OOPS.

class.

* class Box {
  Public:
  double length;
  double breadth;
  double height;
};

- By default access to members of C++ class is private.
- They can only accessed only through method of class

- Class is public by default for structs and private by default for classes.

* Define objects :-

```
Box Box1; // Declare Box1 of type Box
Box Box2; // "    "    Box2 " "  "
```

* Accessing the data members:-

```
using namespace std;
class Box{
    Public:
        double length;
        double breadth;
        double height;
};
int main(){
   Box Box1;
   Box Box2;
```

```
double volume = 0.0;

Box1.height = 5.0;
Box1.lenght = 6.0;
Box1.breadth = 7.0;

Box2.height = 10.0;
Box3.length = 12.0;
Box4.breadth = 13.0;

volume = Box1.height * Box2.length * Box1.breadth;
cout << volume;   // o/p - 210
```

Static members of cpp class :-

- when we declare a member of a class as static it means no matter how many objects of the class are created there is only one copy of static member.
- It is shared by all objects of the class.
- All static data is initialized to zero when the first object is created, if no other initialization is present.
- We can't put in the class definition but it can be initialized outside the class as done in the following ex. by redeclaring the static variable.
  using scope resolution operator :: to identify which class it belongs to.

```
Class Box {
    public:
      static int objectCount;

      //constructor def.
      Box (double l=2.0, double b=2.0, double h=2.0)
    {  cout<< "constructor called " << endl;
       length = l
       breadth = b;
       height = h;
      //increase every time object is created
       objectcount++;
    }
    double volume() {
          return length * breadth * height;
    }
  Private:
     double length;
     double breadth;
     double height;
};
//initilize static member of B class Box
int Box:: objectCount =0;
int ~~void~~ main (void) {
        Box Box1 (3.3, 1.2, 1.5);
        Box Box2 (8.5, 6.0, 2.0);

  cout << "Total objects: " << Box::objectCount <<endl;
  return 0;
}
Constructor called
 Constructor called
Total object = 2.
```

Static function member :-

- By declaring a function member as static, you make it independent of any particular object of the class.

- Static member function can be called even if no objects of the class exist. & static functions are accessed using only the class name & scope resolution operator ::.

- A static member function can only access static data member, other static member functions and any other

- static member functions have a class scope and they do not have to access to this pointer of the class.

Class Access Modifiers:-

Public:-

- You can set & get value of public variables without any member function

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

Class Line {
    public:
      double length;
   void ~~double~~ setLength(double len);
      double getlength(void);
};

// Member functions defination s
double Line :: getLength(void) {
      return length;
}

void Line :: setlength(double len) {
      length = len;

int main(){
    Line line;
    // set line length;
    line.setlength(6.0);
    cout << line.getLength() << endl;

    //set line length without member func.
    line.length = 10.0; //because length is public;
    cout << line.length << end;
```

o/p - 6

10

Private members:-

- Only class & friend functions can access private members.
- By default all the members of class would be private.

```
class Box {
    double width //Private.
    Public:
        double length;
        void setwidth(double wid);
        double getwidth (void);
};

double Box::getwidth (void){
        return width;
}
void Box::setwidth (double wid) {
        width = wid;

in main() {
    Box Pbox;
  //set Box length because it is public
    box.length = 10.0;
     cout << box.length << endl;

    box.width = 10.0; //Error: Private.
    box.setwidth (10.0);
     Cout << box.getwidth ()<<endl;

return 0;

}.  O/P - 10
           10.
```

Protected members :-

- They can be accessed in child classes which are called derived classes.

```
class Box {
    protected:
        double width;
};
class SmallBox : Box {
    public:
        void getsmallwidth (double wid);
        double getsmall width (void);
};

double smallBox :: getsmallwidth (void) {
  return width ;
}

void smallBox :: setsmall width (double wid) {
        width = wid;
}

int main () {
    SmallBox box;

    box. setSmallwidth (5.0);
    cout << box. get small width ();
    return 0;
}
```

o/p = 5.

Class Constructor & Destructor:-

- A class constructor is a special member function of a class that is executed when ever we create new objects of that class.

- Constructor will have exact same name as the class and it doesn't have any return type.

