algorithm
- container
- Iterator
- Function object
- Adapter.

containers are of two types:

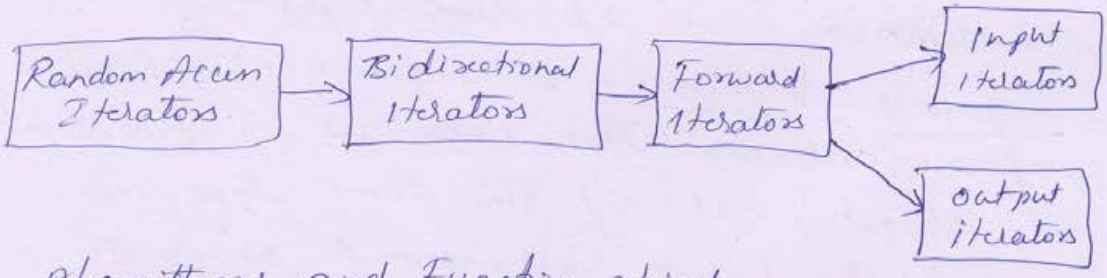
Sequence container:- A sequence is a kind of a container that organizes a finite set of objects, all of the same type, into a strictly linear allangement as in the figure. STL provides three basic kinds of sequence container: Vectors, Lists, & Deque.



Associative Containers:- They provide an ability for fast retrieval of data based on Keys. Here elements are sorted so fast binary Search is possible for data retrieval. STL provides four basic Kinds of Associative Containers. set, Map, multiset and multimap.



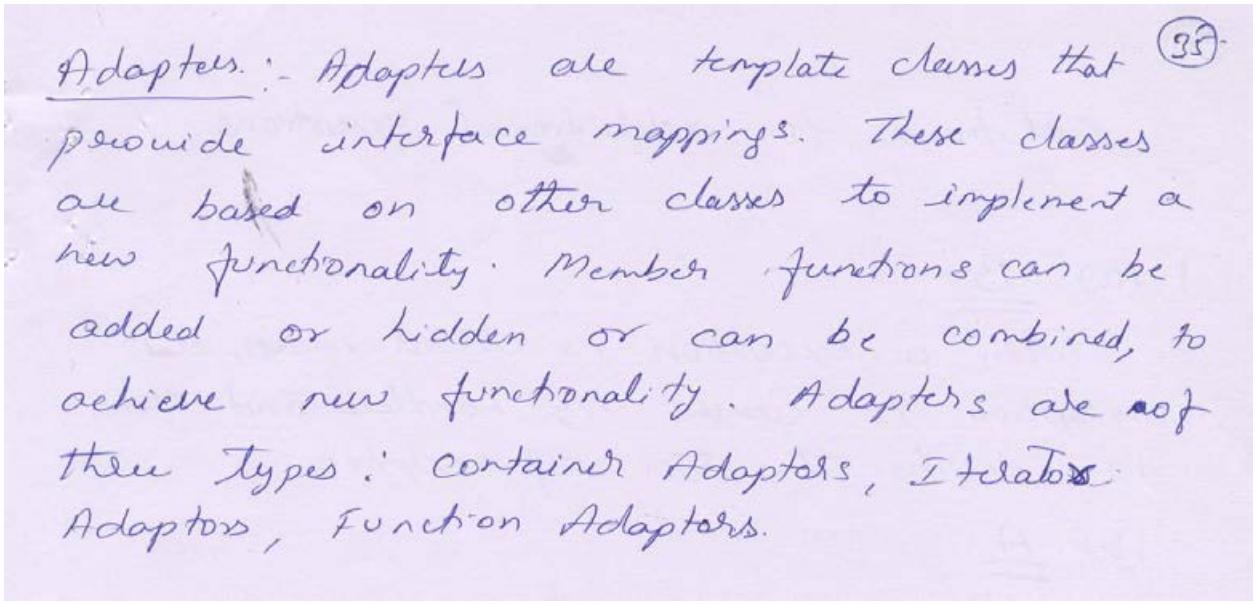

Iterator: Iterators are generalization of pointers that allow a programmer to work with different data structures (containers) in a uniform manner. There are five categories of of iterators.




Algorithms and Function objects.

All the algorithms provided by the library are parameterized by iterator types and are so separated from particular implementations of data structures. Because of that they are called generic algorithms. Function object is a class that has the function-call operator defined.

MODERATION-I




**Text books**

1. **C++ and Object-Oriented Programming Paradigm, Debasish Jana, 2nd Edition, PHI, 2005**