Keywords

asm	double	new	switch
auto	else	operator	template
break	enum	private	this
case	extern	protected	try
catch	float	public	typedef
char	for	register	union
class	friend	return	unsigned
const	goto	short	virtual
continue	if	signed	void
default	inline	sizeof	volatile
delete	int	static	while
do	long	struct	

Added by ANSI C++ (New)

bool	export	reinterpret_cast	typename
const_cast	false	static_cast	using
dynamic_cast	mutable	true	wchar_t
explicit	namespace	typeid	

C++ Data type -

Data type

- Built-in Data type
  - int
  - char
  - float
  - void
- Derived Data type
  - Array
  - Function
  - Pointer
  - Reference
- User Defined Datatype
  - Structure
  - Union
  - Class
  - Enumeration

Name	Description	Size	Range
char	character or small integer	1 byte	signed → -128 to 127 unsigned → 0 to 255
short int (short)	Short integer	2 byte	signed → -32768 to 32767 unsigned → 0 to 65535
int	Integer.	4 bytes	signed → -2147483648 to 2147483647 unsigned → 0 to 4294967295
long int (long)	Long integer	4 bytes	" "
bool	Boolean value. It can take one of two values: true or false	1 byte	true or false
float	Floating point number.	4 bytes	+/- 3.4e +/-38 (~7 digits)
double	Double precision floating point number	8 byte	+/- 1.7e +/-308 (~15 digits)
long double	Long double precision floating point ~~number~~ number	8 bytes	" "
wchar_t	Wide character	2 or 4 bytes	1 wide character

Level of precision
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18

Precedence of operators in descending order-

Level of precedence	Operator	Description	Grouping
1	::	scope	Left to right
2	()[].->++-- dynamic_cast static_cast reinterpret_cast const_cast typeid	postfix	Left to right
3	++--~! sizeof new delete	unary (prefix)	Right to left
	* &	indirection and returned (pointer)	
	+ -	unary sign operator.	
4	(type)	type casting	Right to left
5	.* ->*	Pointer to member	Left to right
6	* / %	multiplicative	Left to right
7	+ -	additive	Left to right
8	>> <<	shift	Left to right
9	< > <= >=	relational	Left to right
10	== !=	equality	Left to right
11	&	bitwise AND	Left to right
12	^	bitwise XOR	Left to right
13	|	bitwise OR	Left to right
14	&&	logical AND	Left to right
15	||	logical OR	Left to right
16	?:	Conditional	Right to left
17	= *= /= %= += -= >>= <<= &= ^= |=	assignment	Right to left
18	,	Comma	Left to right

Header file -

#include<iostream> (For New IDE: ex - Code Blocks)
#include<iostream.h> (For Old IDE: ex - TurboC++)
Istream class supports Input function.
Ostream class supports Output function.

All objects are included in iostream file so, we can use input or output statement in our programs.

Using namespace std; -

→ It includes all C++ standard libraries.
→ We do not use this in old IDE like TurboC++ but we have to use this in all new IDE like Code Blocks, Dev C++ etc.

WAP to print "Geeky Shows" -

```
#include<iostream>
using namespace std;
main()
{
  cout<<"Geeky Shows";
}
```

Output- Geeky Shows.

Built-in Data type -

int-

```
#include<iostream>
#include<iomanip>
using namespace std;
main()
{
 int a =5;
 cout<<a;
}
```

or int a= 5.5

output- 5

output- 5

Char-

```
#include<iostream>
#include<iomanip>
using namespace std;
main()
{
 char a='y';
 cout << a;
}
```

or char 'yop';

Output- y

Output- p

float - यह 6 अंक तक लेता है किन्तु दशमलव के L तक का भी count करता है।
(कभी-कभी 7 तक)

```
# include<iostream>
# include<iomanip>
using namespace std;
main()
{
  float a = 1.12345;      or float. a = 1.1234567;
  cout<<a;
}
```

Output - 1.12345 **Output** - 1.12346 (roundup)

setprecision() - यह float को दशमलव के बाद 7 अंक तक
इसका प्रयोग करने के लिए # include<iomanip> जरूरी है। ही print करवा सकता है। और double 15 अंक तक।

```
#include<iostream>
#include<iomanip>
using namespace std;
main()
{
  float a = 1.1234567;
  cout<< setprecision(8)<<a;      or cout << setprecision(7) <<a;
}
```

Output - 1.1234567 **Output** - 1.123457

double - यह 16 अंक तक लेता है। और दशमलव के बाद 15 अंक तक लेता है।

```
# include<iostream>
#include<iomanip>
using namespace std;
main()
{
  double a = 1.7678970675
  cout<< setprecision(11) <<a;
}
```

Output - 1.7678970675

(It is not a function)
cout(Console output) - It is similar to printf
(It is object) function (C programing). It is used to print / display value of variable. Its an object of Ostream class.

Syntax - cout << variable name;
cout << "Any Massage

Ex- cout << a;
cout << "Hello World";

where << is known as insertion or put to operator.

→
```
#include<iostream>
#include<iomanip>
using namespace std;
main()
{
  cout << "Name";
}
```
Output - Name

or
```
int a=10;
cout << a;
```
Output - 10

```
#include<iostream>
#include<iostreammanip>
using namespace std;
main()
{
  int a=10;
  cout << "Value of A :" << a;
}
```
Output - Value of A : 10.

For new line - ~~Four~~ For new line '\n' or endl used.

(i) '\n' -

→
```
#include<iostream>
#include<iomanip>
using namespace std;
main()
{
 int a = 10;
 cout <<" Value of A:"<< a << '\n' << " Thanks";
}
```

Output- Value of A: 10
Thanks

(ii) endl -

→
```
#include<iostream>
#include<iomanip>
using namespace std;
main()
{
 int a =10;
 cout <<" Value of A:" << a <<endl << " Thanks";
}
```

Output- Value of A: 10
Thanks.

*(ii) (both are same)

(It is not a function)
Cin(Console input) - It is similar to scanf function
(It is an object) (C programing). It is used to read / input value of variable from the keyboard.
It's an object of istream class.

Syntax- cin>> variable_name;

Ex- cin>>a;

where >> is known as extraction or get from operator.

→
```
# include<iostream>
# include<iomanip>
using namespace std;
main()
{
    int a;
    cin >>a;
    cout << a;
}
```

or cout << "Enter Any number";
cin >> a;
or cout << "You have Entered:" <<a;

Output- 10
10

Output- Enter Any number 18
You have Entered: 18

Operators in C++

1- Arithmetic Operators
2- Relational operators
3- Logical Operators
4- Bitwise Operators
5- Precedence Operators
6- Special Operators
7- Escape Sequence.

1- Arithmetic Operators -

Operators	Meaning	Example	Result
+	Addition	4+2	6
-	Subtraction	4-2	2
*	Multiplication	4*2	8
/	Division	4/2	2
%	Modulus Operator to get remainder in integer division	5%2	1

→
```
#include <iostream>
#include <iomanip>
using namespace std;
main()
{
  int a=20 , b=10, c;
  c=a+b;   or c=a-b;   or c=a*b; or c=a/b; or c=a%b;
  cout << "Value of C = " << c << endl;
}
```

Output - Value of c = 30 or 10 or 200 or 2 or 0

2- Relational Operators -

Operators	Meaning	Example	Result
<	Less than	5<2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
!=	Not equal to	5!=2	True

→
```
#include<iostream>
#include<iomanip>
using namespace std;
main()
{
  int a=5, b=2, c;
  c = a>b;  or  c= a<b;  or c=a==b;  or  c=a<=b; or c=a>=b;
                                                  or c=a!=b;
  cout << "c=" << c << endl;
}
```

Output- c = 1 or c = 0 or c = 0 or c = 0 or c = 1 or c = 1

3- Logical Operators -

Operators	Meaning	Example	Result
&&	Logical and	(5>2)&&(5>3)	False
||	Logical or	(5>2)||(5>3)	True
!	Logical not	!(5>2)	True

(i) && (Logical And) -

Operand 1	Operand 2	Result
True	True	True
True	False	False
False	True	False
False	False	False

→
```
#include<iostream>
#include<iomanip>
using namespace std;
main()
{ int a=5, b=2, c=5, d=3, result;
  result = ((a<b)&&(c>d)); or ((a>b)&&(c>d)); or ((a>b)&&(c<d));
                                                  or ((a<b)&&(c<d));
  cout << result << endl;
}
```

Output- 0 or 1 or 0 or 0

(ii) (||) Logical or -

Operand 1	Operand 2	Result
True	True	True
True	~~Tor~~ False	True
False	True	True
False	False	False

→
```
#include <iostream>
#include <iomanip>
using namespace std;
main()
{
  int a=5, b=2, c=5, d=3, result;
  result =((a>b)||(c>d)); or ((a<b)||(c>d)); or ((a>b)||(c<d)); or ((a<b)||(c<d)
  cout<< result << endl;
}
```

Output- 1 or 1 or 1 or 0

(iii) ! (Logical Not) -

Operand	Result
False	True
True	False

→
```
#include <iostream>
#include <iomanip>
using namespace std;
main()
{ int a =5, b =2, result;
  result = !(a>b); or result = !(a < b);
  cout << result<< endl;
}
```

Output- 0 or 1

4-

4- Bitwise

<d)); or((a<b)||(c<d))

b);

Special Operators -

1- Increment and Decrement operator
2- Assignment operator
3- Sizeof operator
4- Comma (,) operator

1- Increment and Decrement operator-

Operator	Meaning
++a	Increment a by 1, then use the new value of a
a++	Use value of a, then increment a by 1
--b	Decrement b by 1, then use the new value of b
b--	Use the current value of b, then decrement by 1

→
```
#include <iostream>
using namespace std;
main ()
{
    int a=3;
    cout << a++ << endl;        or cout << a-- << endl;
    cout << a << endl;          or cout << a << endl;
    cout << ++a << endl;        or cout << --a << endl;
    cout << a << endl;          or cout << a << endl;
}
```

Output - 3
4
5
5

Output - 3
2
1
1

4- Comma Operator- (,)

→
```
#include <iostream>
using namespace std;
main ()
{ int a=10, b=20, d;
}
```

2- Assignment Operators-

Operator	Example	Equivalent Expression (m = 15)	Result
+ =	m+=10	m = m+10	25
- =	m-=10	m = m-10	5
* =	m*10	m = m*10	150
/=	m/=10	m = m/10	1
% =	m%=10	m = m%10	5

→
```
#include <iostream>
using namespace std;
main()
{
  int m=15;
m=m+10; or m+=10 | m=m-10; or m-=10; | m=m*10; or m*=10; |
                m=m/10; or m/=10; | m=m%10; or m%=10;
cout << "Value of M:" << m << endl;
}
```

Output- Value of M:25 or 25 | 5 or 5 | 150 or 150 | 1 or 1 | 5 or 5

3- Sizeof Operator-

→
```
#include <iostream>
using namespace std;
main()
{
 cout << sizeof(int);        or int a =10;
                             or cout << sizeof(a);
}
```
Output- 4 or 4

Escape Sequences-

Sequence	Meaning
\a	Audible bell (Alert)
\b	Backspace
\f	Formatted
\n	Newline or line feed
\r	Carriage return (Enter)
\t	Horizontal Tab
\v	Vertical Tab
\\	Print back slash
\?	Print question mark
\'	Print single quote
\"	Print double quote
\0	Null character

→
```
#include <iostream>
using namespace std;
int main().
{
  cout << " Geeky \n Shower "; or "Geeky \t Shower";
  return 0;
}
```

Output- Geeky
Shower or Geeky Shower

Comments in C++ -

Multiple line - /* ____ */

Single line - // ____

Mixed mode Operation and Automatic (implicit) Conversion-

Char → int → long int → float → double
(Low Ranking) (High Ranking)

Ex- int m=15;
float x=3.1, y;
y = m*x
→ Ste type float is high ranking
→ Ste type int is converted to float

→
```
#include <iostream>
using namespace std;
int main()
{
    int x=15;
    float m=3.1, y;
    y = m*x;
    cout <<"\nValue of Y:" << y << endl;
    return 0;
}
```

Output- Value of Y: 46.5

Cast or Explicit Conversion -

Syntax- (type) expression
or (type) (expression)

Ex-

```
int m=5;
float y;
y=(float)m/2
```

→

```
#include<iostream>
using namespace std;
int main()
{
  int x=5;
  float y;
  y=x/2;                    or y=(float)x/2;
  cout <<"Value of Y:" << Y << endl;
  return 0;
}
```

Output- ~~Varia~~
Value of Y: 2 or Value of Y: 2.5

Punctuators

Brackets [ ]	Opening and closing brackets indicate single and multidimensional array subscript.
Parenthesis ( )	Opening and closing brackets indicate functions calls, function parameters for grouping expressions etc.
Braces { }	Opening and closing braces indicate the start and end of a compound statement.
Comma ,	It is used as a separator in a function argument list.
Semicolon ;	It is used as a statement terminator.
Colon :	It indicates a labelled statement or conditional operator symbol.
Asterisk *	It is used in pointer declaration or as multiplication operator.
Equal sign =	It is used as an assignment operator.
Pound sign #	It is used as pre-processor directive.

