b) i.) a.) A constructor is called at the time of declaration of an object

ii.) a.) Pure virtual function

iii.) classname

iv.) The class which have no object is known as abstract class.

v.) 18

vi.) ~~d.) Two pointers of~~ a

vii.) b) 1

Scanned by CamScanner

9)

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

Class shape
{
  int w, h;
 Public
    Shape area()
{
  Cout << "Enter the value of width and hight";
  cin >> w >> h;
  {
   Area (int);
   Area (int, int);
  };

 Class square : Public shape
{
   Area (int);
 }
 Class rectangle : Public shape
{
   Area (int, int);
 }
     Area (int)
     {
        return (h * h);
     }
Area (int, int)
 {
   return (h * w);
}
 Int main
 {
```

Scanned by CamScanner

```
Shape S, R ;

S. Area ( h ) ;
R. Area ( h, w ) ;
getch ( ) ;
}
```

Scanned by CamScanner

3) a) Inheritance is defined as the process of creating a new class from one or more existing classes. The existing class is known as base class or parent class or super class whereas the newly created class is known as derived class or child class or sub class.

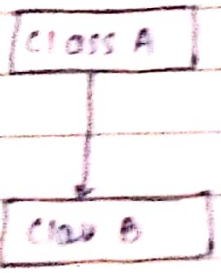
Types of Inheritance

i) Single Inheritance

In this process, a new class is created using an existing class.

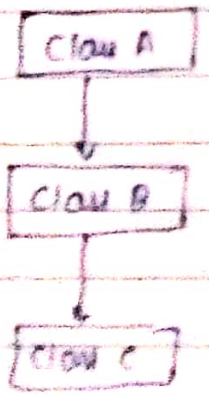
The syntax is as below:

class <derived class name> : <access specifier> <base class name>



ii) Multilevel Inheritance

When the process of single inheritance is extended upto minimum 2 levels, the concept is known as multilevel inheritance.

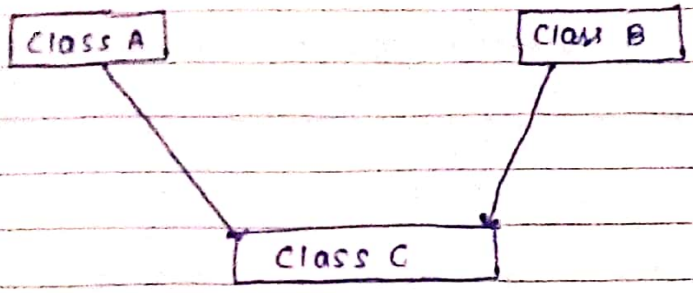


The process of multilevel inheritance involves a new terminology known as "Intermediate class".

Scanned by CamScanner

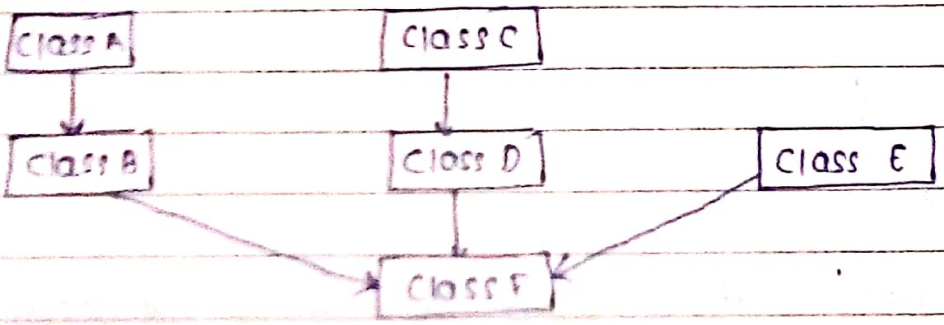
iii) Multiple Inheritance

When a new class is derived from more than one base class, the process is known as multiple inheritance. Here the minimum number of base class is 2.



iv) Hybrid Inheritance

The proper combination of one or more type of inheritance happens together is known as hybrid inheritance.



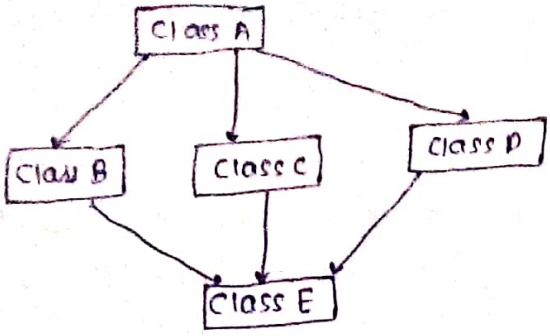
v) Hierarchical Inheritance

In this type of inheritance, more than one derived class are pr
This inheritance is little bit complicated as it involves lots of bas
classes and derived classes. Due to this reason, this inheritance i
not frequently used.

vi) multipath Inheritance

In this type of inheritance, a class is derived from more tha
two or more classes, which are derived from the same base class

Scanned by CamScanner



b) Inline function is a function that is expanded in line when it is called. when the inline function is called, whole code of the inline function gets inserted or substituted at the point of inline function call. This substitution is performed by the C++ compiler at compile time. The syntax for defining the function inline is

