UNIT-3

* Access specifier :- There are three access specifier - public, private and protected. These access specifier define how the member of the class can be access ofcource any member of a class is accessible within that class inside any member function of that same class.

There are 3 types of access specifier -

1- Public - The member declare as a public are accessible from outside the class through an object of the class.

2- Protected - The members declared as a protected are accessible from outside the class but only if a class derived from it.

3- Private - These members are only accessible from within the class, no outside access is allowed.

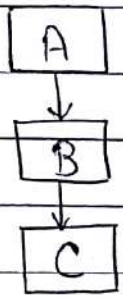
(13) Example of protected access specifier -

```
#include <iostream.h>
#include <conio.h>
class Shape
{
public:
void setwidth(int w)
```

```
{
width = w;
}
void setheight(int h)
{
height =h;
}
protected:
int width;
int height;
};
class Rectangle : public Shape
{
public:
int getarea()
{
return (width * height);
}
};
int main()
{
Rectangle R;
R.setwidth(6);
R.setheight(8);
cout<<"Area of rectangle :"<<R.getarea()<<endl;
getch();
return 0;
}
```

Output :- Area of rectangle : 48

* Multilevel Inheritance :- In multilevel inheritance, the class inherit the feature of another derived class.
In below diagram, class B and class A are as a parent classes depending on the relation. The level of inheritance can be extended any level.



(19) Example of multilevel inheritance -

```
class A
{
public:
int x;
void getdata()
{
cout << "Enter value of x:";
cin >> x;
}
};
class B : public A
{
public:
int y;
void readdata()
```

```
{
cout<<"Enter value of y:";
cin>>y;
}
};
class C : public B
{
private:
int z;
public:
void indata()
{
cout<<"\n Enter value of z:";
cin>>z;
}
void product()
{
cout<<"\n Product="<<x*y*z;
}
};
int main()
{
C a; // object of derived class
a.getdata;
a.readdata;
a.indata;
a.product;
getch();
return 0;
}
```

Output :- Enter value of x : 5
Enter value of y : 5
Enter value of z : 5
Product = 125.

* Hierarchical Inheritance :- In this type of inheritance more than one sub-class is inherited from a single base class. More than one derived class is created from a single base class.

Example :

```
# include < iostream.h >
# include < conio.h >
class A
{
public :
int x, y ;
void getdata ()
{
cout << "\n Enter value of x and y" ;
cin >> x >> y ;
}
} ;
class B : public A
{
public :
void product()
{
cout << "\n Product = " << x * y ;
```

```
}
};
class C : public A
{
public:
void sum()
{
cout<<"\n sum = "<<x+y;
}};
int main()
{
B obj1;
C obj2;
obj1.getdata();
obj1.product();
obj2.getdata();
obj2.sum();
getch();
return 0;
}
```

Output :-

```
Enter value of x and y:
6
5
Product = 30
Enter value of x and y:
6
6
sum = 12
```

* Polymorphism :- Polymorphism is the ability to use an operator or method in different ways. Polymorphism gives different meaning or function to the operator or methods.

Types of polymorphism — There are two types of polymorphism —

1- Compile time polymorphism
2- Run time polymorphism

1- Compile time polymorphism —
(i) Function overloading
(ii) Operator overloading

2- Run time polymorphism —
Function overriding / Virtual function

* Example of function overloading :-

```
class printdata
{
public :
void print (int i)
{
cout << "Print int :" << i << endl;
}
void print (double f)
{
cout << "Print float :" << f << endl;
```

```
}
void print (char *c)
{
cout <<" Print character :"<<c<< endl;
}};
int main()
{
printdata pd;
pd. print(5);
pd. print (500.263);
pd. print ("Hello C++");
getch ();
return 0;
  }
```

Output :-

Print int : 5
Print float : 500.263
Print character : Hello C++

* Function Overloading :- Function overloading refer to using the same thing for different purpose. This process of using two or more function with the same name but different in the signature, is called function overloading.

In term of type of argument.
In term of number of argument.

* Example of function overloading in term of number of argument -

```
class printdata
{
public:
void print (int i)
{
cout <<" Print int :" << i << endl;
}
void print (int b, int c)
{
cout <<" Print int:" <<b<<c << endl;
}
void print (int a, int b, int c)
{
cout <<" Print int:" <<a<<b<<c << endl;
}};
void main ()
{
printdata pd;
pd. print (5);
pd. print (6,6);
pd. print (7,8,9);
getch ();
}
```

Output :-

```
Print int: 5
Print int: 6 6
Print int: 7 8 9
```

* Function overriding — If derived class function name is same as that of base class member function with same argument then it is called as a function overriding.

```
class A
{
public:
void m1()
{
cout<<" M1 of base";
};
class B: public A
{
void m1()
{
cout<<" M1 of derived";
}};
int main()
{
 B b1;
 b1.m1;
 getch();
 return 0;
}
```

Output :- M1 of derived

* Operator overloading :- C++ also provide option to overload operators.
For ex - we can make the operator '+' string class to concate two strings.
We know that it is the addition operator whose task is to add operands. So a single operator when place between integer operand add them and when place between string operand concate them.

// Using operator with built-in data type-

```
main()
{
int n1 = 5                    Output - 15
int n2 = 10
cout << " n1+n2;
}
```

Using operator with user define data type-

```
class test
{
int num;
public:
test()
{
num = 1;
}
};
```

```
main()
{
test t1,t2;
cout<< t1+t2;
  }
```

Ouput - Error

-
```
class test
{
int num;
public:
 test()
 {
num=0;
 }
void operator++()
 {
++num;
 }};
main()
 {
test t1 {
 ++t1;
 }}
```