Local and Global Variable -

```
#include <iostream>
using namespace std;
int a=10;                    // Global Variable
int main()
{
    int b=20;                // Local variable
    cout << "Value of A:" << a << endl;
    cout << "Value of B:" << b << endl;
    return 0;
}
```

Output - Value of A : 10
Value of B ; 20

if statement -

→
```
#include <iostream>
using namespace std;
int main()
{
    int a=5, b=2;
    if(a>b)                  or if(a<b)
    {
        cout << "A is greater." << endl;
    }
    return 0;
}
```

Output - A is greater. or

Nested if Statement-

```
#include <iostream>
using namespace std;
int main()
{
    int a=5, b=2, c=4, d=1;
    if (a>b)
    {
        if (c>d)                    or if (c<d)
        {
            cout << "Good Job" << endl;
        }
        if (c>d)
        {
            cout << "C greater:" << endl;
        }
    return 0;
}
```

Output- Good Job or C greater

if else Statement -

```
#include <iostream>
using namespace std;
int main()
{
  int a=5, b=2;
  if (a>b)                    or if (a < b)
  {
    cout<<"A is greater." <<endl;
  }
  else
  {
    cout << "B is greater." <<endl;
  }
  return 0;
}
```

Output- A is greater or B is greater

Else if Statement -

```
#include <iostream>
using namespace std;
int main()
{
    int a=5, b=7, c=6, d=9, f=8;
    if (a > b)
    {
      cout <<" A " << endl;
    }
  else if (b > c)
    {
      cout <<" B" << endl;
    }
  else if (c > d)
    {
      cout <<" C" << endl;
    }
  else
    {
      cout <<" F" << endl;
    }
  return 0;
}
```

Output - B.

for loop-

Syntax- for(initialization ; condition ; increase / decrease)
{
block of statements;
}

→
```
#include <iostream>
using namespace std;
int main()
{
    int i;
    for (i=0; i<10; i++)
    {
        cout <<'\t' <<i;
    }
    return 0;
}
```

Output- 0 1 2 3 4 5 6 7 8

→
```
#include <iostream>
using namespace std;
int main()
{
    int a=0 , i;
    for (i=0; i<10; i++)
    {
        a= a+2;
        cout << a << '\n';
    }
    return 0;
}
```

Output-
```
2
4
6
8
10
12
14
16
18
20
```

Nested for loop-

→
```
#include <iostream>
using namespace std;
int main()
{
    int i, j;
    for (i=1;i<=4 ; i++)
    {
        for ( j=1; j<=5; j++)
        {
            cout << j;
        }
        cout<<endl;
    }
    return 0;
}
```

Output-
```
12345
12345
12345
12345
```

While Loop - इसका Use तब करते हैं जब ये न पता हो की Loop कितनी बार चलेगा।

Syntax

```
initialization;
while (Condition)
{
    block of statement;
    increment / decrement;
}
```

→

```
#include<iostream>
using namespace std;
int main()
{
  int i=0;
  while (i<10)
  {
    cout << i <<'\n';
    i++;
  }
  return 0;
}
```

Output -

```
0
1
2
3
4
5
6
7
8
9
```

→
```
#include <iostream>
using namespace std;
int main()
{
    int i=0;
    while (1)                    // 1 means true
    {
      if (i==10)
        {
          break;
        }
      i++;
      cout << i << '\n';
    }
    return 0;
}
```

Output-
```
0
1
2
3
4
5
6
7
8
9
```

→
```
#include<iostream
using namespace std;
int main()
{
    int i;
    cout << "Enter 0 to exit \n\n";
    while (1)
    {
      cout <<"Enter any value: ";
      cin >> i;
      ~~cout << "You Entered:" << i << endl << endl;~~
      if (i==0)
      {
        cout <<"Program Exit. \n\n";
        break;
      }
      cout <<"You Entered:" << i << endl << endl;
    }
  return 0;
}
```

Output- Enter 0 to exit
Enter any value: 324234
You Entered: 324234
Enter any value: 354543
You Entered: 354543
Enter any value: 65756
You Entered: 65756
Enter any value: 0
Program Exit.

Note- user जितनी बार चाहे उतनी बार Loop चला सकता है।

Do While Loop -

Syntax-

```
do
{
    block of Statement;
}
while (condition);
```

→

```
#include<iostream>
using namespace std;
int main()
{
  int a;
  do
  {
    Cout << "Enter any number:";
    Cin >>a;
    Cout << "You entered:" << a << endl;
  } while(a!=0);
  return 0;
}
```

Output- Enter any number: 53454
You entered : 53454
Enter any number: 23423
You entered :23423
Enter any number: 67
You entered : 67
Enter any number: 435345
You entered:435345
Enter any number: 0
You entered: 0

Break Statement-

→
```
#include <iostream>
using namespace std;
int main()
{
    int n;
    for (n=1; n<10; n++)
    {
        cout << n<<'\t';
        if ( n==7)
        {
            cout <<" Countdown aborted";
            break;
        }
    }
    cout <<'\n';
    return 0;
}
```

Output-

1 2 3 4 5 6 7 Countdown aborted

Continue Statement -

```
#include <iostream>
using namespace std;
int main ()
{
    int n;
    for (n=1; n<=10; n++)
    {
        if (n==7)
        {
            continue;
        }
        cout << n << '\t';
    }
    return 0;
}
```

Output-

1 2 3 4 5 6 8 9 10

Switch Statement-

```
#include <iostream>
using namespace std;
int main()
{
  int a;
  cout << "Enter Your choice(1 or 2): ";
  cin >> a;
  switch(a)
  {
    case 1:
    {
      cout << "I am not good";
    } break;
    case 2:
    {
      cout << "I am bad";
      break;
    }
    default:
    {
      cout << "Wrong choice dude!";
      break;
    }
  }
  return 0;
}
```

Output- (i) Enter your choice(1 or 2): 1
I am not good
(ii) Enter Your choice(1 or 2): 2
I am bad
(iii) Enter Your choice(1 or 2): 4
Wrong choice dude!

One D char Array for string -

```
#include <iostream>
using namespace std;
int main()
{
    char name [20]= " geekyshows";  or  char name [20];
                                        or  cin >> name;
    cout << name;
    return 0;
}
```

Output- geekyshows or geekyshows
geekyshows

Two D CharArray for string -

```
#include <iostream>
using namespace std;
int main()
{
    char name [2] [20] = { "geekyshows", "hello"};
    cout << name [0];   or  cout << name [1];
    return 0;
}
```

Output- geekyshows or hello

→
```
#include <iostream>
using namespace std;
int main()
{
    char name[2][20];
    cin >> name[0] >> name[1];
    cout << name[0] <<'\t' << name[1];
    return 0;
}
```

Output- geeky
show
geeky show

→
```
#include <iostream>
using namespace std;
int main()
{
    char name[2][20];
    int i;
    for (i=0; i<2; i++)
    {
        cout << "Enter String:");
        cin >> name[i];
    }
    for (i=0; i<2; i++)
    {
        cout << name[i] << endl;
    }
    return 0;
}
```

Output- Enter String: geeky
Enter String: show
geeky
show

Two D Array in int -

→
```
#include<iostream>
using namespace std;
int main()
{
    int a[2][3] = {{1,2,3},{4,5,6}};
    cout << a[1][2];
    return 0;
}
```

Output- 6

→
```
#include<iostream>
using namespace std;
int main()
{
   int a[2][3] = {{1,2,3},{4,5,6}};
   int i,j;
   for(i=0; i<2; i++)
   {
      for(j=0; j<2; j++)
      {
        cout<< a[i][j] <<"  ";
      }
      cout << '\n';
   }
   return 0;
}
```

Output- 1 2 3
4 5 6

→
```
#includi<iostream>
using namespace std;
int main()
{
    int a[2][3];
    int i, j;
    for (i=0; i<2; j++)
    {
        for ( j= 0; j<3; j++)
        {
            cin >> a[i][j];
        }
    }
    for (i=0; i<2; i++)
    {
        for( j=0; j<3; j++)
        {
            cout << a[i][j] << "  ";
        }
        cout << '\n';
    }
    return 0;
}
```

Output-
```
1
2
3
4
5
6
1  2  3
4  5  6
```

Function

Function Prototype Declaration -

Syntax- datatype function_name (type d1, type d2, ... type dn);

Ex- int fact(int a);
int add (int a, int b);
int add(int, int);

Function Declaration -

Syntax - datatype Function_name(type argu_1, type argu_2, ..., type argu_n)

```
{
    Local Declaration;
    block of statement;
    return(variable or expression);
}
```

Function Body

Note- No data-type = integer by default

Ex-
```
int fact(int n, int m)
{
    int c;
    c = n+m;
    return(c);
}
```

Return Statement -

Syntax- return (3); ✓
return (x+y); ✓
return (y); ✓
~~return~~ return (2,4); ✗
return (x,y); ✗

1- No argument but return integer value-
→ int show(void);

2- Argument available but return no value-
→ void add (int, int);

3- No argument No return-
→ void add (void);

4- Arguments available and return an integer value-
→ square (int a);
or int square (int a);

Actual and Formal Argument-

→ Function call arguments are actual arguments
→ Function declaration arguments are formal arguments
Actual arguments value ⟹ Formal arguments value.

Calling a Function-

Syntax- function_name( );
Ex- add ();

Category of Function Calling-

1- With no arguments and with no return value
2- With no arguments and with return value
3- With arguments and with no return value
4- With arguments and with return value

4→
```
#include<iostream>
using namespace std;
int add (int, int);
int main()
{
    int a, b, sum;
    cout << "Enter Two Numbers: ";
    cin >> a >> b;
    sum = add(a, b);
    cout << "Total = " << sum;
    return 0;
}
int add (int x, int y)
{
    int t;
    t = x+y;
    return (t);
}
```

Output- Enter Two Numbers: 2
3
Total =5

Note- यदि हम Function Prototype को नहीं लिखना चाहे तो Function Prototype के स्थान पर Function Definition लिखना होगा।

1→
```
#include <iostream>
using namespace std;
void disp (void);
int main()
{
    disp ();
    return 0;
}
void disp (void)
{
   cout << "I am Void";
}
```

Output- I am void

Inline Function-

Function declaration-

Syntax- inline return type function name (arguments)
```
{
    function body;
}
```

Ex-
```
inline int add (int a, int b)
{
    return (a+b);
}
```

Note- इसका use Control time को reduce करने के लिये किया जाता है। इस में Function call के दौरान Control function definition के पास नहीं जाता बल्कि function call में ये function return से replace कर देता है ये यदि Function definition में बड़ा हो (Loop, condition, function recursion) में काम नहीं करता पर इनमें Error नहीं आयेगा। Program पहले जैसा ही प्रोग्राम (प्रोग्राम)

Note-

→
```
#include<iostream>
using namespace std;
inline int add (int a, int b) // Inline function
{
 return (a+b);
}
int main ()
{
  int x = 10, y = 20;
  cout << add (x,y) << endl; // Function call
  return 0;
}
```

Output - 30

Note - यह Single या फिर 2, 3 Line के Function definition में हि काम करता है।

Pointer-

→
```
#include<iostream>
using namespace std;
{
    int a = 10;
    int *p;
    p = &a;
                    or *p = 20;
    cout << "Address of a: " << p << endl;
    cout << "Value of a: " << *p << endl;
    return 0;
}
```

Output- Address of a: 0x 28fef8
Value of a: 10 or Value of a: 20

Structure -

```
#include < iostream >
using namespace std;
struct student
{
    int roll;
    char name[20];
};
int main ()
{
    struct student stu;
    cout << "Enter Name : ";
    cin >> stu.name;
    cout << "Enter Roll : ";
    cin >> stu.roll;
    cout << "Name : " << stu.name << endl;
    cout << "Roll : " << stu.roll << endl;
    return 0;
}
```

a:20

Output - Enter Name : Geekyshows
Enter Roll : 101
Name : Geekyshows
Roll : 101

Union -

→
```
#include <iostream>
using namespace std;
union student,
{
    int roll;
    char name[20];
};
int main()
{
  union student stu;
  cout << "Enter Name:";
  cin >> stu.name;
  cout << "Enter Roll: ";
  cin >> stu.roll;
  cout << "Name : " << stu.name << endl;
  cout << "Roll: " << stu.roll << endl;
  return 0;
}
```

Output- Enter Name : Geekyshows
Enter Roll : 101
Name : e
Roll: 101

→
```
#include<iostream>
using namespace std;
union student;
{
    int roll;
    char name[20];
};
int main()
{
    union student stu;
    Cout <<"Enter Name:";
    cin >> stu.name;
    cout <<" Name :" << stu.name << endl;
    Cout << "Enter Roll:";
    cin >> stu.roll;
    cout << "Roll: " << stu.roll << endl;
                or Cout <"Name:" << stu.name << endl;
    return 0;
}
```

Output- Enter Name: Geekyshows
Name: Geekyshows
Enter Roll: 101
Roll: 101
or Name: e

Procedure Programing
Look in C programing:-

- All Data Type (int, char, float, double etc)
- Arithmetic Operators
- Library Functions
- Relation Operations
- Logical Operators
- Bitwise Operators
- Increment / Decrement Operators
- Assignment Operators
- Sizeof operator
- Comma Operator
- Mixed Mode Operation
- Explicit Conversion
- Symbolic Constant
- Address of Operators
- Escape Sequence
- Multiple Assignment
- getch, getchar, getche, putchar, putch functions
- Everything related to If Statement
- Switch, Break, Continue Statement
- All Loop Statements
- All Topics related to Array
- String
- All Topics related to Function
- Recursion
- All Topic related to Structure
- All Topic related to Union
- All Topic related to Pointer

Object-Oriented Programing (OOP) Concepts-

→ Class and Objects
→ Encapsulation
→ Information Hiding
→ Abstraction (data)
→ Message Passing
→ Interface
→ Association
→ Inheritance
→ Polymorphism

Note-

Class- A class is a group of common behavior and functions.
A class is a user defined datatype like any other built-in data type for example int, char, float, ... etc.

