08/06/'22.

* FUNCTION OVERLOADING

- This happens when two or more functions in a single program have the same name and same return type.
- These functions can be differentiated by
  1) number of arguments
  2) argument name
  3) data type of arguments.

eg) A program can have the following functions:

```
void Copy ();
void Copy (int a);
void Copy (float a);
void Copy (int a, int b);
```

=>
```
#include <iostream.h>
void sum();
void sum(int,int);
Int main()
{
    int x=12, y=13;
    sum();
    sum(x,y);
    return 0;
}
void sum()
{
    int a=2, b=3;
    cout << "Sum = " << a+b;
```

```
}
void sum(int a, int b)
{
    cout<<"\nSum = "<<a+b;
}
```

⊙ Output:

Sum = 5

Sum = 25

13/06/'22

Date
Page 62

* CONSTRUCTORS:

1. Constructor is a special function having the same name as class name.
2. A constructor has no return type.
3. A constructor may or may not have a parameter/argument.
4. A constructor is called when an object is initialized.
5. A class can have more than one constructor.

⇒ (Not using constructors)

```
#include<iostream.h>
class Add
{
    private:
        int a,b;
    public:
        void getdata()
        {
            cout<<" Enter a,b :";
            cin>>a>>b;
        }
        void sum()
        {
            cout<<" Sum = "<<a+b;
        }
}a1;
int main()
{
    a1.getdata();
```

Date
Page 63

```
    a1.sum();
    return 0;
}
```

⇒

```
#include<iostream.h>
class Add
{
    private:
        int a,b;
    public:
        Add()
        {
            cout<<"Enter a,b:";
            cin>>a>>b;
        }
        void sum()
        {
            cout<<"Sum = "<<a+b;
        }
}a1;
int main()
{
    a1.sum;
    return 0;
}
```

⊙ Output:

```
Enter a, b: 2
3
Sum = 5
```

(Same output for both programs).

17/06/'22.

Date Page 64

* PARAMETERIZED CONSTRUCTORS :

⇒
```
#include<iostream.h>
class Add
{
        int a,b;
    public:
        Add()
        {
            cout<<"Enter a,b:";
            cin>>a>>b;
            cout<<"\nSum = "<<a+b;
        }
        Add (int x, int y)
        {
            cout<<"\nSum = " x+y;
        }
};
int main()
{
    Add a1, a2(5,7);
    return 0;
}
```

⊙ Output :

```
Enter a and b: 3
4
Sum = 7
Sum = 12
```

Here, 5 and 7 will be passed as the values of x and y

Date Page 65

⇒
```
#include<iostream.h>
class Add
{
    int a,b;
    public:
        Add(int x, int y)
        {
            cout<<" Sum = "<<x+y;
        }
        Add()
        {
            cout<<"Enter a and b:";
            cin>>a>>b;
            cout<<"\nSum = "<<a+b;
        }
};
int main()
{
    int c,d;
    cout<<"\nEnter c and d:";
    cin>>c>>d;
    Add a1, a2(c,d);
    return 0;
}
```

(Here we can use a and b instead of c and d. These a and b will be different from those defined in class).

⊙ Output:

```
Enter c and d: 1
2
Enter a and b: 3
4
Sum = 7 (Adds a and b)
Sum = 3 (Adds c and d)
```

* CONSTRUCTOR OVERLOADING

Q. Write a program to find area of square, rectangle and circle using concept of constructor overloading

→
```
#include<iostream.h>
#include<iomanip.h>
class Area
{
  public:
    int s, r;
    float l, b;
    Area(int s)
    {
      cout<<"\n The area of the square is :"<<
            s*s;
    }
    Area( float l, float b)
    {
      cout<<"\n The area of the rectangle is :"<<
            l*b;
    }
    Area (int r)
    {
      cout<<" The area of the circle is :"<<
            3.14*r*r;
    }
};
int main()
{
  cout<<"Enter the side of the square:";
```



```
  cin>>s;
  cout<<"\nEnter the length and breadth of the
        rectangle:";
  cin>>l>>b;
  cout<<"\nEnter the radius of the circle:";
  cin>>r;
  Add Area a1(s), a2(l,b), a3(r);
  return 0;
}
```

⊙ Output:

```
Enter the side of the square 2
Enter the length and breadth of the rectangle: 3
5
Enter the radius of the circle : 7
The area of the square is : 4
The area of the rectangle is : 15
The area of the circle is : 154
```

24/06/'22.

