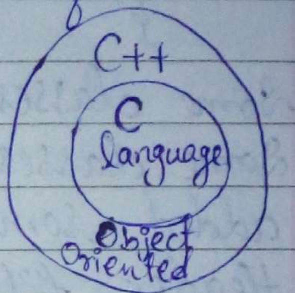
C++

C++ is a middle-level programming language developed by Bjarne Stroustrup starting in ~~1979~~ 1981 at Bell Labs.

Bjarne Stroustrup added the C language with the help of object oriented programming language named it C with Class.
Class & Objects are parts of C++.

C++ is an extended form of C language



C	C++
Global function	Local function
variables are both local & global	local variables included
	Classes are included in these programs
	Definitions are also included

```
#include <iostream.h>
#include <conio.h>
Class short
{
void
=
```

Some functions are related to classes
Example:

```
      #
      #
      {
        class student

              total c
               ≡
      }
```

* Some classes are predefined.
* Some classes are user-defined.
* added some functions are library classes
* Header files included library classes
  e.g. #<iostream.h>
* All steps are separated same as C-program

* C++ programs are saved as .CPP extension.
  e.g.→

C program-

First.cpp

```
#include<stdio.h>
void main()
{
 printf("Chal gaya");
}
```

Header Fiels

$$\begin{tikzcd} & istream \\ iostream \arrow[ur] \arrow[r] & ostream \end{tikzcd}$$

$$\begin{tikzcd} \text{Lib class} \arrow[d] & \text{header file} \arrow[d] \\ \text{ostream} \arrow[d, "\text{object}"] & \text{iostream} \\ \text{cout} & \end{tikzcd}$$

Cout:- object of lib class ostream.

Syntax:-

```
cout<<"        ";
```

<< → Insertion op./put to op.

first.cpp

```
#include<conio.h>
#include<iostream.h>
void main
{
 clrscr();
 cout<<"Chal gaya";
 getch();
}
```

```
int n=60;
cout<<n;
Output: 60
```

```
if int n=60;
cout<<"n";
Output: n
```

```
cout<<"Chal gaya"<<"   "<<"Bhai";
```

```
cout<<"Total is:"<<n;
Output: Total is: 60
```

cin :- object of lib class istream

Syntax: $\nearrow$ ex~~tra~~ction op. / get from op.

```
cin >> var ;
```

e.g. →
```
cin>> n;
cin>> var1>> var2>> ---- ;
cin>>a>>b>>c ;
```

WAP to add 2 numbers in C++.

```
#include<iostream.h>
#include<conio.h>
void main()
{
 int a,b,c;
 clrscr();
 cout<<"Enter first number:";
 cin>>a;
 cout<<"Enter second number:";
 cin>>b;
 c=a+b;
 cout <<"Total is:"<< c;
 getch( );
}
```

Tuesday
31-01-17

Scope Resolution operator :-

```
#
#
int n=5;
void main()
{
 int n=10;
 cout<<n; //10
 cout<<::n; //5
 cout<<n+::n; //15
}
```

Scope Resolution operator is used to access a global variable in a function where a local variable is declared as same name.

Wednesday
01-02-17

Reference variable-

```
int n=17
int &K=n;
 K=10;
cout<<n; //10
   n=21
cout<<n; //21
cout<<K; //21
```

```
 2021
 |17|
  n
  K
```

```
#
~~void main()~~
void xyz(int &b)
{
 cout<<b;
   b=18;
}
void main()
{
 int a=5
  xyz(a);
 cout<<a;
}
```

Manipulators-
Manipulators are used to get formatted output. All manipulators are defined in <iomanip.h
endl- It is used to get output in next line.

```
Cout<<"Hello Rohit"; // Hello Rohit
cout<<"Hello \n Rohit"; // Hello
                           Rohit

or cout<<"Hello"<<\n<<"Rohit"; —↑ ↑
or cout<<"Hello"<<endl<<"Rohit";   | same O/P
or cout<<"Hello"<<endl; } same O/P ↑
   cout<<"Rohit";
```

Setw( ) - This function is used to set width of O/P.