Class ⇒ Tablet

Attributes	Action
• RAM	• Internet()
• Processor	• Gaming()
• Memory	• Call()
• CAM	• Message()
• OS	• Music()

Samsung, Intex, iBall, Micromax } Object

Defining a Class-

Syntax-

```
class Class_name
{
    private:
        Datatype variable_name;
        Return_type function_name (argument);
    public:
        Datatype variable_name;
        Return_type function_name (argument)
        {
            Function body;
        }
    protected:
        Datatype variable_name;
        Return_type function_name (argument);
};
```

Note- private and protected के variable का उपयोग class के बाहर नही कर सकते तथा public के variable का use class के बाहर कर सकते है।

यदि class मे variable के आगे कुछ न लगाये (private, protected, public) तो वह by default private हो जाता है। private लिखना Optional है। इसे लिखे या न लिखे।

Example-

```
class student
{
    private:
        int id;
        char name [20];
    public
        void Getdata(void);
        void display (void)
        {
            cout << id << '\t' << name << endl;
        }
};
```

Object - It is class type variable.

Object Declaration -

Mathod 1 -

Syntax - class student
{
..
..
..
} stu1, stu2, stu3;

where stu1, stu2, stu3 are objects (class type variable)

Mathod 2 - In main function -

Syntax - class ClassName objectName 1, objectName 2, ...;
or ClassName objectName 1, objectName 2, ...;

Note - class keyword optional है।

Ex - class student stu1;
or
student stu1;

```
#include <iostream>
using namespace std;
class student
{
    private:
        int id;                          } Data Members
        char name[20];
    public:
        void Getdata(void);              }
        void display (void)               }
        {                                 } Member
            cout << id <<'\t' <<name<<endl;  } Functions
        }                                 }
};
int main()
{
    student stu1, stu2;        { Objects }
    . . . . . . . . . . ;
    . . . . . . . . . . ;
```

Note- जो भी variable class में declare करते हैं चाहे वो किसी का भी हो उसे private, public, protected Data Members कहते हैं इसी प्रकार सभी Function को Member Functions कहते हैं चाहे वो Class में किसी भी जगह Declare हुआ हो।

पर ज्यादातर हम Data Members को हम private रखते हैं और Member Functions को हम public में लिखते हैं। ताकि हम Data Members को Class के बाहर Member Functions के through access कर सके।

Encapsulation- Encapsulation is the process of enclosing within classes and objects the attributes and the methods. in other word encapsulation allow wrapping up of data and functions into a sigle unit.

Advantage-

Data hiding- Restrictions of external access to features of a class.

Information hiding- The implementation details are not known to the user.

Implementation independence- A change in the implementation is done easily without affecting the User's interface.

classes
and
objects

Access Specifiers / Accessibility Rules / Lables -

- Private - No Direct Access
- Protected - No Direct Access
- Public - Direct Access

Access-control specifiers	Accessible to Own class members	Object of a class
Private	Yes	No
Protected	Yes	No
Public	Yes	Yes

Access Class Member using Objects -

→
```
#include<iostream>
using namespace std;
class student
{
    private:
        int id;
        char name[20];
    public:
        int a;
        void getdata(void);
        void display(void)
        {
            cout << id << '\t' << name << endl;
        }
};
int main()
{
    student stu1;
    - - - - - - -;
    - - - - - - -;
```

Syntax- Object_name.data_member
Object_name.member_function

Ex- stu1.a;
stu1.getdata();
stu1.display();

stu1.id; (X) (Object के through हम private data को Access नहीं कर सकते हैं।)

Note- Object के through हम केवल public के data को ही Access कर सकते हैं।

Member Function – A member function performs an operation required by the class. It may be used to read, manipulate, or display the data member.

Member functions of a class can be defined in two places:

(i) Inside the class definition
(ii) Outside the class definition

(i) **Inside The Class Definition** –

```
#include <iostream>
using namespace std;
class student
{
  private :
        int roll;
        char name[20];
  public :
        void getdata (void);
        void display (void)
        {
          cout << roll << endl;
        }
};
int main ()
{
    student stu1;
    . . . . . .;
    . . . . . .;
```

Inside The class Definition

Note - जब हम हम Member Function को Inside the class define करते हैं तो उसे inline function कि तरह treat किया जाता है। इसलिये यदि Function definition एक या दो line का है तो हि उसे Inside the class define करना चाहिये।

(ii) Outside The class Definition-

Syntax- return_type class_name :: function_name (argument declaration)
{
Function_body
}

(where :: → Scope resolution operator)

Ex-

```
→ #include <iostream>
using namespace std;
class student
{
    private :
        int roll;
        char name[20];
    public :
        void getdata (void);
        void display (void);
};
void student :: getdata (void)
{
    cout << "Enter Name : ";
    cin >> name;
    cout << "Enter Roll";
    cin >> roll;
}
int main ()
{
    student stu1;
    - - - - - - - - ;
    - - - - - - - - ;
```

Note- Outside the class Definition को class और main() function के बीच में लिखते हैं।

```
#include <iostream>
using namespace std;
class student
{
    private:
            char name[20];
            int roll;
    public:
            void getdata();
            void display()
            {
                cout << "Name:" << name << endl;
                cout << "Roll: " << roll << endl;
            }
};
void student :: getdata(void)
{
    cout << "Enter Name:";
    cin >> name;
    cout << "Enter Roll:";
    cin >> roll;
}
int main()
{
  student stu;
  stu.getdata();
  stu.display();
  return 0;
}
```

Output
```
Enter Name: Geekyshows
Enter Roll: 101
Name: Geekyshows
Roll: 101
```

Nesting of Member Function -

```
#include <iostream>
using namespace std;
class number
{
    int a;
    public :
        void getdata (void);
        int task (void);
        void display (void);
};
void number :: getdata(void)
{
    cout << " Enter Any Number : ";
    cin >> a;
}
int number :: task(void)
{
    int x;
    if (a % 5 == 0)
    {
        x = 0;
        return (x);
    }
    else
    {
        x = a % 5;
        return (x);
    }
}
```

P.T.O.

```
void number :: display (void)
{
    cout << "Remainder =" << task() << endl;
}
int main()
{
    number num;
    num.getdata();
    num.display();
    return 0;
}
```

Output-

(i) Enter Any Number : 10
Remainder = 0

(ii) Enter Any Number : 7
Remainder = 2

Function (Pass by Value)-

```
#include <iostream>
using namespace std;
int add (int, int);
int main()
{
    int a=10, b=20, sum;
    sum= add (a, b);
    cout<<"Addition = " <<sum << endl;
}
int add (int x, int y)
{
```

```
        int total;
        total = x+y;
        return 0;
}
```

Output- Addition= 30

Function (Pass by Address)-

→
```
#include <iostream>
using namespace std;
void age (int *, int *);
int main()
{
    int rani = 10, amit = 20;
    age (& rani, & amit);
    cout <<"Rani's Age :" << rani << endl;
    cout << "Amit's Age : " << amit << endl;
    return 0;
}
void age (int *r, int *a)
{
    *r = 20;
    *a = 10;
}
```

Output- Rani's Age : 20
Amit's Age : 10

Refrence Variable -

(एक ही Variable का दूसरा नाम)

→
```
#include <iostream>
using namespace std;
int main()
{
    int a=10;
    int &nicka=a;
    cout << "Value of a :" << a << endl;
    cout << "Value of nicka :" <<nicka<< endl;
    cout <<"Address of a :" << &a << endl;
    cout <<"Address of nicka : " <<&nicka << endl;
    return 0;
}
```

Output- Value of a : 10
Value of nicka : 10
Address of a : 0x28f1f8
Address of nicka : 0x28f1f8

Syntax- &newname_of_variable

Pass by Reference -

ii) Function prototype -
Syntax- return-type function name (datatype &1, datatype &2 ---);

iii) Function Call - function_name (variable 1, variable 2 ---);

iii) Function Definition -

Syntax- returntype function_name (datatype &variable 1, ---)

* Function (Pass by Reference) -

→
```
#include <iostream>
using namespace std;
void age(int & , int&);
int main()
{
    int rani=10, amit=20;
    age (rani, amit);
    cout<<"Rani's Age:" << rani << endl;
    cout<<"Amit's Age:" << amit << endl;
    return 0;
}
void age (int &r, int &a)
{
    r= 50;
    a = 30;
}
```

Output- Rani's Age: 50
Amit's Age: 30

Note- Pass by Reference में Variable को Null से initialize ~~कर~~ नहीं कर सकते हैं तथा reference variable का Array के साथ use नहीं कर सकते हैं।
जबकि Pass by Address में ये सब कर सकते हैं।

Array within a Class-

Class definition syntax-

```
class class name
{
        data member; // array type
        data_type variable_name[size];
    public:
        data member;
        member function;
};
```

Ex-

```
class student
{
        int roll[SIZE];    // array within class
        char name[SIZE][20];
    public:
        void getdata(void);
        void display(void);
};
```

Function definition-

```
void student :: getdata(void)
{
                                (use for loop)
    cin >> name[i];

    cin >> roll[i];
}
```

```
void student :: display(void)
{
                                (use for loop)
    cout << name [i];

    cout << roll [i];
}
```

Object declaration and Function call-

```
student st;

st.getdata();
st.display();
```

```
#include <iostream>
#include <conio.h>
#define SIZE 3
using namespace std;
class student
{
    int roll [SIZE];
    char name [SIZE][20];
public :
    void getdata (void);
    void display (void);
};
void student :: getdata (void)
{
    for (int i = 0 ; i < SIZE ; i++)
    {
        cout <<"\n Enter Name:";
        cin >> name[i];
        cout << "Enter Roll: ";
        cin >> roll[i];
    }
}
void student :: display (void)
{
    for (int i = 0; i< SIZE ; i++)
    {
        cout <<"\n Name: " << name [i] << endl;
        cout <<" Roll : " << roll [i] << endl;
        getch ();
    }
}
```

```
int main()
{
    student st;
    st.getdata();
    st.display();
    return 0;
}
```

Output-

```
Enter Name: raja
Enter Roll: 101

Enter Name: rani
Enter Roll: 102

Enter Name: sahil
Enter Roll: 103

Name: raja
Roll: 101

Name: rani
Roll: 102

Name: sahil
Roll: 103
```

Array of object -

Class definition Syntax-

```
class class_name.
{
        data member;
        member function;
    public;
        data member;
        member function;
};
```

Ex-

```
class company
{
        int id;
        char name[20];
    public;
        void getdata(void);
        void display(void);
};
```

Function definition-

```
void company :: getdata (void)
{
    cin >> name;
    cin >> id;
}

void company :: display (void)
{
    cout << name;
    cout << id;
}
```

Object declaration and Function call -

```
Company manager[size], HR[size];
manager[i].getdata(); // with for loop
HR[i].getdata(); // with for loop
manager[i].display(); // with for loop
HR[i].display(); // with for loop
```

→

```
#include<iostream>
#include<conio.h>
#define SIZE 3
using namespace std;
class student
{
    int roll;
    char name[20];
    public:
      void getdata(void);
      void display(void);
};
void student :: getdata(void)
{
      cout << "\n Enter Name: ";
      cin >> name;
      cout << " Enter Roll: ";
      cin >> roll;
}
void student :: display(void)
{
      cout << "\n Name: " << name << endl;
      cout << " Roll: " << roll << endl;
```

```
        cout<<"Press Enter to continue....."<<endl;
        getch();
}
int main()
{
  student st[SIZE];
  for(int i=0; i<SIZE; i++)
  {
      cout<<"\nEnter Details of Student:"<<i+1;
      st[i].getdata();
  }
  cout<<endl;
  for(int i=0; i<SIZE; i++)
  {
     cout<<"\n Details of Student:"<<i+1;
     st[i].display();
  }
  return 0;
}
```