```
class Line {
    public:
        void setLength(double len);
        double getLength(void);
        Line();
        Private:
            double length;
    };
Line::Line(void) {
    Cout<<"Object is being created"<<endl;
}
void Line::setLength(void) {
    return length;
}
int main() {
    Line Line;
    line.setLength(6.0);
    cout <<" ling.getLength()<<endl;
    return 0;
}
```

O/p- Object is being created
~~length of line~~ 6.

Parameterized Constructor:-

```
class Line {
   public:
      void setLength (double len);
      double getLength (void);
      Line(double len);
   private:
      double length;
};

Line:: Line (double len) {
    cout << len << endl;
    length = len;          // 10
}
void Line :: setLength (double len) {
      length = len;
}

void double Line :: getLength (void) {
          return length;
 }
int main () {
    Line line (10.0);

    cout << line.getLength() << endl; // 10

    line.setLength (6.0);
    cout << line.getLength () << endl; // 6

  return 0;
}
```

o/p -

$O(n^2)$ medium.

class destructor:-

- It is a special member function of a class that is executed whenever an object of its class goes out of scope or whenever the delete expression is applied to a pointer to the object of that class.

- Destructor can be useful for releasing resources before coming out of the program like closing files, releasing memories etc.

```
class Line {
   public:
      void setLength(double len);
      double getLength(void);
      Line();
      ~Line();

Private:
    double length;
};
Line::Line(void) {
    cout << "object is created" << endl;
}
Line::~Line(void) {
    cout << " object is deleted" << endl;
}
void Line::setLength(double len){
      length = len;
}
double Line::getLength(void) {
      return length;
}
```

```
int main(){
    Line line;
    line.setlength(6.0);
    cout << line.getlength() << endl;
  return 0;
}
```

O/P- Object is created
6
Object is deleted.

COPY Constructor:-

- It is a constructor which creates an object by initilizing it with an object of same class, which has been created previously.
  - * Initialize one object from another of same type.
  - * Copy an object to pass it as an argument to a function.
  - * Copy an object to return it from a function.

- If it is not define in a class, the Compiler itself defines one.

- If the class has pointer var and has some dynamic memory allocation. then, it must to have a copy Constructor.

```
classname (const classname &obj) {
    // body of constructor
}
```

```
class Line {
  public:
    int getlength(void);
    Line (int len);
    Line (const Line &obj); // copy constructor.
    ~Line ();
  private:
    int *ptr;
};
Line :: Line (int len) {
    cout << "Normal cons allocating Ptr" << endl;
    ptr = new int;
    *ptr = len;
}
Line :: Line (const Line &obj) {
  cout << "Copy cons allocating Ptr" << endl;
    ptr = new int;
    *ptr = *obj.ptr // copy the value.
}
Line :: ~Line (void) {
    cout << "freeing memory" << endl;
    delete ptr;
}
int Line :: getlength (void) {
    return *ptr;

void display (Line obj) {
  cout << "length of line: " << obj.getlength()
                              << endl;
}
```

```
int main () {
    Line line (10);
    display (line);
    return 0;
}.
```

o/p - Normal cons allocating ptr.
Copy Constr allocating ptr.
10
Freeing memory
Freeing memory.

C++ this pointer :-

* Every object has access to its its own address through an important pointer called this pointer.

- This pointer is an implicit pointer to all member functions.
Therefore, inside a member function, this may be used to refer to the invoking object.

- Friends are not members of a class therefore Friend func don't have this pointer

- Only member functions have a this pointer.