* CONSTRUCTOR WITH DEFAULT ARGUMENT:

CASE 1: If we create a constructor without any arguments then it is known as default constructor.

eg)
```
#include<iostream.h>
class Add
{
    int a,b;
public:
    Add()
    {
        cout<<"Enter a and b:";
        cin>>a>>b;
        cout<<"\nThe sum is "<<a+b;
    }
};
int main()
{
    Add a1;
    return 0;
}
```

⊙ Output:

```
Enter a and b: 2
3
The sum is 5
```


CASE 2: Constructor with default arguments:

eg)
```
#include<iostream.h>
class Add
{
public:
    Add(int a=2, int b=3)
    {
        cout<<"\nThe sum is "<<a+b;
    }
};
int main()
{
    Add a1, a2(8,7);
}
```

⊙ Output:

```
The sum is 5
The sum is 15
```

* INHERITANCE : The process of inheriting the properties of one class to another class is known as inheritance. (Private access specifier is not shared)
Base class and Derived class
Parent class and Child class

⊙ Benifit of Inheritance - Reusability of code.

• There are 5 types of inheritance :

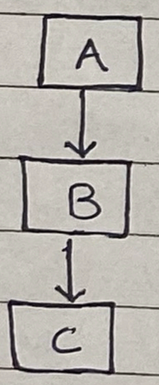
1) Single Inheritance :

In single inheritance, there is only one parent class and one child class.

A : parent class
B : child class

2) Multi level Inheritance :

In multi level inheritance there are multiple parent classes and multiple child classes.



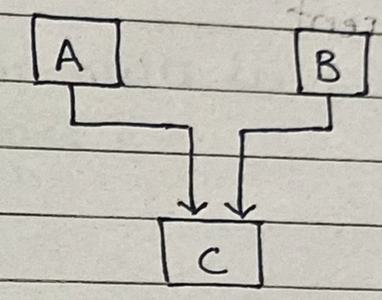
A : parent class
B : child class and parent class
C : child class.

Class C can inherit the properties of Class A indirectly through Class B.

Class B is the child class of Class A and the Parent class of Class C.

3) Multiple Inheritance :

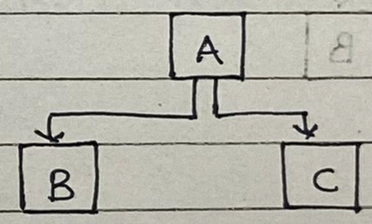
In multiple inheritance, one child class inherits from multiple parent classes.



A : parent class
B : parent class
C : child class

4) Hierarchial Inheritance :

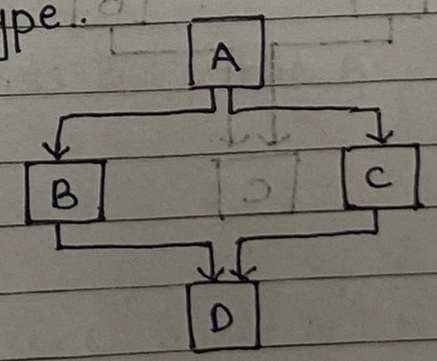
In hierarchial inheritance multiple child classes inherit the property of one parent class.



A : parent class
B : child class
C : child class

5) Hybrid Inheritance :

In hierarchial inheritance more than two inheritance are combined. The inheritance may or may not be of the same type.