Output- Enter Details of Student: 1
Enter ~~Details~~ Name: raja
Enter Roll: 101
Enter Details of Student: 2
Enter Name: rani
Enter Roll: 102
Enter Details of Student: 3
Enter Name: ram
Enter Roll: 103
Details of Student: 1
Name : raja
Roll : 101
Press Enter to continue.....

```
Details of Student : 2
Name : rani
Roll : 102
Press Enter to continue.....
Details of Student : 3
Name : ram
Roll : 103
Press Enter to continue....
```

```
#include <iostream>
#include <conio.h>
using namespace std;
class company
{
    char id [10];
    char name [20];
    char dep [20];
  public:
    void getdata (void);
    void display (void);
};
void company :: getdata (void)
{
    cout <<"\n Enter Name: ";
    cin >> name;
    cout<< "Enter id:";
    cin >> id;
    cout << "Enter Department: ";
    cin >> dep;
}
void company :: display (void)
```

```
{
    cout <<"\nID No. :" <<id << endl;
    cout << "Name : " << name <<endl;
    cout << "Department :" << dep<< endl;
    getch();
}
int   main()
{
    company   manager[3], teamlead [5];
    for (int i=0; i<3; i++)
    {
      cout <<"\nEnter Details of Manager:" << i+1;
      manager [i]. getdata();
    }
    for (int i=0; i<5; i++)
    {
      cout <<"\nEnter Details of Team Leader:"<i+1;
    }
    cout << endl;
    for (int i=0; i<3;i++)
    {
      cout <<"\n Details of Manager :" << i+1;
      manager[i]. display ();
    }
    for (int i=0; i<5; i++)
    {
      cout<<" \nDetails of Team Leader:" << i+1;
      teamlead [i]. display ();
    }
    return 0;
}
```

Output -
```
Enter Details of Manager : 1
Enter Name : raja
Enter id : AB123
Enter Department : Linux
Enter Details of Manager : 2
Enter Name : rani
Enter id : CD123
Enter Department : Windows
Enter Details of Manager : 3
Enter Name : Yang
Enter id : EF123
Enter Department : Admin
Enter Details of Team leader : 1
Enter Name : ram
Enter id : AB67
Enter Department : Linux
Enter Details of Team Leader : 2
Enter Name : rocky
Enter id : AB54
Enter Department : Linux
Enter Details of Team Leader : 3
Enter Name : sunaina
Enter id : CD87
Enter Department : windows
Enter Details of Team Leader : 4
Enter Name : naina
Enter id : EF89
Enter Department : Admin
Enter Details of Team Leader : 5
Enter Name : rahul
Enter id : CD80
Enter Department : windows
```

Details of Manager : 1
ID No. : AB123
Name : raja
Department : Linux
Details of Manager : 2
ID No. : CD123
Name : rani
Department : windows
Details of Manager : 3
ID No. : EF123
Name : yug
Department : Admin
Details of Team Leader : 1
ID No. : AB67
Name : ram
Department : Linux
Details of Team Leader : 2
ID No. : AB54
Name : rocky
Department : Linux
Details of Team Leader : 3
ID No. : CD87
Name : sunaina
Department : windows
Details of Team Leader : 4
ID No. : EF89
Name : raina
Department : Admin
Details of Team Leader : 5
ID No. : CD80
Name : rahul
Department : windows

Object as Function Arguments

1- Pass by Value -

Class definition syntax-

Syntax- class class_name

```
{
        data member;
    public:
        data member;
        member function;
        data_type function_name(class_name, class_name);
};
```

Ex-

```
class total
{
    int n;
public:
    void getdata(void);
    void putdata(void);
    void add(total, total);
```

Function definition Syntax-

```
void class_name :: function_name(class name object, class name object 1)
{
    function body; // do work using above objects
}
```

Ex-

```
void total :: add(total x, total y) // x and y are objects
{
    n = x.n + y.n;
}
```

Object declaration and Function call or Passing Object (Value) -

Syntax-
```
int main()
{
    total r, s, result;
    r. getdata();
    s. getdata();
    r. putdata();
    s. putdata();
    result. add (r, s); // Passing objects
    result. putdata();
}
```

→
```
#include <iostream>
using namespace std;
class total
{
    int n;
  public:
    void getdata (void);
    void putdata (void);
    void add (total, total);
};
void total :: get data (void)
{
    cout << "Enter number:";
    cin >> n;
}
Void total :: putdata (void)
{
    cout << n << endl;
}
```

```
void total :: add (total x, total y)
{
    n = x.n + y.n;
}
int main()
{
    total x, s, result;
    x.getdata();
    s.getdata();
    cout << "Value of A:";
    x.putdata();
    cout << "Value of B:";
    s.putdata();
    result.add(x, s);
    cout << "Addition :";
    result.putdata();
}
```

Output- Enter numbers : 2
Enter numbers : 3
Value of A:2
Value of B:3
Addition : 5

→
```
#include <iostream>
using namespace std;
class matrix
{
        int item[3][3];
    public:
        void getdata (void);
        void putdata (void);
        void add(matrix, matrix);
};
void matrix :: getdata (void)
{
    for(int i=0; i<3; i++)
    {
        for(int j=0; j<3; j++)
        {
            cin >> item [i][j];
        }
    }
}
void matrix :: putdata (void)
{
    for (int i=0; i<3; i++)
    {
        for (int j=0; j<3; j++)
        {
            cout << item[i][j] << '\t';
        }
        cout << endl;
    }
}
```

```
void matrix :: add (matrix m, matrix n)
{
        int i, j;
        for (i=0; i<3; i++)
        {
                for (j=0; j<3; j++)
                {
                        item[i][j] = m.item[i][j] + n.item[i][j]
                }
        }
}
int main()
{
        matrix a, b, result;
        cout << "Matrix A: \n";
        a.getdata();
        cout << "Matrix B: \n";
        b.getdata();
        cout << "You Entered Matrix A: \n";
        a.putdata();
        cout << "You Entered Matrix B: \n";
        b.putdata();
        result.add(a, b);
        cout << " Addition : \n";
        ~~return~~ result.putdata();
        return 0;
}
```

P.T.O.

Output-

```
Matrix A:
1
2
3
4
5
6
7
8
9
Matrix B:
11
22
33
44
55
66
77
88
99
You Entered Matrix A:
1    2    3
4    5    6
7    8    9
You Entered Matrix B:
11   22   33
44   55   66
77   88   99
Addition:
12   24   36
48   60   72
84   96   108
```

Objects as Function Arguments-

2- Pass by Reference -

Class definition syntax-

```
class class_name
{
        data member;
    public:
        data member;
        member function;
        data_type function_name (class_name &, class_name &);
};
```

Ex-

```
class total
{
        int n;
    public:
        void getdata(void);
        void putdata (void);
        void add (total&, total &);
}
```

Function definition Syntax-

```
void class_name :: function_name (class_name &object, class_name
                                  &object1)
{
    function body; // do work using above objects
}
```

Ex-

```
void total :: add (total &x, total &y) // x and y are objects
{
    n = x.n + y.n;
}
```

Objects as Function Arguments - Pass by Reference -

Object declaration and Function call as Passing object (reference) -

Syntax - int main()

```
{
    total r, s, result;
    r.getdata();
    s.getdata();
    result.add(r,s); // Passing Objects
    r.putdata();
    s.putdata();
    result.putdata();
}
```

→

```
#include <iostream>
using namespace std;
class total
{
    int n;
  public:
    void getdata(void);
    void putdata(void);
    void add(total &, total &);
};
void total:: getdata(void)
{
    cout << "Enter numbers:";
    cin >> n;
}
```

```
void total :: putdata (void)
{
    cout << n << endl;
}
void total :: add (total &x, total &y) // x and y are objects
{                                  or  x.n = 10;
                                       y.n = 20;
    n=x.n + y.n;
}
int main ()
{
    total r, s, result;
    r.getdata ();
    s.getdata ();
    result.add (r, s);
    cout <<" Value of A: ";
    r.putdata ();
    cout << " Value of B: ";
    s.putdata ();
    cout << "Addition : ";
    result. putdata ();
}
```

Output- Enter numbers : 2
Enter numbers : 3
Value of A: 2
Value of B: 3
Addition: 5

or Output- Enter numbers: 2
Enter numbers: 3
Value of A: 10
Value of B: 20
Addition: 30

Returning Object-

Class Definition Syntax-

```
class class_name
{
  .....
  .....
  public:
  .....
    class_name function_name( class_name, class_name);
}
```

Ex-
```
class matrix
{
    int a;
    public:
      void getdata();
      void putdata();
      matrix add (matrix, matrix);
}
```

Function Declaration Syntax-

```
class_name class_name :: function_name( class_name object)
{
  class_name object;// create new object
   task;
   return (Object);
}
```

Ex-
```
matrix matrix :: add( matrix m, matrix n)
{
    matrix h;
    h.n = m.a + n.a;
    return (h);
}
```

Function Call –

```
matrix a, b, c, ~~auto~~ result;
    result = c.add (a, b);
    result.putdata ();
```

→

```
#include <iostream>
using namespace std;
class matrix
{
    int item[3][3];
  public:
        void getdata (void);
        void putdata (void);
        matrix add (matrix , matrix);
};
void matrix :: getdata(void)
{
    for (int i=0; i<3; i++)
    {
        for (int j=0; j<3; j++)
        {
            cin>>item [i][j];
        }
    }
matrix matrix :: add (matrix m, matrix n)
{
    matrix h;
    int i, j;
    for (i=0; i<3; i++)
    {
        for (j=0; j<3; j++)
```

```
            {
                h.item[i][j]=m.item[i][j]+n.item[i][j];
            }
        }
        return (h);
}
void matrix :: putdata (void)
{
    for (int i=0; i<3; i++)
    {
        for (int j=0; j<3; j++)
        {
            cout << item[i][j] << "\t";
        }
        cout << endl;
    }
}
int main()
{
    matrix a, b, c, result;
    cout << "Matrix A:\n";
    a.getdata();
    cout << "Matrix B:\n";
    b.getdata();
    cout << "You Entered Matrix A:\n";
    a.putdata();
    cout << "You Entered Matrix B:\n";
    b.putdata();
    result = c.add(a,b);
    cout << "Addition:\n";
    result.putdata();
    return 0;
}
```

Output- Matrix A:

1
2
3
4
5
6
7
8
9

Matrix B:

11
22
33
44
55
66
77
88
99

You Entered Matrix A:

1	2	3
4	5	6
7	8	9

You Entered Matrix B:

11	22	33
44	55	66
77	88	99

Addition:

12	24	36
48	60	72
84	96	108

Returning Object -

Class Definition with function definition Syntax-

```
class class_name
{
    ....

    public:
    ...

    class_name function_name (class_name object)
    {
        class_name object;
        ...
        return (object);
    }
}
```

Ex-

```
class matrix
{
    public:
        matrix add (matrix m, matrix n)
        {
            matrix h;
            h.n = m.a;
            return (h);
        }
}
```

Function call -

```
matrix a, b, c, result;
result = c.add(a, b);
result.putdata();
```

→ #include <iostream>

```
using    namespace    std;
class    matrix
{
        int    item [3] [3];
    public :
        void    getdata (void);
        void    putdata (void);
        matrix add (matrix m, matrix n)
    {
        matrix h;
        int  i, j;
        for (i=0; i<3; i ++)
        { for (j=0; j<3; j++)
          h.item [i][j] = m.item [i][j] + n.item [i][j];
        }
        return (h);
    }
};
void    matrix ~~(void)~~ :: getdata (void)
{
    for (int i=0 ; i<3 ; i++)
    {
        for (int j=0; j<3 ; j++)
        {
            cin >> item [i][j];
        }
    }
}
```

P. T.O.

```
void        matrix :: putdata (void)
{
    for(int i=0; i<3; i++)
    {
        for (int j=0; j<3; j++)
        {
            cout << item [i][j] << '\t';
        }
        cout << endl;
    }
}

int    main()
{
    matrix    a, b, c, result;
    cout << "Matrix A: \n";
    a. getdata();
    cout << "Matrix B: \n";
    b. getdata ();
    cout << "You Entered Matrix A: \n";
    a. putdata ();
    cout << "You Entered Matrix B: \n";
    b. putdata ();
    result = c. add (a, b);
    cout << "Addition : \n";
    result. putdata ();
    return 0;
}
```

Output -

Friend Function- इसका उपयोग करके हम Praivate या protected data Mumbers को direct access कर सकते हैं। यह भले हि class के inside लिखा जाता है किन्तु ये class का हिस्सा नहीं होता है। अत: इसे चाहे Praivate, public या protected के नीचे लिखे इससे इसको कोई फर्क नही पड़ता है। यह Non Mumber Function कहलाता है।