e.g.-
```
#include<iostream.h>
#include<conio.h>
void main ()
{
int a=2146, b=18, c=27254;
cout<<"Name"<<"marks"<<"\n";
cout<<"Amit"<<a<<"\n";
cout<<"Raj"<<b<<"\n";
cout<<"Abhishek"<<c<<"\n";
getch();
}
```

Output-
```
Name Marks
Amit 2146
Raj 18
Abhishek 27254
```

DATE: / /
PAGE NO:

```
#include<iostream.h>
#include<conio.h>
void main()
{
int a=2146, b=18, c=27254;
cout<< setw(20)<<"Name"<<setw(10) <<"Marks"<<endl;
cout<<setw(20)<<"Amit"<< setw(10)<<a<<endl;
cout<<setw(20)<<"Raj"<<setw(10)<<b<<endl;
cout<<setw(20)<<"Abhishek"<<setw(10)<<c<<endl;
getch();
}
```

Output:

```
Name          Marks
Amit          2146
Raj           18
Abhishek      27254
```

02-02-17
Thursday

C++ Tokens-
(1.) Keywords (32+16)
(2.) Identifiers
(3.) Literals/ Constants
(4.) Special Symbols
(5.) Operators

Public	try
Private	catch
Protected	throw
Class	new
Framed	delete
Virtual	inline

```
returntype function name (arg. list)
{
 body of function
}
```

```
char xyz(int a, float b)
{
 char k;
 ==
 return k;
}
```

```
int a=10, b;
b=a;
b=16
```

Output:
a=10
b=16

Teacher's Signature

Call by value-

```
# include <iostream.h>
#include <conio.h>
void xyz (int);
void main ()
{
 int a=10;
 cout<<a;
 xyz (a);
 cout<< a;
 getch();
}
void xyz (int m)
{
 cout<< m;
 m=20;
}
```

(eg.) int a= 10, b;
b=a;
b=16;
Ouput:
a=10
b=16

Call by address-

```
#
#
void xyz (int *);
void main ()
{
  int a= 10;
  cout<<a;
  xyz(&a)
  cout<<a;
  getch();
}
void xyz (int *m)
{
  cout <<*m;
  *m= 20;
}
```

2021
|20| |2021|
a m

Output
10 10 20

(eg.) int a=10, *b;
b=&a
*b=16;
value at address ~~*b=16~~

Function reduces size of program.

Call by Reference

```
#
#
void xyz (int &);
void main()
{
  int a=10;
  cout<<a;
  xyz(a);
  cout<<a;
  getch();
}
void xyz (int &m)
 {
  cout<<m;
  m=20;
 }
```

|20|
a
m

Output
10 10 20

(eg.)
```
int a=10;
int &b=a;
b=16;
```

Inline Function—

```
void main()
{
 ===
 xyz (10, 20);
 ===
}
inline void xyz(int a, int b)
{
 ===
}
```

Inline Function—
- In this function the code is changed.
- Memory is more required.
- Size increased
- Jump the value of argument.

Inline Function-
In C++, we can define function as inline when the function body is small and need to be called many times, thus reduces the overhead in calling a function like passing values, passing control, returning values, returning control.

Function Overloading-
In C++, we can define more than one function with same name but signature (number of argument or type of argument) must be different.

```
#include<iostream.h>
#include<conio.h>
void sum(int a, int b)
{
cout<< a+b;
}
void sum( int a, int b, int c)
{
cout << a+b+c;    ...
}
void main( )
{
clrscr( );
Sum (10,20,30)
Sum (10,15)
getch( );
}
```

Tuesday
DATE 07.02.17
PAGE NO :

Default Argument function—

```
#
#
void xyz(int=10, int=5);
void main ()
{
 xyz();
 xyz(2,3);
 xyz(8);
}
void xyz(int a, int b)
{
 cout<< a+b<< endl;
}
```

Output
```
25
5
28
```

```
#
#
void printline(int=5, char='*');
void main()
{
 printline (7,'-');
 printline (8);
 printline ();
 getch();
}
void printline(int n, char c)
 {
 int i;
 for(i=1; i<=n; i++)
 {
 cout << c;
 }
 cout << endl;
}
```

Output
```
-------
********
*****
```

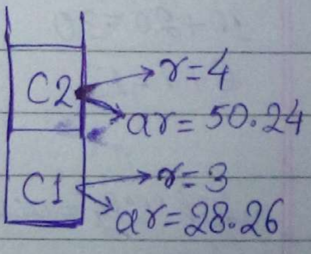
Teacher's Signature

Object Oriented Program—
Object Oriented Programming of C++ gives the concept of working on objects. An object is treated as a single entity which is a combination of data and functions that operates on data.