Inheritances :-
Multiple
Multi-level
Hierarchial

⊙ Syntax:

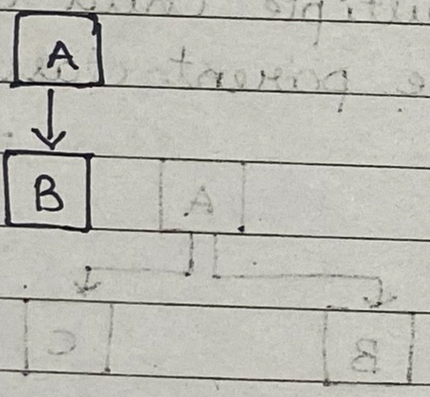
⇒ For single inheritance -

Done when declaring the child class.

```
Class child : visibility parent
{

};
```

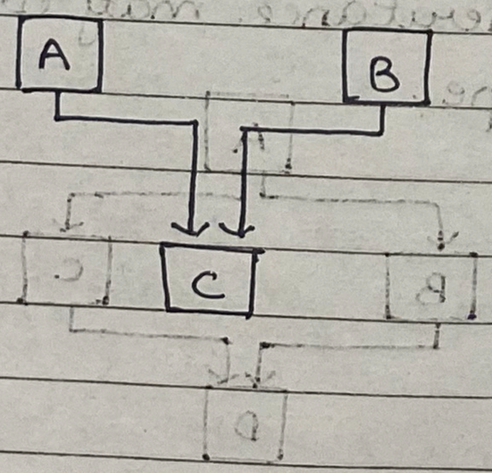
This is done only after the parent class is declared. Hence, the syntax becomes,

```
Class A
{

};
class B : public A
{

};
```



⇒ For multiple inheritance,

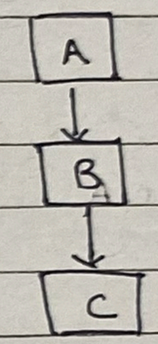
```
class A
{

};
Class B
{

};
```




```
class C : public A, public B
{

};
```

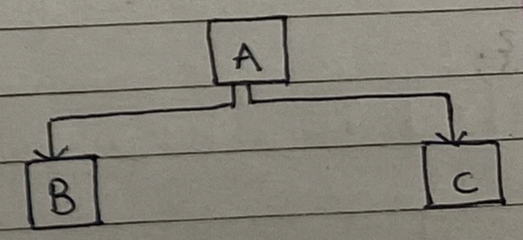
⇒ For multi level inheritance,

```
class A
{

};
class B : public A
{

};
class C : public B
{

};
```



⇒ For hierarchial inheritance,

```
class A
{

};
class B : public A
{

};
class C : public A
{
```

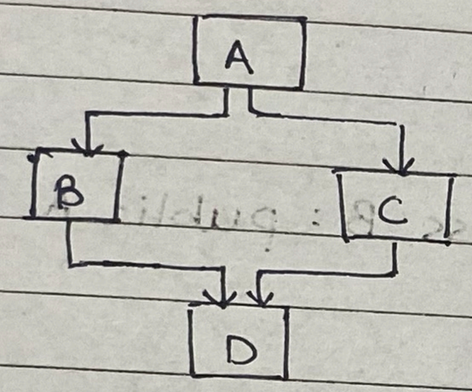


};

⇒ For hierarchial inheritance, we can give different visibility to different child classes.

⇒ For hybrid inheritance,

```
class A
{

};
class B : public A
{

};
class C : public A
{

};
class D : public B, public C
{

};
```



* Example :

```
#include<iostream.h>
class Data
{
    private:
        int a, b;
    public:
        void getdata()
        {
            cout<<"Enter a,b :";
            cin>>a>>b;
        }
};
class Sum : private Data
{
    public:
        void add()
        {
            getdata();
            cout<<a+b;
        }
};
int main()
{
    Sum s1;
    s1.add();
    return 0;
}
```

- This code will show error as 'a' and 'b' are undeclared in class Sum and are private in class data(they cannot be inherited in class Sum).
- If we declare "int a,b;" in class Sum, the program will run but garbage value will be displayed.