Class Definition Syntax-

```
class class_name
{
    - -
    - -

    public:
    - -
    - -

    friend return_type function_name(arguments);
    friend return_type function_name(class_name object);
}
```

Ex-

```
class total
{
    int n;
    - - -

    public:
        void getdata();
        void putdata();
        friend int add(void);
        friend int add(total k); // k is object
}
```

Function Definition Syntax-

For Arguments-

```
Return_type function_name(arguments)
{
    class_name object;
    object.function_name();
    return 0;
}
```

Ex-

```
int add(void)
{
    total k;
    k.n;
    k.getdata();
    return 0;
}
```

For Object -

```
Return_type function_name(class_name Object)
{
    object.function_name();
    return 0;
}
```

Ex-

```
int add(total k).
{
    k.n;
    k.getdata();
    return 0;
}
```

Function Call -

For Arguments -

add();

if it return any integer then:-

int sum;

sum = add();

For object -

add (p);

if it return any integer and passing object:-

int sum;

sum = add (p);

where p is object of main function

→

```
#include <iostream>
using namespace std;
class sum
{
    int a, b;
    public:
        void get_number (void); // friend function, It is non-member function
        friend function int add (void);
};
void sum:: get_number (void)
{
    cout << "Enter Any two Numbers: \n";
    cin >> a >> b;
}
int add (void) // function definition
{
    sum s;
    int temp;
```

```
    z.get_number();
    temp = z.a + z.b;  // accessing private data (a and b)
    return temp;                  from non member function
}                                 (friend fun)
int main()
{
    ~~Sum z;~~
    ~~int result;~~
    ~~z.get_number();~~
    ~~result = add(z);~~
    ~~cout << "Sum = " << result << endl;~~
    ~~return 0;~~
}
```

```
{
    int result;
    result = add();
    cout<<"Sum= " <<result<<endl;
    return 0;
}
```

Output- Enter Any two Numbers:
2
3
Sum =5

```
#include<iostream>
using namespace std;
class Sum
{
    int a, b;
    friend int add(void);
};
int add (void)
{
    Sum z;
    int temp;
    z.a =10
    z.b = 20
    temp = z.a + z.b;
    return temp;
}
```

```
int main()
{
    int result;
    result = add();
    cout << "Sum=" << result << endl;
    return 0;
}
```

Output- Sum = 30

object pass-

```
#include <iostream>
using namespace std;
class Sum
{
    int a, b;
  public:
    void get_number(void);
    friend int add(Sum s);
};
void Sum :: get_number(void)
{
    cout <<"Enter Any two Numbers: \n";
    cin >> a >> b;
}
int add(Sum s)
{
    int temp;
    temp = s.a + s.b;        //accessing private data
    return temp;
}
int main()
{
```

```
Sum z;
int result;
z.get_number();
result = add(z);
cout << "Sum = " << result << endl;
return 0;
}
```

Output- Enter Any two Numbers:
2
3
Sum = 5

A Function Friendly to Two or more Classes-

1- Member function of one class can be friend functions of another class.

Ex-

```
class rahul
{
    int a;
    int add();
};
class rani
{
    int b;
    friend int rahul::add(); //add() of class rahul
                                is friend of class rani

    - - - - -

};
```

```
#include<iostream>
using namespace std;
class rahul                    or class rani; (optional)
{
    private:
        int hi;
    public:
        void disp();
};
class rani
{
    private:
        int bye;
        friend void rahul :: disp ();
};
void rahul :: disp()
{
    rani love;
    rahul hate;
    love.bye = 143;
    hate.hi = 322;
    cout << "Rani says:" << love.bye << endl;
    cout << "Rahul says:" << hate.hi << endl;
}
int main()
{
    rahul what;
    what.disp();
    return 0;
}
```

Output- Rani says: 143
Rahul say: 322

2- Operate on Objects of two Different class using friend Function -

```
class rani; // forword declaration
class rahul
{
    int a;
    friend return_type function_name(All class_name separated by
                                                    camma);
    - - - - - -
};
class rani
{
    int b;
    friend return_type function_name(All class_name separated
                                                    by camma);
    - - - -
};
```

Function Definition Syntax-

```
Return_type function_name(class_name object, class_name1 object)
{
    object.function_name();
    return 0;
}
```

Ex-

```
void add(rahul m, rani n)
{
    m.a;
    n.b;
}
```

Function call-

```
rahul k;
rani v;
k.getdata();
v.getdata();
function_name(object, object)
```

Ex- disp(k,v);

→

```
#include <iostream>
using namespace std;
class rani;
class rahul
{
    int a;
  public:
      void getdata(void)
      {
          cout <<"Enter Rahul's Total Marks: ";
          cin >>a;
      }
    friend void add(rahul,rani);
};
class rani
{
      int b;
  public:
      void getdata(void)
      {
          cout<<"Enter Rani's Total Marks: ";
          cin >> b;
      }
```

```
    friend void add (rahul, rani);
};
void add (rahul m, rani n)
{
    cout << " Marks : " << m.a + n.b << endl;
}
int main ()
{
    rahul k;
    rani v;
    k.getdata ();
    v.getdata ();
    add (k, v);
}
```

Output - Enter Rahul's Total Marks: 100
Enter Rani's Total Marks: 200
Marks: 300

Note - यहाँ class rani को पहले ही declaration करना जरूरी है नहीं तो compailer को पहली बार class rani का ~~[illegible]~~ मतलब नहीं समझ में आयेगा।

3- All the member function of one class as the friend of another class -

```
class dell; // forward declaration
class logitech
{
        int product;
    friend dell; // all member function of logitech are
                                    friend of dell.
};
class dell
{
    public:
        void lappy (logitech a)
        {
            cout << "\n Laptop = " << a.lappy << endl; /* produ
            it private data of logitech class */
```

→

```
#include< iostream>
using namespace std;
class dell;
class logitech
{
        int product;
    public :
        void getdata (void )
        {
            cout << " Enter any number : ";
            cin >> product ;
        }
    friend dell; // friend class
};
```

```
class dell
{
    public:
        void lappy (logitech a) // accessing logitech class,
                                   private data.
        {
            cout <<"\nLaptop = "<<a.product << endl; // product
                                   logitech private data
        }
        void kyybo(logitech);
};
void dell :: kyybo(logitech m)
    {
        cout << "Keyboard = " << m.product<<endl;
    }
int main()
{
    logitech x;
    dell y;
    x.getdata();
    y.lappy(x);
    y.keybo(x);
    return 0;
}
```

Output Enter any number: 6
Laptop = 6
Keyboard = 6

Special Properties Possessed by Friend Function:-

→ A friend function is not in the scope of the class to which it has been declared as friend.

→ A friend function cannot be called using the object of that class. It can be invoked like a normal function without the use of any object.

→ Unlike member functions, it can not access the members directly. However, it can use the object and dot membership operator with each member name to access both private and public members.

→ It can be declared either in the public or the private part of a class without affecting its meaning.

→ Generally, it has got objects as arguments.

Returning Object with friend -

class Definition with function definition Syntax-

```
class class_name
{
    ----
    public:
    ----
    friend class_name function_name(class_name object)
    {
        class_name object;
        ----
        return (object)
    }
}
```

Function call-

```
Sum z, result;
result = add(z);
cout << " Addition: \n";
result.putdata();
```

Ex-

```
friend Sum add (Sum s)
{
    Sum temp;
    temp.a = s.a + s.b;
    return (temp);
}
```

→
```
# include<iostream>
using namespace std;
class Sum
{
    int a, b;
  public:
    void get_number(void);
    void putdata(void);
    friend Sum add (Sum s) // friend function
    {
     Sum temp;
     temp.a = s.a+s.b; //accessing private data from non-member
     return temp;
    }
};
void Sum :: get number (void)
{
    cout<<"Enter Any two Numbers : \n";
    cin >> a >> b;
}
void Sum :: putdata(void)
{
   cout<<"Addition : "<< a;
}
int main()
{
  Sum z, result;
  z.get_numbers();
  result = add(z);
  result.putdata();
  return 0;
}
```

Output - Enter Any two Numbers:
2
3
Addition: 5

friend Definition out of class-

Class definition Syntax-

```
class class_name
{
    ---
    public :
    ---
    friend class_name function_name(class_name);
};
```

Ex-

```
class Sum
{
    int a, b;
    ---
    public :
        void getdata();
        void putdata();
        friend Sum add(Sum);
}
```

Function Definition Syntax-

```
class_name function_name(class_name Object)
{
    class_name object; // create new object
    task;
    return(object);
}
```

Ex-

```
Sum add(Sum s)
{
    Sum temp;
    temp.a = s.a + s.b;
    return (temp);
}
```

Function call -

```
Sum z, result;
  result = add(z);
  cout << "Addition: \n";
  result.putdata();
```

→

```
#include <iostream>
using name_space std;
class Sum
{
    int a, b;
  public:
      void get_number(void);
      void putdata(void);
      friend Sum add(Sum); // friend function
};
Sum add(Sum s) // friend function definition
{
    Sum temp;
    temp.a = s.a + s.b;
    return temp;
}
void Sum:: getnumber(void)
{
    cout << "Enter Any two Numbers: \n";
    cin >> a >> b;
}
void Sum :: putdata(void)
{
    cout << "Addition: " << a;
}
```

```
int main ()
{
    Sum z, result;
     z. get_number();
     result = add (z);
     result. putdata ();
     return 0;
}
```

Output- Enter Any two Numbers:
2
3
Addition: 5

A Function Friendly to Two or more classes.
Friend function called by reference-

```
class rahul; // forward declaration
class rani
{
    int b;
  public:
    void getdata ();
    friend return_type function_name (rani &, rahul &);
    ---
};
class rahul
{
    int a;
    public:
    void getdata ();
    friend return_type function_name (rani &, rahul &);
    ---
};
```

Function definition-

```
void exchange (rani &m, rahul &n)
{
    friend body;
}
```

Function Call-

```
exchange(a,b);
```

→

```
#include <iostream>
using namespace std;
class rahul;
class rani
{
    int x;
  public:
    void getdata()
    {
        cout <<"Rani has Rs. :";
        cin >>x;
    }
    void display()
    {
        cout <<" Rani has Rs. :" << x << endl;
    }
    friend void exchange(rani &, rahul &);
};
class rahul
{
    int y;
  public:
    void getdata()
```

```
    {
        cout << "Rahul has Rs. : ";
        cin >> y;
    }
    void display ()
    {
        cout << "Rahul has Rs. : " << y << endl;
    }
    friend void exchange (rani &, rahul &);
};
void exchange (rani &m, rahul & n)
{
    int temp;
    temp = m.x;
    m.x = n.y;
    n.y = temp;
}
int main ()
{
    rani a;
    rahul b;
    a.getdata ();
    b.getdata ();
    cout << "\n Before exchange : \n";
    a.display ();
    b.display ();
    exchange (a, b);
    cout << "\n After exchange : \n";
    a.display();
    b.display ();
    return 0;
}
```

Output- Rani has Rs. : 10
Rahul has Rs. : 20
Before exchange :
Rani has Rs. : 10
Rahul has Rs : 20
After exchange :
Rani has Rs. : 20
Rahul has Rs. : 10

Constant / Litural –

→ Variable -
const int PI = 3.14;

→ Function by value-
add (const int a)

→ Function by referenca-
add (const int & a)

→ Pointer -
add(const int * a)

→
```
#include <iostream>
using namespace std;
int main ()
{
    int a=5;          or const int a=5;
    a = a+2;
    cout << "Value : " << a;
}
```

Output- Value : 7 or (Error आयेगा)

Note- const variable की value को use कर सकते हैं किन्तु change नहीं कर सकते

→
```
#include <iostream>
using namespace std;
void add (int);
int main ()
{
    int a = 5;
    add (a);
}
void add (const int x)        or void add( int x)
{
    x = x + 2;
    cout << "Value: " << x;
}
```

Output - Error आयेगा। or Value: 7

Constant Object - यह केवल constant member fun^n or data member को ही access कर सकता है।

(i) Inside class definition -

→
```
#include <iostream>
#include <conio.h>
using namespace std;
class student
{
  public:
    void disp(void) const
    {
      cout << "I am constant.";
    }
};
int main()
{
    const student obj;
    obj.display();
}
```

int a=5;

(change नहीं कर सकते)

Output- I am constant.

(ii) Out side class definition-

→
```
#include <iostream>
#include<conio.h>
using namespace std;
class student
{
    public:
        void disp(void) const;
        {
        ~~cout << "I am constant"~~
        }
};
void student :: disp(void) const
{
    cout <<"I am constant.";
}
int main()
{
    const student obj;
    obj.disp();
}
```

Output- I am constant.