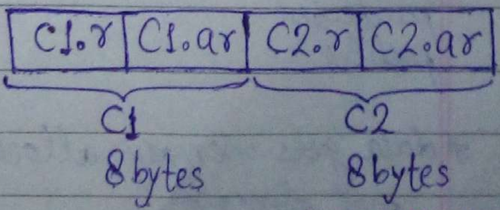
Objects and Class—
Object: Any entity that can have some data and functions that can operate using its data can be defined as an object.
Class: Collection of similar objects is known as a class.

Syntax—
~~eg.~~→
```
Class Classname
{
type member1; or type data1;  }— data members.
type member2; or type data2;  }
  ——
  returntype main function1()
  {
   ≡
  }
  returntype main function2()
  {
   ≡
  }
};
```

e.g.-
```
# include<iostream.h>
# include<conio.h>
class circle
{
public: float r;
        float ar;
        void area()
        {
        ar=3.14*r*r;
        cout<< ar;
        }
        void radle()
        {
        cout<<"Enter radius";
        cin>>r;
        }
        void main()
        {
         circle c1,c2;
                c1.r=3
                c1.area()
                c2.radle()
                c2.area()
        getch();
        }
```
(object name. member name)



Output:
```
28.26
Enter radius
50.24
```



WAP to add two numbers.

```
#include<iostream.h>
#include<conio.h>
void main()
{
int a,b,c;
cout<<"Enter two no.";
cin>>a>>b;
c=a+b;
cout<<c;
getch();
}
```

Output

```
 Enter two no. 6↵
 7↵
 6+7=13
 Enter two no. 9↵
 12↵
 Enter two no. 10↵
 20↵
 9+12=21
 10+20=30
 6+7=13.
```

visibility = accessibility

private
protected
public

* data gets memory allocation not function.

```
#include<iostream.h>
#include<conio.h>
class sum
{
public: int a,b,c;
void valuele()
{
cout<<"Enter two no.";
cin>>a>>b;
}
void showresult()
{
cout<<a<<"+"<<b<<"="<<c<<endl;
}
void add()
{
 c=a+b;
}
};
void main()
{
 sum S1,S2,S3;
 S2.valuele();
 S2.add();
 S2.showresult();
 S1.valuele();
 S3.valuel();
 S3.add();
 S1.add();
 S1.showresult();
 S3.showresult();
 S2.showresult();
getch();
}
```

→ In Class, default visibility and accessibility is private.

→ The private members can be used only by the respective class in which they are created.

→ Their visibility and accessibility is restricted in the class it is declared.

#
```
class fan
{
  int n; →(by default private)
public: int p; →(public member)
        int s;   (public data member until other visibility
protected: int i;     is mentioned).
};
```

Private members cannot be used outside the class if it is not declared in respective class.

#
```
class fan
{
 int a;
public: int b;
        void xyz();
        {
        cout<<a<<b;
        }
}
```

```
void area ()
  {
    fan d1;
    d1.a=10;  ×  (private cannot be used in other
    d1.b=20;  ✓    function declared outside class)
  }
```

→ To use private function:-

```
class fan
{
private: xyz ();
        {
         int a, b;
         cout << a<<b;
public: void pk ();
        {
         xyz();
        }
 }
  void area ()
     {
      fan d1;
      d1.pk ();
     }
```

→ Program calculating area:-

```
#
class circle
{
private  ~~public~~: float r, ar; ←(It should be declared
public: void area()    as private for better result)
       {
        ar = 3.14 * r * r;
       }
void show ()
  {
   cout<<r<<"   "<<ar<<endl;
  }
void  getrad()
  {
   cout<<"Enter radius";
   cin>>r;
  }
void main()
{
 circle c1, c2;
 c1.getrad();
 c2.getrad();
 c2.area();
 c2.show();
 c1.area();
 c1.show();
}
```

Output-

```
Enter radius 4
Enter radius 5
 5    78.5
 4    49.9
```

Tuesday
DATE 14/02/17
PAGE NO.:

Access Modifier Class

	In Same Class	Friend Function	In Sub Class	Global Function or Non-subclass
Private	Yes	Yes	No	No
Protected	Yes	Yes	Yes	No
Public	Yes	Yes	Yes	Yes

When we define number function outside class.
Syntax:-
returntype classname:: functionname()
#
```
class A
{
    int a, b;
    float c;
    void xyz();
    void PC ();
};
  void A:: xyz()
  {
    =
  }
  void A:: PC ( )
  {
    =
  }
```
(i) Member function definition inside the class
(ii) Member function definition outside the class.
Teacher's Signature
Scanned by CamScanner

Wednesday
DATE 15/02/17
PAGE NO. :

Array of Objects -

```
#
#
class student
{
 private: int rn, mo;
          void getdata()
          {
          cout<<"Enter roll no.:";
          cin>>rn;
          cout<<"Enter marks obtained:";
          cin>>mo;
          }
          void showresult()
           {
            float p;
            p= mo/700.0*100;
           cout<<rn<<"  "<<p<<endl;
           }
};
   void main()
   {
   int i,n;
   student st[1000];
   cout<<"How many students?";
   cin>>n;
   for(i=0; i<n; i++)
   {
     cout<<"Enter data of"<<i+1<<"student:"<<endl;
     st[i].getdata();
   }
```

Teacher's Signature

```
cout<<"result\n";
for(i=0; i<n; i++)
  {
   st[i].showresult();
  }
 getch();
}
```

Output-

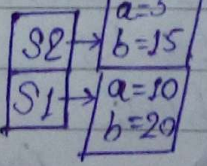
```
How many students?: 83↵
Enter data of 1 student:
Enter roll no.: 601↵
Enter marks obtained: 559↵
Enter data of 2 student:
Enter roll no.: 602↵
Enter marks obtained: 550
Result
601    79.85....
602    78.57....
```

Teacher's Signature

Scanned by CamScanner

Wednesday
DATE 22/2/17

Member Function with Argument-
#
```
class sum
{
int a, b;
public: void getnum(int p, int q)
        {
         a=p;
         b=q;
        }
        void showresult()
         {
         cout<<a+b;
         }
       };
       void main()
        {
        sum S1,S2;
        S1.getnum(10,20);
        S1.showresult();
        S2.getnum(5,15);
        S2.showresult();
        getch();
       }
```



Output-
```
30
20
```

Member function returning value-

```
#
class sum
{
 int a, b;
public: void getnum(int p, int q)
      {
        a=p;
        b=q;
      }
      int showresult ()
      {
       return a+b;
      }
};
void main ()
 {
  int k
  sum S1, S2;
  S1.getnum(10, 20);
  K= S1.showresult ();
  cout << k;
  S2.getnum(5,15);
  k=S2.showresult();
  cout<<k;
  getch();
}
```

Output-

```
30
20
```

Teacher's Signature

Nesting of member function-

#

```
class sum
{
 int a, b, c;
public: void getnum(intp, intq)
        {
         a=p;
         b=q;
        }
        int tot()
        {
         return a+b;
        }
        void show()
        {
         c= tot();
         cout<<c;
        }
};
      void main()
      {
       sum S1;
       S1.getnum(10,20);
       S1.show();
       getch();
      }
```

$$\begin{tikzcd} \boxed{S1} \arrow[r] & \boxed{\begin{array}{l} a=10 \\ b=20 \\ c=30 \end{array}} \end{tikzcd}$$

DATE / /
PAGE NO.

Member Function Overloading -
#
```
class sum()
{
 int a, b;
public: void getnum(int p, intq)
{
 a=p;
 b=q;
}
void getnum()
  {
   cout <<"Enter two values:";
   cin>>a>>b;
  }
void getnum(int p)
  {
  a=p;
  b=p;
  }
  void show()
  {
  cout<<a+b;
  }
};
 void main()
 {
 sum S1, S2, S3
```

Teacher's Signature

```
S1.getnum();
S2.getnum(5,7);
S3.getnum(10,20);
S1.show();
S2.show();
S3.show();
getch();
}
```