```
inline return-type function name ( parameters )
{
    // function code
}
```

Advantages:

i) It avoids the overhead of calling the actual function.

ii) It proceeds faster execution of program.

iii) It reduces space as no separate set of instructions in memory is written.

Disadvantages:

i) It increases the size of the executable file.

ii) Compiled program can take up more room in memory.

iii) It is not useful for embeded system where large binary size is not preferred at all due to memory size constraints.

Scanned by CamScanner

4.)

```
#include<iostream.h>
#include <conio.h>
int main()
{
  clrscr();
  int n, sum = 0;
  cout << " Enter an Integer number" << endl;
  cin>>n;
  for (int i=1; i <=n; i++)
  {
     sum = sum + i;
  }
  cout << " sum = " << sum;
  getch();
  }
```

Scanned by CamScanner

5.) a.) what is polymorphism. Explain different types of polymorphism with example.

Ans. Polymorphism is the process in which various forms of a single function can be defined and utilized by different objects to perform similar types of operation.
In other words,
Polymorphism refers to codes, operations or objects that behave differently in different contexts.

Types of Polymorphism

C++ provide two different types of polymorphism
- Run-time / Dynamic polymorphism
- Compile-time / Static polymorphism

In Run time polymorphism, the appropriate member function could be selected while the program is running. And, object is bound to the function call only at the run time.

Scanned by CamScanner

Virtual function is an example of dynamic polymorphism.
It is used in situation, when we need to invoke derived class function using class pointer. Giving new implementation of derived class method into base class and the calling of this new implemented function with base class object is done by making base class function as virtual function. This is how "Runtime Polymorphism" is achieved.

Compile Time Polymorphism
In this method, object is bound to the function call at the compile time itself.
Function overloading and operator overloading are perfect example of compile time polymorphism.
Function overloading is a programming concept that allows programmers to define two or more functions with the same name and in the same scope.
And, operator overloading is defined as the capability to relate the existing operator with a member function so that the resulting operator function may be used with the objects of its class without changing the nature of the operator.

Scanned by CamScanner

b)

```
#include<iostream.h>
#include<conio.h>
class Num
{   int n1, n2, n3;
  Public:
     Num (int x, int y, int z)
     {   n1 = x;
         n2 = y;
         n3 = z;
     }
     void show();
     friend void operator ++(Num);
};
void Num ::show()
{   cout<<"n1="<<n1<<endl<<"n2="<<n2<<endl<<"n3="
                <<n3<<endl;
}
void operator ++(Num o1)
{
    o.n1 = o1.n1 + 5;
    o1.n2 = o1.n2 + 10;
    o1.n3 = o1.n3 + 15;
cout<<"n1"<<o1.n1<<endl<<"n2="<<o1.n2<<endl<<"n3="
     <<o1.n3<<endl;
}
 void main()
 {
```

Scanned by CamScanner

```
clrscr();
Num obj(10,20,30);
cout<<" The entered numbers are as below."<<endl;
obj.show();
cout<<" After execution of operator function: "<<endl;
operator ++(obj);
getch();
}
```

Scanned by CamScanner