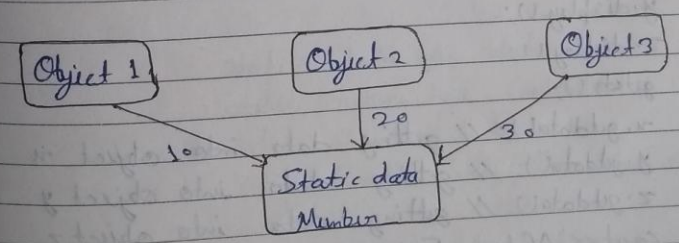
Static Data Member- इसका एक ही मेमोरी एलोकेट होता है जिसे सारे objects बारी-2 यूज करते हैं Union की तरह। यदि इसे किसी से initialize न करे तो Bydefault इसका मान 0 होता है।

Syntax-

```
class class_name
{
    - - - -
    static data_type data member;
    - - - -
};
data_type class_name :: data member ;  or data member = value;
```

Ex-

```
class Number
{
    - - -
    static int num;
    - - -
};
int Number :: num ; // definition of static data member
or int Number :: num = 10;
```



Object 1 = 30
Object 2 = 30
Object 3 = 30

→

```
#include<iostream>
#include<conio.h>
using namespace std;
class Number
{
    static int num; // num is static member
  public:
    void getdata()
      {
        cout <<"Enter Number:";
        cin >> num;
      }
    void display(void)
      {
        cout << "Number:"<< num<< endl;
      }
};
int Number :: num; // definition of static data member
int main()
{
  Number x, y, z;
  x.display();
  y.display();
  z.display();
  getch();
  x.getdata(); // getting data into object x
  y.getdata(); // getting data into object y
  z.getdata(); // getting data into object z
  cout<<"After Entering data:"<<endl;
  x.display();
  y.display();
  z.display();
```

```
    return 0;
}
```

Output- Number : 0
Number : 0
Number : 0
Enter Number: 12
Enter Number: 13
Enter Number: 15
After Entering data:
Number :15
Number:15
Number : 15

Static Member Function- यह केवल और केवल static data member or static Member Functions को ही access कर सकता है।

Syntax- (Inside class definition)-

```
class class_name
{
    - - -
    - - -
    static data_type data member;
    static return_type function_name()
    {
        function body; // only use static data member
    }
};
data_type class_name :: data member = initial_value;
```

Syntax- (outside class definition)-

```
class class_name
{
    - - - -
    static data_type data member;
    static return_type function_name( );
};
data_type class_name :: data member = initial_value;
data_type class_name :: function_name( )
{
    function body; // only use static data member
}
```

Function call-

Syntax- class name :: function_name( );

Ex- Number :: display( );

→

```
#include <iostream>
using namespace std;
class Counter
{
    static int count; // static member: count is static
           int n;
    public:
    void set_data (void) // initialize object's member
    {
        count ++;
        n = count;
    }
    void show_value_of_n (void)
{
```

```
        }  cout<<"The Value of n is = "<<n<<endl;
static void show_count(void)
{
        cout << " Count : " << count << endl;
}
};
int Counter:: count;
int main()
{
    Counter C1,C2;
    Counter:: show_count();
    C1.set_data();
    C2.set_data();
    Counter:: show_count();
    Counter C3;
    C3.set_data();
    Counter:: show_count();
    C1.show_value_of_n();
    C2.show_value_of_n();
    C3.show_value_of_n();
    return 0;
}
```

Output- Count : 0
Count : 2
Count : 3
The value of n is : 1
The value of n is : 2
The value of n is : 3

Parameter and Argument-

```
int sum ( int a , int b)
{                          → Parameters
    return (a+b);
}
int main ()
{
    int total;
    total = sum ( 10, 145);
}                          → Arguments
```

Constructor- C++ provides a special member function called constructor for initializing an object when it is created. This is known as automatic initialization of objects.

A class constructor if defined is called whenever a program creates an object of that class.

with out constructor →

```
#include <iostream>
using namespace std;
class comment
{
    public :
        void show (void)
        {
            cout<<"I am sick";
        }
};
int main ()
```

```
{
    Comment yes;
    yes.show();
}
```

Output- I am sick.

→

```
#include <iostream>
using ~~namp~~ namespace std;
class comment
{
    public:
        comment()
        {
            cout<<"I Love programing."<< endl;
        }
        void show(void)
        {
            cout<<"I am sick";
        }
};
int main()
{
    Comment yes;
                    or    yes.show();
}
```

Output- I Love programing or I Love programing
I am sick.

Characteristics of Constructors -

- They should be declared in the public section so that its objects can be created in any function.
- They are invoked directly when an object is created.
- They don't have return type, not even void and hence can't return any value.
- Constructor can't be friend function.
- They can't be inherited.
- Like other C++ functions, they can have default arguments.
- Constructors can't be virtual.
- Constructor can be inside the class definition or outside the class definition.

Limitations of Constructor -

→ **No return type** - A constructor cannot return a result, which means that we cannot signal an error, during the object initization. The only way of doing it is to throw an exception from a constructor.

→ **Naming** - A constructor should have the same name as the class, which means we cannot have two constructors that both take a singal argument.

→ **Compile time bound** - At the time when we create an object, we must specify the name of a concrete class which is known at compile time. There is no way of dynamic binding constructors at run time.

→ **There is no virtual constructor** - We cannot declare a virtual constructor. We should specify the exact type of the object at compile time, so that the compiler can allocate memory for that specific type. If we are constructing derived object, the compiler calls the base class constructor first and the derived class hasn't been initialized yet. This is the reason why we cannot call virtual methode from the constructor.

Declaration of Constructors -

(i) Syntax

```
// class with constructor, defined inside class.
class student
{
    private:
        int id;
        int roll;
        public:
        student () // constructor defined
        {
            id = 10; // constructor value initialize
            roll = 145;
        }
}
```

→

```
// constructor member function inside class
#include <iostream>
using namespace std;
class student
{
    private:
        int id;
        int roll;
    public:
        student ()
        {
            id = 10;
            roll = 145;
            cout << "ID:" << id << endl;
            cout << "Roll :" << roll << endl;
        }
};
```

```
int main()
{
    student stu;
}
```

Output- ID : 10
Roll: 145

Syntax

iii // class with constructor, defined outside class

```
class student
{
    private:
        int id;
        int roll;
        public:
            student (); // constructor declared
};
student :: student() // constructor definition outside class
    {
        id = 10;   // constructor value initialized
        roll = 145;
    }
```

→ // constructor member function outside class

```
#include <iostream>
using namespace std;
class student
{
    private:
        int id;
        int roll;
    public:
```

```
        student ();  // constructor member function declar
};
student :: student () // constructor member function definition
{
    id = 10; // constructor value initialized
    roll = 145; // constructor value initialized
    cout << "ID: " << id << endl;
    cout << "Roll: " << roll << endl;
}
int main()
{
    student stu;
    return 0;
}
```

Output- ID: 10
Roll: 145

Default Constructor- A constructor is a constructor that either has no parameters; or if it has parameters, all parameters have default values.

Syntax-

```
class student
{
    int roll_no;
    public:
        student ();
};
student :: student () // default constructor defin
{
    ------
}
student object 1;
```

→
```
#include <iostream>
using namespace std;
class student
{
    int id;
    public:
        student ()
        {
            id = 10;
            cout <<"ID : " <<id <<endl;
        }
};
int main ()
{
    student stu;
    return 0;
}
```

Output - ID = 10

Note - जब हम Generally कोई class create करते हैं जिसमें हम Constructor नहीं लिखते हैं तो Compailer हमारे लिये एक Default constructor create करता है। और उसे define करता है जिसके through हम लोग Object member (data member) को initialize कर पाते हैं।

Parameterized Constructor- The constructors that can take the arguments are called parameterized constructor

Ex-

```
class student
{
        int id, roll;
    public:
        student (int, int); // parameterized constructor
};
student :: student (int n, int r)
{
        id = n;
        roll = r;
}
```

Create object-

→ By calling the constructor explicitly-

Syntax-

```
student object1 = student (10, 145);
```

→ By calling the constructor implicitly / Shortha

Method-

Syntax-

```
student object1 (10, 145);
```

```
#include <iostream>
using namespace std;
class student
{
        public:
        student (int , int);
};
student :: student (int a, int b)
{
    cout << "Value of A:" << a << endl;
    cout << "Value of B:" << b << endl;
}
int main()
{
  student stu = student(10, 145); // explicitly call
  or student stu (10, 145); // implicit call
    return 0;
}
```

Output- Value of A: 10
Value of B: 145

Note- जब भी हम लोग कोई Object create करते हैं किसी class का तो C++ के पास एक default constructor होता है जो call होता है वो भले हि हम class मे define करे या ना करे लेकिन C++ के पास एक default constructor है जो हमेशा call होता है जब हम एक object को declare करते हैं तब लेकिन यह work नहीं करता जब हम एक Parametrized constructor का प्रयोग करते हैं। तो Compiler समझ नहीं पाता की उसके पास एक default constructor है या नही है। इसलिए जब हम कोई object को create करते हैं तो वह उस default constructor को call नही कर पाता है यानि उसे पता नहीं है या नही है। तब हमें error होता की अभी default constructor मिलेगा। उससे बचने के लिये हमे Manually एक default constructor

define करना पड़ेगा।

→ #include<iostream>

```
using namespace std;
class student
{
    public:
        student ()
        {
        }
    student (int, int);
};
student :: student (int a, int b)
{
    cout << "Value of A: " << a << endl;
    cout << "Value of B: " << b << endl;
}
int main()
{
    student stu1, stu2;
    student stu = student (10, 145);
}
```

Output- Value of A: 10
Value of B: 145

Copy Constructor- A copy constructor is used to declare and initialize an object from another object.

Syntax- outside class definition

```
class student
{
    - - - - ;
    public:
        student (student &obj) // Copy constructor
        {
            - - - - ;
        }
};
```

Call Copy Constructor-

Syntax-

```
class_name target_object (source_object);
class_name target_object = source_object;
```

Ex-

```
student target (source);
student target = source;
```

→

```
#include <iostream>
using namespace std;
class student
{
    int roll;
    int mark;
    public:
        student (int m, int n)
        {
            roll = m;
            mark = n;
        }
```

```
        student (student & t); // copy constructor
        void show_data()
        {
            cout<<"\n Roll No: "<< roll;
            cout << " \n Mark : "<< mark;
        }
};
student :: student (student & t) // copy constructor
    {
        roll = t.roll;                    or 30
        mark = t.mark;
    }

int main()
{
    cout << " \n Parametrized constructor output r\n";
    student r(60,130); // Parametrized constructor
    r.show_data(); // Parametrized constructor output
    cout << " \n Copy constructor output stu(r) \n";
    student stu(r); // copy constructor
    stu.show_data(); // copy constructor output
    return 0;
}
```

Output - Parametrized constructor output r.
Roll no: 60
Mark : 130
Copy constructor output stu(r)
Roll no : 60
Mark : 130 or 30

Constructor Overloading - When more than one constructor function is defined in a class, then it is called constructor overloading or use of multiple constructor in a class.

Syntax-
```
class student
{
        int id, roll;
    public:
        student ();
        student (int);
        student (int, int);
};
```

→
```
#include <iostream>
using namespace std;
class student
{
        int roll;
        int mark;
    public:
        student ();
        student (int);
        student (int, int);
        void show(void);
};
student :: student ()
{
    roll = 3;
    mark = 3;
}
```

```
student :: student (int a)
{
    roll = mark = a;
}
student :: student (int x, int y)
{
    roll = x;
    mark = y;
}
void student :: show (void)
{
    cout << " Roll : " << roll << endl;
    cout << "Mark : " << mark << endl;
}
int main ()
{
                    //कर सकता था ()
    student stu;  // student stu, stu1, stu2 ;
    student stu1(6); //stu1 = student (6) ;
    student stu2(3,4); // stu2 = student (3, 4) ;
    stu. show ();
    cout << "\n One Argument \n";
    stu1. show ();
    cout << "\n Two Argument \n";
    stu2. show();
    return 0;
}
```

Output - Roll : 3
Mark : 3
One Argument
Roll : 6
Mark : 6
Two Argument
Roll: 3
Mark: 4

Destructors - Destructors are functions that are complimentary to constructors. A ~~destructo~~ destructor, as the name implies, is used to destroy the objects that have been created by using constructor. A destructor is function that has the same name as that of the class but is prefixed with a ~(tilde).
यह अपने से जल करके कि constructor के द्वारा Create किये गये मेमोरी का उपयोग हो चुका है और अब इसमें कोई काम नहीं है तब यह खुद को कोल करके मेमोरी को free कर देता है और object को distroy कर देता है।

Characteristics Of Destructors -

→ Distructor functions are invoked automatically when the objects are destroyed.

→ We can have only one distructor for a class, i.e. distructors can't be overloaded.

→ No argument can be provided to a distructor, neither does it return any value.