* VISIBILITY TABLE :

Base class Visibility (Access Specifier)	Derived class visibility		
	Public	Private	Protected
Public	Public	Private	Protected
Private	Not allowed to inherit	Not allowed to inherit	Not allowed to inherit
Protected	Protected	Private	Protected

① A public : int a
↓
B class B : public A
↓
C class C : public B

This is allowed.
Class C can inherit the properties of class A indirectly because its visibility is public in class B.

② A public : int a;
↓
B class B : private A
↓
C class C : public B (Not allowed)

Here we cannot access the properties of class A in class C as the visibility in class B is private.

③ A public : int a;
↓
B class B : protected A
↓
C class C : protected B (Allowed)

Here, we can use the properties of class A inside class B and class C, but not inside main() function because their visibility is protected.

* Example:

```
#include <iostream.h>
class A
{
  public:
      int a;
};
class B : public A
{
  public:
      int b;
};
int main()
{
  B b1;
  cout<<" Enter a and b:";
  cin>>b1.a>>b1.b;
  cout<<"\n Sum = "b1.a+b1.b;
  return 0;
}
```

- This code will work without any error.
If we change the visibility from public to private, error message will be generated.

* Example:

```
#include<iostream.h>
class A
{
private:
      int a;
};
class B : public A
{
  public:
      int b;
};
int main()
{
  A a1;
  B b1;
  cout<<"Enter a:";
  cin>>a1.a;
  cout<<"\nEnter b:";
  cin>>b1.b;
  cout<<"\n Sum = "<<a1.a+b1.b;
  return 0;
}
```

- This code will generate an error message because the variable 'a' is declared private inside the class. main() function can not access it. It can't be inherited to class B.
- This can be removed by changing access specifier to public or creating a function in class A and nesting it in class B.

PAGE NUMBER
80 AND 81
AFTER THIS.

Date
Page 82

Q. STUDENT — public → MARKS

STUDENT: private: roll no, name
public: void getdata1(), void display1()

MARKS: private: 5 marks of 5 subjects, total
public: void getdata2(), void display2()

```
#include<iostream.h>
class Student
{
    private:
        int roll;
        char name[30];
    public:
        void getdata1()
        {
            cout<<"Enter your roll number and name:";
            cin>>roll>>name;
        }
        void display1()
        {
            cout<<"\nName = "<<name;
            cout<<"\nRoll Number = "<<roll;
        }
};
class Marks : public A
{
    int m[5], i, total;
    public:
        void getdata2()
        {
            total=0; cout<<"\nEnter marks:";
            for(i=0;i<5;i++)
            {
                cin>>m[i];
                total=total+m[i];
            }
        }
        void display2()
        {
            cout<<"\nTotal = "<<total;
        }
};
int main()
{
    Marks m1;
    m1.getdata1();
    m1.getdata2();
    m1.display1();
    m1.display2();
    return 0;
}
```

Date
Page 83

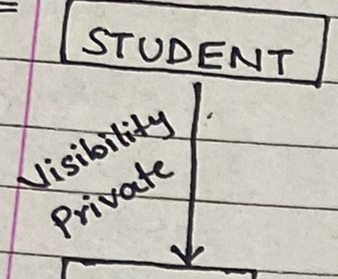
⊙ Output:

```
Enter your roll number and name: 4136
Yashasvi
Enter marks:1
2
3
4
5
Name = Yashasvi
Roll Number = 4136
Total = 15
```

07/07/'22.

Date
Page 80

Q.



STUDENT: private - roll no, name
public - void getdata1()
void display1()

MARKS: private - 5 marks of 5 subjects, total
public - void getdata2()
void display2()

O/P : Name =
Roll Number =
Total =

→
```
#include<iostream.h>
class Student
{
private:
    int roll;
    char name[30];
public:
    void getdata1()
    {
    cout<<"Enter your roll number and name:";
    cin>>roll<<name;
    }
    void display1()
    {
    cout<<"\nName = "<<name;
    cout<<"\nRoll Number = "<<roll;
    }
};
```

Date
Page 81

```
class Marks : private Student
{
private:
    int m[5],total=0,i;
public:
    void getdata2()
    {
        getdata1();
        cout<<"\nEnter your marks:";
        for(i=0;i<5;i++)
        {
            cin>>m[i];
            total=total+m[i];
        }
    }
    void display2()
    {
        display1();
        cout<<"\nTotal = "<<total;
    }
};
int main()
{
    Marks m1;
    m1.getdata2();
    m1.display2();
    return 0;
}
```

08/07/22.