```
class Box {
  public:
  Box(double l=2.0, double b=2.0, double h=2.0) {
    cout << "constructor called" << endl;
    length = l;
    breadth = b;
    height = h;
  }
  double volume() {
    return length * breadth * height;
  }
  int compare(Box box) {
    return this->volume() > box.volume();
  }
  private:
    double length;
    double breadth;
    double height;
};
int main(void) {
  Box Box1(3.3, 1.2, 1.5);
  Box Box2(8.5, 6.0, 2.0);
  if (Box1.compare(Box2)) {
    cout << "Box2 is small" << endl; }
  else
    { cout << "Box1 is ~~small~~ equal or small";
    }
  return 0;
}
```

Constructor called
Constructor called
Box2 is equal to or larger than Box1

Friend Functions:-

- If func. is defined as friend function then private & protected data of a class can be accessed using the function.
- for accessing data, the declaration of a friend should be done inside the body of class starting with keyword friend.

```
class class_name {
    friend. data_type function_nam(args);
};
```

- function defination doesn't use either the keyword friend or scope resolution operator.

Characteristics of friend function:-

* function is not in the scope of the class to which it has been declared as a friend.
* It cannot be called using the object as it is not in the scope of that class.
* It can be invoked like a normal function without using object.
* It cannot access the member names directly and has to use an object name and dot membership operator with member nam
* It can be declared either in private or in public part.

classmate
Date
Page

* Ex- #include <iostream>
using namespace std;
class Box {
private:
int length&;
public:
Box(): length(0) {}
friend int printLength(Box);
};
int printLength(Box b) {
{ b.length += 10;
return b.length;
}
int main()
{ Box b;
cout << "Length of box:" <<
printLength(b) << endl;
return 0;
}

o/p = 10.

friend function with two classes:—

```
class B; //forward declaration.
class A {
    int x;
    public:
    void setdata(int i){
        x=i;
    }
    friend void min(A,B); //friend function.
};
class B {
    int y;
    public:
    void setdata(int i)
    { y=i;
    }
    friend void min(A,B); //friend function.
};
void min(A a, B b);
if(a.x<b.y)
  cout<<a.x;
  else
    cout<<b.y;
}
int main(){
    A a;
    B b;
  a.setdata(10);
  b.setdata(20);
    min(a,b);
    return 0;
}
```

o/p — 10

Friend class:—

```
class A {
    int x = 5;
    friend class B;
};
class B {
  public:
    void display (A &a) {
        cout << a.x;
    }
};
int main () {
    A a;
    B b;
    b.display (a);
        return 0;
};
```

o/p = 5

Here B is declared as friend inside A
Therefore, B is friend of class A.
class B can access the private members of class A.

Inheritance :—

* Private by default.

```
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 rect;

    rect.setWidth(5);
    rect.setHeight(7);
    cout << rect.getArea() << ;
```

output = 35.

Access Control & inheritance:-

Access	public	protected	Private
Same class	yes	yes	yes
Derived	yes	yes	no
outside	yes	no	no

- Derieved class inherits all base class methods with following -

* Constructor, destructors and copy constructor of base class
* Overload operators of the base class.
* The friend ~~etc~~ functions of base class.

- **Public inheritance:-** when deriving a class from a public base class, public members of base class becomes public members of derieved class,
protected members of base class becomes protected of derived class
A base class's private members are never accessible directly from derived class but can be accessed through calls to the public and protected members of base class.

- **Protected inheritance:-**
When deriving from a protected base class, public and protected members of the base class become protected members of derieved class.

Private inheritance:-

when deriving from a private base class, public and protected members of the base class become private members of derived class.

Multiple Inheritance:-

```
class shape
    public:
        void setwidth (int w) {
            width = w;
        }
        void setheight (int h) {
            height = h;
        }
    protected:
        int width;
        int height;
};
class paintcost {
    public:
        int getcost (int area) {
            return area * 70;
        }
};

class Rectangle: public shape, Public paintcost{
    public:
        int getArea () {
            return (width * height);
        }
};
```

```
int main(void) {
    Rectangle rect;
    int area;

    Rect.setWidth(5);
    Rect.setHeight(7);

  area = Rect.getArea();

  cout << Rect.getArea();

  cout << Rect.getCost(area);
```

o/p - 35
$2450

Polymorphism:-

- means many forms.
- It occurs when there is a heirarchy of classes and they are related by inheritance.

- It means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function