→ Destructor can't be inherited.
→ A distructor may not be static.

Limitations Of Destructors -

→ Thy are case sensitive.
→ Thy are not good for big program having thousand lines of code.

Declaration Of Destructor -

(i) inclass -

(ii) outside of class

```
~student ()        // destructor declaration
{
    cout<<" Thanx for using this program" << endl;
}
```

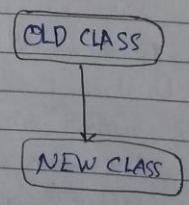
→

```
#include<iostream>
#include< conio.h>
using namespace std;
class student
{
    private:
        char name[20], add[20];
        int roll, zip;
    public:
        student ();  // Constructor
        ~student ();  // Destructor
        void read();
        void disp();
};
student :: student ()
    {
        cout <<"\nStudent Details:"<< endl;
    }
void student :: read()
    {
        cout <<"Enter Student Name :";
        cin >> name;
        Cout << "Enter Student Roll No : " <<roll<<
        ~~cout <<"Address~~ cin >> roll;
```

```
        cout << " Enter Student Address : ";
        cin >> add;
        cout << " Enter Pincode : ";
        cin >> zip;
    }
void student :: disp()
    {
        cout << endl << " Student Name : " << name << endl;
        cout << " Roll no : " << roll << endl;
        cout << " Address : " << add << endl;
        cout << " Zipcode : " << zip;
    }
student :: ~ student ()
    {
        cout << " \n\n Student Detail is closed and related
                                object destroyed";
    }
int main()
    {
        {
            student s1;
            s1.read();
            s2.disp();
        }
        getch();
        student s2;
        s2.read();
        s2.disp();
        getch();
        return 0;
    }
```

Output - Student Details:
Enter Student Name : rahul
Enter Student Roll No: 34
Enter Student Address : delhi
Enter Pincode : 656757
Student Name : rahul
Roll no: 34
Address : delhi
Zipcode : 656757
Student Detail is Closed and related object distroyed.
Student Details:
Enter Student Name: raj
Enter Student Roll No: 87
Enter Studen Address: mum
Enter Pincode : 6786
Student Name : raj
Roll no: 87
Address : mum
zipcode : 6786
Student Detail is Closed and related object destro

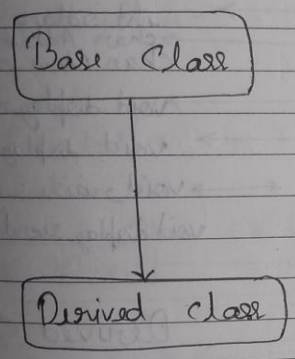
Inheritance - The mechanism of deriving a new class from an old one is called inheritance or derivation.

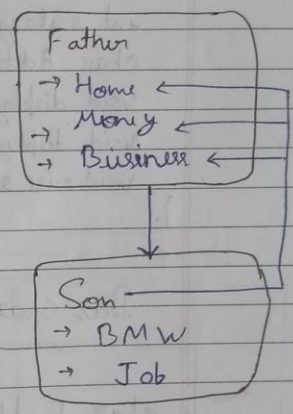


Base Class and Derived - The old class is referred to as the base class and the new one is called the derived class.

Base class - Super Class, or Parent class

Derived Class - Sub Class, Extended Class, or child class.





Why do We need inheritance -

```
class Employee
{
    public:
      int id;
      char Name[50];
      int age;
      char Department Name[50];
      int salary;
      char Address[100];
      void display_name(void);
      void display_id(void);
      void raise_salary(float percent);
};
```

```
class Manager
{
    public:
      int id;
      char Name[50];
      int age;
      char Department Name[50]
      int salary;
      char Address[100];
      char Name_of_secretary
      void display_name(void)
      void display_id(void);
      void raise_salary(float
      void display_secretary_name(vo
```

Base class

```
class Employee
{
    public:
      int id;
      char Name[50];
      int age;
      char Department Name[50];
      int salary;
      char Address[100];
      void display_name(void),
      void display_id(void);
      void raise_salary(float percent);
};
```

Derived class

```
class Manager:: public Employ
{
    public:
      char name_of_secretary
      void display_secretary_name(vo
      void raise_salary(float p
      void
};
```

Declaration of Derived Class -

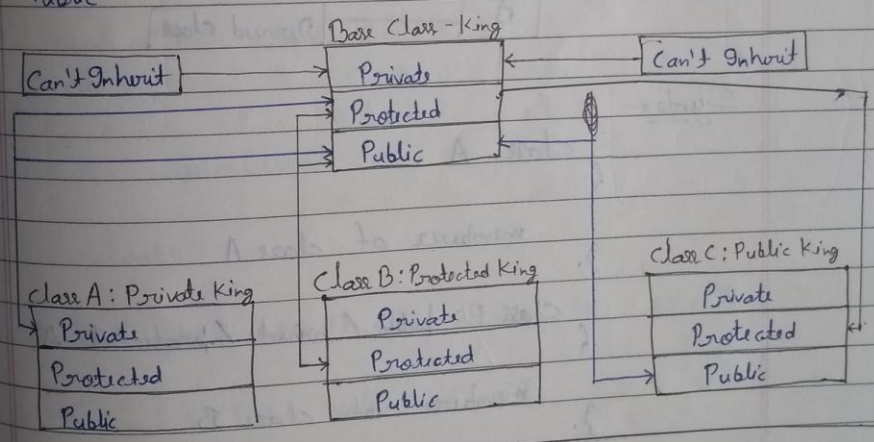
Ex- class Derived Class Name : VisibilityMode Base Class Name
{
    members of derived class
};

- Private
- Protected
- Public

Note- यदि हम Visibility Mode न लिखें तो यह By default Private हो जायेगा।

Visiblity Mode -

1- Private
2- Protected
3- Public

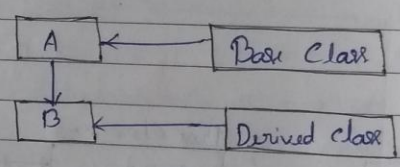


Base class Visibility	Derived Class Visibility		
	Private	Protected	Public
Private	Not Inherited	Not Inherited	Not Inherited
Protected	Private	Protected	Protected
Public	Private	Protected	Public

Type of Inheritance-

1- Single Inheritance
2- Multiple Inheritance
3- Multi-level Inheritance
4- Hierarchical Inheritance
5- Hybrid Inheritance

1- **Single Inheritance** - If a class is derived from one base class, it is called Single Inheritance.



Syntax-

```
class A
{
    members of class A
};
class B: public / private / protected A
{
    members of class B
};
```

→

```
#include<iostream>
using namespace std;
class A
{
    int m;
```

```
public:
        void  getdata_m(void);
        int  retm(void);
};
class B: public A
{
    int  n, sum;
    public:
        void  getdata_n(void);
        int  add(void);
        void  display(void);
};
void A:: getdata_m(void)
{
    cout <<"Enter value of M:";
    cin >> m;
}
int A:: retm(void)
{
    return m;
}
void B:: getdata_n(void)
{
        cout << "Enter value of N:";
        cin >> n;
}
int B:: add(void)
{
    sum = retm() + n;
    return (sum);
}
```

```
void B :: display (void)
{
    cout << " Value of M: " << retm() << endl;
    cout << " Value of N: " << n << endl;
    cout << " Sum: " << sum << endl;
}
int main()
{
    B num;
    num.getdata_m();
    num.getdata_n();
    num.add();
    num.display();
    return 0;
}
```

Output- Enter Value of M: 3
Enter Value of N: 4
Value of M: 3
Value of N: 4
Sum: 7

→

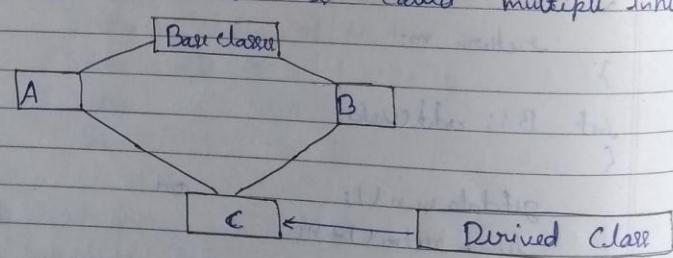
```
#include <iostream>
using namespace std;
class A
{
    int m;
  public:
    int n;
    void getdata_mn(void);
    int retm(void);
};
```

```
class B: private A
{
        int sum;
    public:
        int sum;
        void display (void);
};
void A:: getdata_m_n(void)
{
    cout << "Enter value of M:";
    cin >> m;
    cout << "Enter value of N:";
    cin >> n;
}
int A:: retm (void)
{
    return m;
}
int B:: add (void)
{
    getdata_m_n();          or
    sum = retm() + n;
    return (sum);
}
void B:: display (void)
{
    cout << " Value of M:" << retm() << endl;
    cout << " Value of N:" << n << endl;
    cout << " Sum :" << sum << endl;
}
int main()
```

```
{
    B num;
                    or num.getdata_m_n();
    num.add();
    num.display();
    return 0;
}
```

Output- Enter value of M: 3
Enter value of N: 4
Value of M: 3
Value of N: 4
Sum: 7

2- Multiple Inheritance- If class is derived from more than one parent class (base classes), then it is called multiple inheritance



Syntax-

```
class A
{
    member of class A
};
class B
{
    member of class B
};
```

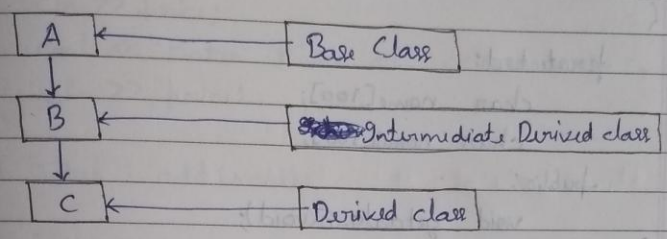
```
class C: Public / Private / Protected A, Public / Private / Protected B
{
    member of class C
};

#include <iostream>
using namespace std;
class A
{
    protected:
        int x;
    public:
        void get_x(void);
};
class B
{
    protected:
        int y;
    public:
        void get_y(void);
};
class C: public A, public B
{
    int sum;
    public:
        void add(void);
        void disp(void);
};
void A:: get_x(void)
{
    cout << "Enter value of X: ";
```

```
    cin >> x;
}
void B :: get_y (void)
{
    cout << "Enter Value of Y: ";
    cin >> y;
}
void C :: add(void)
{
    sum = x+y;
}
void C :: disp(void)
{
    cout << "Value of X:" << x << endl;
    cout << "Value of Y:" << y << endl;
    cout << " Sum :" << sum << endl;
}
int main ()
{
        c num;
        num. get_x();
        num. get_y();
        num.add();
        num. disp();
        return 0;
}
```

Output- Enter Value of X: 3
Enter Value of Y: 4
Value of X: 3
Value of Y: 4
Sum : 7

3- <u>Multilevel Inheritance</u> - In multi-level inheritance, the class inherits the feature of another derived class.