Date Page 84

* Method Overloading

⇒

```
class A
{
    int a=1;
    public:
        void display()
        {
            cout<<" a = "<<a;
        }
};
class B : private A
{
    int b=2;
    public:
        void display()
        {
            display();
            cout<<" b = "<<b;
        }
};
int main()
{
    B b1;
    b1.display();
    return 0;
}
```

- Here, the code will run, but the display() method will call itself recursively and go in to an infinite loop.

Date Page 85

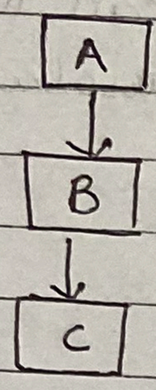
- If we want the function of class B to call the function of class A, we make use of the scope resolution operator.
- In class B,

```
void display()
{
    A::display();
    cout<<" b = "<<b;
}
```

- Here, B is the child class and A is the parent class.
- This will work even when the visibility mode of this inheritance is public.

11/07/'22.

* Multi-Level Inheritance:



- Process in which we can derive a class from an already derived class.
- If we want to inherit properties of A into C through B, then visibility mode when B is child class should be public or protected.

⇒
```
#include<iostream.h>
class A
{
  public:
     void eat()
     {
        cout<<"Eating";
     }
};
class B : public A
{
   public:
      void sleep()
      {
         cout<<"Sleeping";
      }
};
class C : public B
{
    void read()
```

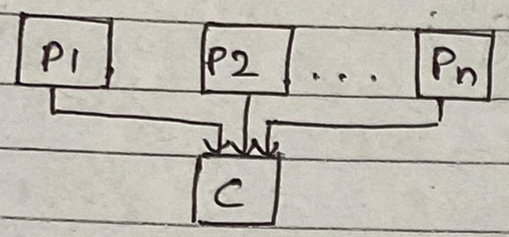


```
    {
        cout<<"Reading";
    }
};
int main()
{
    C c1;
    c1.eat();
    c1.sleep();
    c1.read();
    return 0;
}
```

⊙ Output:

Eating Sleeping Reading

* Multiple Inheritance :



⇒
```
#include <iostream.h>
class A
{
    protected:
        int a;
    public:
        void get_a(int n)
        {
            a=n;
        }
};
class B
{
    protected:
        int b;
    public:
        void get_b(int n)
        {
            b=n;
        }
};
class C : public B, public A
{
    public:
```

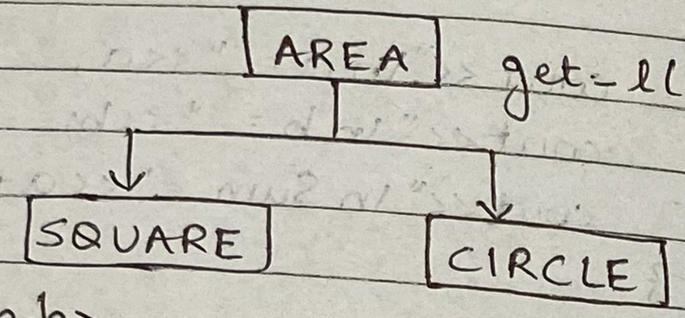


```
    void display()
    {
        cout<<"\n a = "<<a;
        cout<<"\n b = "<<b;
        cout<<"\n Sum = "<<a+b;
    }
};
int main()
{
    C c1;
    c1.get_a(10);
    c1.get_b(20);
    c1.display();
    return 0;
}
```

⊙ When integers 'a' and 'b' are declared private, we can call get_a() and get_b(), but we cannot get their values from the main() function.