Thursday
23-02-2017

Passing Object as Argument-

#

```
Class circle
{
 float r,ar;
public: void getr()
      {
       cout<<"Enter radius:";
       cin>>r;
      }
      void show area()
       {
        ar=3.14*r*r;
        cout<<ar;
       }
};
 void xyz(circle ob)
  {
   ob.showarea();
  }
```

```
void main()
{
 circle obj;
 obj.getr();
 xyz(obj);
 getch();
}
```

Function returning object-

```
#
#
class circle
{
  float r,ar;
public: void getr()
      {
       cout<<"Enter radius:";
       cin>>r;
      }
      float showarea()
       {
        ar=3.14*r*r;
        cout<<ar;
        return ar;
       }
};
 circle xyz()
  {
   circle ob;
 ob.getr();
 return ob;
}
void main()
{
circle obj;
obj=xyz();
obj.showarea();
getch();
}
```

Friend Function-

In principle, private and protected members of a class cannot be accessed from outside the same class in which they are declared. However, this rule does not effect friend.

If we want to declare an external function as friend of a class, thus allowing this function to have access to the private and protected members of this class, we do it by declaring a prototype of this external function within the class, and preceding it with the keyword friend.

Syntax:

```
        friend function name();
```

eg.→
```
#include<iostream.h>
#include<conio.h>
class circle
{
    float r, ar;
    public: void getr()
    {
    cout<<"Enter radius:";
    cin>>r;
    }
    void showarea()
    {
     ar= 3.14*r*r;
     cout <<ar;
    }
    friend void xyz();
};
    void xyz()
    {
     circle c1;
     c1.getr();
     c1.showarea();
    }
   void main()
   {
    clrscr();
    xyz();
    getch();
   }
```

Output

```
Enter radius: 5↵
78.5
```

A single friend function for more than one class-

There may be a Situation where two different classes use a member function that performs similar type of activity. Now in this condition instead of declaring and defining the function in two different class can be avoid by declaring a single friend function in two classes and later on defining it only once.

e.g.→

```
#include <iostream.h>
#include <conio.h>
class B;
class A
{
      int n1;
      public: void getn1()
      {
      cout<<"Enter first number:";
      cin>>n1;
      }
      friend void sum(A x, B y)
};
class B
{
      int n2;
      ~~public: void getnum2()~~
      public: void getn2()
```

```
    {
     cout<<"Enter second number:";
     cin>>n2;
    }
    friend void sum(A x,B y)
};
    void sum(A x, B y)
    {
    cout<<x.n1+y.n2;
    }
    void main()
    {
    clrscr();
    A ob1;
    B ob2;
    ob1.getn1();
    ob2.getn2();
    sum(ob1,ob2);
    getch();
    }
```

Output-

```
Enter first number:2↵
Enter second number:3↵
 5
```

Friend class-

Just as we have the possibility to define a friend function we can also define a class as friend of another one, granting that first class access to the protected and private members of the second one.

e.g.→
```
#include<iosfream.h>
#include<conio.h>
class sum
{
int a,b,c;
 public:void getnum()
 {
  cout<<"Enter two values:";
  cin>>a>>b;
 }
 void showsum()
 {
 c=a+b;
 cout<<c<<endl;
 }
 friend class circle;
};
class circle
{
     float r,ar;
     public:void getr()
     {
     cout<<"Enter radius:";
     cin>>r;
     }
     void showarea()
     {
     ar=3.14*r*r;
     cout<<ar;
     }
};
```
Teacher's Signature

Scanned by CamScanner

```
void main()
{
 clrscr();
 sum s1;
 circle c1;
 s1.getnum();
 s1.showsum();
 c1.getr();
 c1.showarea();
 getch();
}
```

Output-

```
Enter two values: 5↵
3↵
8
Enter radius: 5↵
78.5
```

Monday
27-02-17

Constructor-

(1.) If a member function has similar name as class, that function will be known as constructor.

(2.) It gets called automatically with each and every object created for that class.

e.g.→ #

```
class demo
{
public: demo()
      {
       cout<<"Welcome";
      }
};
 void xyz()
 {
  demo d1,d2,d3;
  getch();
 }
```

Teacher's Signature

DATE : / /

PAGE NO :

(3.) Generally, it is used to initialize the objects.