- One way to get values from the main function when they ~~the~~ are private is to make the return type of functions to int, and store their values in variables declared in derived class.

* Hierarchial Inheritance:

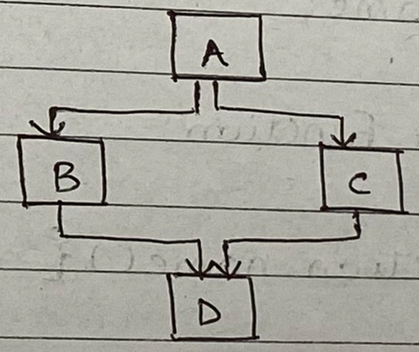


⇒
```
#include<iostream.h>
class Area
{
  protected:
        int l;
  public:
        void get-l(int n)
        {
            l=n;
        }
};
class Square : public Area
{
    public:
        void Area()
        {
            cout<<"Area of square = "<< l*l;
        }
};
class Circle : public Area
{
    public:
        void ar()
        {
```



```
            cout<<"Area of circle = "<< 3.14 * l* l;
        }
};
int main()
{
    Square s1;
    Circle c1;
    s1.get-l(10);
    s1.Area();
    c1.get-l(7);
    c1.ar();
    return 0;
}
```

⇒



During this type of inheritance the variables of A get passed to class D twice by class B and class C.
This duplicates the values of variables of class A. To prevent this we use the concept of Virtual base class.
The parent class is the virtual base class.

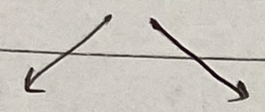
12/07/22.

Date Page 92

* STATIC DATA MEMBER, STATIC MEMBER FUNCTION

class A
↳ int a
↳ int b

As an integer is of 2 bytes, both objects a1 and a2 store 4 bytes



a1	a2
a1.a	a2.a
a1.b	a2.b

⊙ Syntax of Static Data Member:

```
static datatype var_name;
```

⊙ Syntax of Static Member Function:

```
static return_type function_name(){
}
```

⊙ When we declare a variable as static, it forms one common memory location for whole class.

- So we can store only one value for all objects instead of different values for all objects.
- When we change the value in one object, it is changed for all objects.
- It is generally used for count / sum / total.
- We need to initialise the value of count outside the class.

Date Page 93

- Static data member can be accessed by any member function but a static member function can only use static data members.

⇒
```
#include<iostream.h>
//using namespace std;
class A
{
    static int count;
    public:
    static void plus()
    {
        count++;
    }
    void display()
    {
        cout<<"\n"<<count;
    }
};
int A :: count = 0;
int main()
{
    A a1,a2;
    a1.plus();
    a1.display();
    a2.display();
    return 0;
}
```

⊙ Output : 1
1

Date Page 94

* INLINE FUNCTIONS :

⊙ eg) sum(int a, int b)

- When we call a function like the one above, the process is :
  (i) Send actual parameter to formal parameter
  (ii) execute the whole function
  (iii) return the value
  (iv) print the result.
- When this function is called multiple times, it will take a lot of time to execute the whole process.
- We make such functions inline. This replaces the function calling with the function definition which saves time.
- The function runs at the time of compilation and so the whole function will be executed in the "cout" statement itself.

- Syntax :

```
inline datatype func_name(arg(s))
{
}
```

eg)
```
#include<iostream.h>
inline int pro(int a, int b)
{
    return a*b;
}
int main()
```

Date Page 95

```
{
    int a,b;
    cout<<"Enter values of a and b:";
    cin>>a>>b;
    cout<<"Product is :"<<pro(a,b);
    return 0;
}
```

- Here, the function pro(int a, int b) will be executed in the cout statement where it is called.

- Cases in which we cannot use inline :
  (i) Function contains a loop
  (ii) Function contains static variables
  (iii) Function is recursive
  (iv) Function doesn't contain return statement when its return type is other than void.
  (v) Function contains switch or goto statement.