#
```
class demo
{
 public: demo()
      {
       int a, b;
       cout<<"enter the values";
       cin >> a>> b;
      }
};
void xyz()
    {
     demo d1, d2, d3;
     getch();
    }
```

4. It never return any value not even 'void'.
5. A constructor should already be declared in public.
6. Parameterized construction or constructor with arguments.

#
```
class demo
{
 int a, b;
 public: demo(int p, int q)
  {
   a=p;
```

Teacher's Signature

Scanned by CamScanner

```
        b=q;
      }
};
void xyz()
{
  demo d1(7,9), d2(12,5);
  getch();
}
```

d1 → a=7
     b=9

d2 → a=12
     b=5

7. Overloading of a Constructor-

```
#
class demo
{
  int a,b;
  public: demo(int p, int q)
   {
     a=p;
     b=q;
   }
  demo (int k)
   {
        a=k;
        a=k;
   }
   void sum()
   {
     cout<<a+b;
   }
};
```

d1 → a=6
     b=6

d2 → a=7
     b=9

From the above example it is cleared that a constructor can be overloaded.

```
void main()
{
 demo d1(6), d(7,3);
 d1.sum();
 d2.sum();
 getch();
}
```

8. Implicit Constructor-
Compiler generate an implicit constructor, when programmer does not make any constructor it has an empty body so it does not effect the program.

9. If a program does not have any constructor then only a compiler generate implicit constructor with no statements. If it has more than one constructor passing arguments, then it is required to define a constructor explicitely for an object that has no arguments.

e.g.->
```
 #
 class demo
 {
  int a,b;
  demo (int p, int q)
```

Scanned by CamScanner

```
{
 a=p;
 b=q;
}
demo (int k)
{
 a=k;
 b=k;
}
demo ()
{
}
void sum()
{
 cout << a+b;
}
};
void xyz()
{
 demo d1(7,10), d2(6), d3;
 getch();
}
```

10. A constructor can also be defined outside a class by using scope resolution operator.

e.g.→

```
demo:: demo(int k)
{
 ==
}
```

DATE: / /
PAGE NO.:

11. #
```
class demo
{
 int a,b;
  public: demo(int a=10, int b=20)
   void sum()
   {
    cout<<a+b;
   }
};
 demo:: demo(int p, int q)
 {
   a=p;
   b=q;
 }
  void main()
   {
    demo d1(4), d(7,8);
     d1. sum();
     d2. sum();
      getch();
   }
```
Hence we can set default value in constructor.

Teacher's Signature

SHREE
DATE / /
PAGE NO. :

Default Constructor-
If we define any empty constructor then it will be known as default constructor.

Copy Constructor-
Copy constructor takes the reference of the object of the class as an argument.

e.g.→
```
#
class circle
{
    float r,ar;
    public: circle()
     {
      cout<<"Enter radius:";
      cin>>r;
     }
    circle(circle &ob)
     {
       r=ob.r
     }
    void display()
     {
      ar=3.14*r*r;
      cout<<ar;
     }
};
    void main()
    {
     circle c1
     circle c2(c1);
```

c2→ r=6
ar=

c1→ r=6
ar=

Teacher's Signature

```
c1.display();
c2.display();
getch();
}
```

Wednesday
01-03-2017

Destructor

A destructor is a special member function of a class that is executed whenever an object of its class goes out of scope. A destructor has same name as the class name prefixed with a tilde symbol(~) and it can neither return a value nor can take arguments.

Destructor overloading is not allowed. When a destructor is not defined in a class by the programmer then compiler generate implicit destructor in the class.

e.g.→

```
#
class demo
{
    public:~demo()
        {
        cout<<"Hello";
        getch();
        }
};
```

Output

```
OK Hello
```

```
void main()
{
 demo d1;
 cout<<"OK";
}
```

e.g.->

```
#
class demo
{
 public:~demo()
   {
    cout<<"Hello";
    getch();
   }
  demo()
   {
    cout<<"Welcome";
   }
};
 void xyz()
 {
  demo d1;
  cout<<"TV";
 }
 void main()
 {
  demo ob1, ob2;
  cout<<"Bulb";
  xyz();
  cout<<"Fan";
 }
```

Output

```
Welcome
Welcome
Bulb
Welcome
TV
Hello
Fan
Hello
Hello.
```