<u>Syntax</u>-

```
class A
{
    members of class A
};
class B : Public / Private / Protected A
{
    members of class B
};
Class C : Public / Private / Protected B
{
    members of class C
};
```

Base class

Intermediate Derived class

Derived class

→
```
#include <iostream>
using namespace std;
class Food
{
    protected:
        char name[100];
        char name1[100];
    public:
        void getdata_n(void);
};
class Rate : public Food
{
    protected:
        int price;
        int price1;
    public:
        void getdata_p(void);
};
class Total : public Rate
{
    int sum;
  public:
        void add (void);
        void disp (void);
};
void Food:: getdata_n (void)
{
    cout<<"Food Name : ";
    cin >> name;
    cout<< "Food Name : ";
    cin >> name1;
}
```

```
void Rate :: getdata_p(void)
{
    cout << "Enter Price of " << name << " :";
    cin >> price;
    cout << "Enter Price of " << name1 << ": ";
    cin >> price1;
}
void Total :: add(void)
{
    sum = price + price1;
}
void Total :: disp(void)
{
    cout << endl << " ********************" << endl;
    cout << " Food Name \t Price " << endl;
    cout << name << '\t' << '\t' << price << endl;
    cout << name1 << '\t' << '\t' << price1 << endl;
    cout << '\t' << '\t' << " ------ " << endl;
    cout << " Sum \t \t " << sum << endl;
}
int main()
{
    Total num;
    num.getdata_n();
    num.getdata_p();
    num.add();
    num.disp();
    return 0;
}
```

Output- Food Name: pizza
Food Name: dosa
Enter Price of pizza: 100
Enter Price of dosa: 200

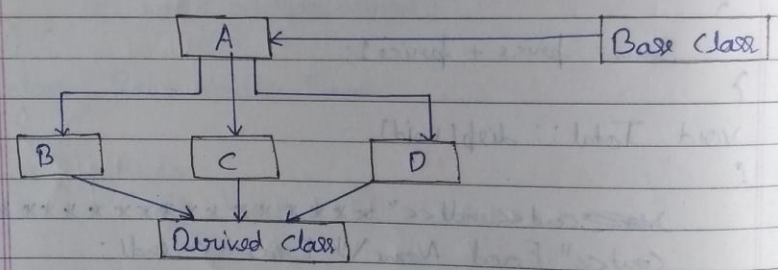
P.T.O

****

Food Name	Price
Pizza	100
dosa	200

Sum	300

4- <u>Hierarchical Inheritence</u> -



<u>Syntax</u>-

```
class A
{
    members of class A
};
class B : Public / Private / Protected A
{
    member of class B
};
class C : Public / Private / Protected A
{
    members of class C
};
```

Base class

Derived class

Derived class

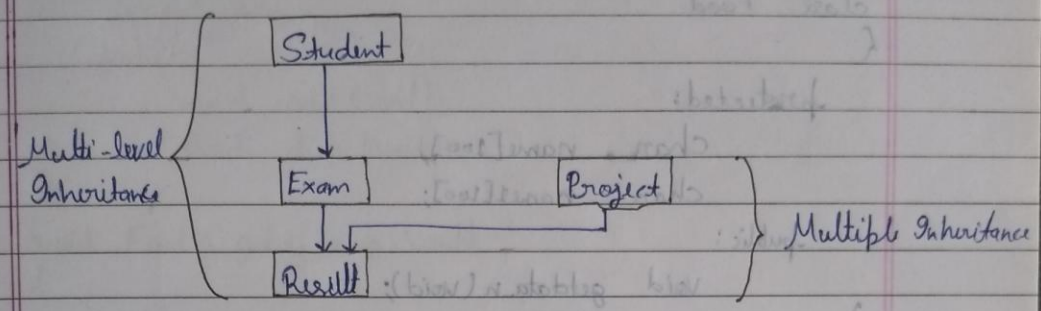
```
→ #include <iostream>
using namespace std;
class Number
{
  protected:
      int m, n;
  public:
      void getnum(void)
      {
          cout<<" Enter any Number:";
          cin >> m;
          cout <<"Enter any Number:;
          cin >> n;
      }
};
class Addition : public Number
{
  public:
      void add (void)
      {
          cout <<" * * * * * * Addition * * * * * * * * * * << endl;
          getnum();
          cout <<" Addition:" << m+n << endl << endl;
      }
};
class Subtraction: public Number
{
  public:
      void sub (void)
      {
          cout <<" * * ** ** Subtraction ** ** ** ** " << endl;
```

```
    getnum();
    cout<<" Subtraction:" <<m-n<<endl;
    }
};
int main()
{
    Addition a;
    Subtraction s;
    a.add();
    s.sub();
    return 0;
}
```

Output-

```
****** Addition ******
Enter Any Number: 3
Enter any Number: 4
Addition : 7
****** Subtraction ******
Enter any Number: 4
Enter any Number: 3
Subtraction : 1
```

5- <u>Hybrid Inheritance</u>- Combination of two or more type of inheritance to design a program.



<u>Syntax</u>-
```
class Student
{
        members of class Student
};
class Project
{
        members of class Project
};
class Exam : Private/Public/Protected Student
{
        members of class Exam
};
class Result : Public/Private/Protected Exam, Public/Private/Protected Project
{
        members of class Result
};
```

X

→
```
#include <iostream>
using namespace std;
class Food
{
    protected:
        char name[100];
        char name1[100];
    public:
        void getdata_n(void);
};
class Rate : public Food
{
    protected:
        int price;
        int price1;
    public:
        void getdata_p(void);
};
class order
{
    protected:
        int p;
        int p1;
    public:
        void getdata_each(void)
        {
            cout << " No. of Order:";
            cin >> p;
            cout << " No. of Order:";
            cin >> p1;
        }
};
```

```
class Total : public Rate, public Order
{
        int sum;
    public:
        void add (void);
        void disp (void);
};
void Food :: getdata_p(void)
{
    cout << " Enter Price of " <<name<<":";
    cin >> price ;
    cout<<"Enter Price of " <<name1<<":";
    cin >> price1;
}
void Total :: add (void)
{
    sum = (price *p) + (price1*p1);
}
Void Total :: disp (void)
{
    cout <<endl <<"*****************************************<<endl;
    Cout<<"Food Name\t No. of order\t Price Each"<<endl;
    cout<<name<<'\t'<<'\t'<<"  " <<p <<'\t'<<'\t'<<price<<endl;
    cout<<name1<<'\t'<<'\t'<<"  "<<p1<<'\t'<<'\t'<<price1<<endl;
    Cout<<"Total = " << sum << endl;
}
int main()
{
    Total sum;
    num.getdata_n();
    num.getdata_p();
```

```
    num.getdataeach();
    num.add();
    num.disp();
    return 0;
}
```

Output Food Name : Piza
Food Name : Dosa
Enter Price of Piza : 100
Enter Price of Dosa : 200
No. of Order : 2
No. of Order : 1
****

Food Name	No. of Order	Price Each
Piza	2	100
Dosa	1	200

Total = 400

Ambiguity Resolution in Inheritance -

→

```
#include <iostream>
using namespace std;
class A
{
  protected:
      int x = 10;
  public:
      void display (void)
      {
          cout << " x = " << x << endl;
      }
};
```

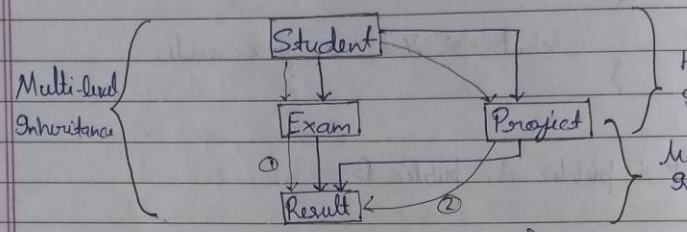
```
class B
{
    protected:
        int y = 20;
    public:
        void display (void)
        {
            cout << " y = " << y << endl;
        }
};
class P : public A, public B
{
    protected:
        int sum;
    public:
        void add (void)
        {
            sum = x+y;
            cout << " Sum = " << sum << endl;
        }
};
int main ()
{
    P num;
    num. display ();        or  num. A:: display ();
                            or  num. B:: display ();
    num. add ();
    return 0;
}
```

Output- Error आयेगा। क्योंकि एक ही नाम के दो function हैं A और B में (display ();)

or x = 10
y = 20
sum = 30

Always ahead...

Virtual Base Class- When a class is made a virtual base class, C++ takes necessary care to see that only one copy of that class is inherited, regardless of how many inheritance paths exist between the virtual base class and a derived class.



Note- Student के files की 2 copy Result के पास चला जा रहा है तो Compailer confuse हो जाता है कि कौन सा Path चुनें। इससे बचने के लिये Virtual Base class का use करते हैं।
(यह Dimond Problem को दूर करने के लिये use करते हैं।)

Declaration of Virtual Base Class-

Syntax-
```
class Student
{
    ------
};
class Exam: virtual public Student   or: public virtual Student
{
    ------
};
class Project: public virtual Student
{
    ------
};
class Result: public Exam, public Project
{
    ------ // only one copy of class student will be inherited
};
```

```
→ #include<iostream>
using namespace std;
class Student
{
  protected:
        char name[100] = "Geekyshows";
};
class Exam : public Student      or : virtual public Student
{
  protected:
        int mark = 100;
   public:
      void disp_mark(void)
      {
          cout<<" Marks: " << mark<< endl;
      }
};
class Project : public Student      or : public virtual Student
{
   protected:
        int p_mark = 200;
    public:
        void disp_pmark (void)
        {
            cout<<"Project Marks: " << p_mark<<endl;
        }
};
class Result : public Exam, public Project
{
    public:
        void disp_total (void)
```

```
        {
            cout<<"Name:" << name << endl;
            cout << "Total Marks:" << mark+pmark<<endl;
        }
};
int main()
{
    Result obj;
    obj.disp_mark();
    obj.disp_pmark();
    obj.disp_total();
}
```

Output- Error 'name' का पता compiler नहीं पा रहा है। (2 cols)

or

```
Marks: 100
Project Marks: 200
Name Crekyshow
Total Marks: 300
```

Constructor in Derived class - If any base class contains a constructor with arguments (one or more), it is necessary for the derived class to have a constructor and pass the arguments to the base class constructors.

Note- यदि Base class में कोई constructor define है तो derived class में constructor define करना आवश्यक है।

→
```
#include <iostream>
using namespace std;
class A
{
    private:
```

```
        int a;
    protected:
        int b;
    public:
        A(int i, int j)
        {
            a=i;
            b=j;
            cout<<"A initialized"<<endl;
        }
        void display_ab(void)
        {
            cout<<"\nThe value of a is:"<<a;
            cout<<"\nThe value of b is:"<<b;
        }
        int get_a(void)
        {
            return a;
};
class B
{
    private:
        int c;
    protected:
        int d;
    public:
        B(int i, int j)
        {
            c=i;
            d=j;
            cout<<"B initialized"<<endl;
        }
```

```
    void display_cd (void)
    {
        cout<<"\nThe value of c is:" <<c;
        cout<<"\nThe value of d is:"<<d;
    }
    int getc (void)
    {
        return c;
    }
};
class C: public B, publicA   or : public A , public A
{                            // जो पहले लिखा जायेगा वह पहले initialize होगा।
    int e, f, total;
public:
    C( int m, int n, int o, int p, int q, int r): A(m,n), B(o,p)
    {
        e = q;
        f = r;
        cout<<" C initialized";
    }
    void sum(void)
    {
        total = get_a() + b + get_c() + d + e + f;
    }
    void display (void)
    {
        cout <<"\nThe value of e is: "<<e;
        cout <<"\nThe value of f is:"<<f;
        cout<<"\nThe sum of a,b,c,d,e and f is:"<<total;
    }
};
```

```
int main()
{
    C objc(10,20,30,40,50,60);
    objc.display_ab();
    objc.display_cd();
    objc.sum();
    objc.display();
    return 0;
}
```

Output- B initialized or A initialized
A initialized or B initialized
C initialized
The value of a is : 10
The value of b is : 20
The value of c is : 30
The value of d is : 40
The value of e is : 50
The value of f is : 60
The sum of a, b, c, d, e and f is : 210

Order of Execution of Constructors:-

Method of Inheritance	Order of Execution

• class B: public A
{
};
 	
class C: public B, public A
 	
• class C: public B, virtual A
 	
• class B: public A
{
};
class C: public B
{
};
	A(): base constructor
	B(): derived constructor
	
	B(): base constructor
	A(): base constructor
	C(): derived constructor
	
	A(): virtual base constructor
	B(): base constructor
	C(): derived constructor
	
	A(): super base constructor
	B(): base constructor
	C(): derived constructor

Note- Virtual → Base → Derived

Constructor in Derived Class with Multilevel Inheritance -

→
```
#include<iostream>
using namespace std;
class A
{
    protected:
        int a;
    public :
        A(int i)
        {
            a=i;
            cout<<"\n A initialized";
        }
};
class B: public A
{
    protected:
        int b;
    public:
        B(int x, int y) : A(x)
        {
            b=y;
            cout <<"\nB initiallized";
        }
};
class C: public B
{
    protected:
        int c;
    public:
```

```
    C(int m, int n, int o): B(m,n)
    {
        c=o;
        cout <<"\nC initialized";
    }
    void disp(void)
    {
     cout <<"\nValue of A:" <<a<<endl;
     cout <<"Value of B:"<<b <<endl;
     cout <<"Value of C: <<c<<endl;
    }
};
int main()
{
    C objc(10,20,30);
    objc.disp();
    return 0;
}
```

Output -
A initialized
B initialized
C initialized
Value of A: 10
Value of B: 20
Value of C: 30

Initiallization List in Constructor -

→
```
#include<iostream>
using namespace std;
class A
{
  protected:
      int a;
  public:
      A (int k)
      {
          a=k;
          cout<<" A initialized"<< endl;
      }
      void display_ab(void)
      {
          cout<<"\n The value of a is:"<<a;
      }
};
class B
{
  protected:
      int c, d;
  public:
      B(int i, int j) : c(i), d(j)        //Initialization List.
      {                                   or c(i*2), d(j*c) or c(d*i) d(j)
          cout <<" B initialized" << endl;
      }
      void display_cd (void)
      {
          cout<<"\n The value of c is:"<<c;
```

```
        cout<<"\n The value of d is:"<<d;
    }
};
class C : public B, public A
{
    int e,f;
  public:
      C(int n, int o, int p, int q, int r): A(n), B(o,p), e(q)
      {
        f=r;
        cout<<" C initialized";
      }
      void display(void)
      {
        cout<<"\n The value of e is:"<<e;
        cout<<"\n The value of f is:"<<f;
      }
};
int main()
{
    C objc(10,20,30,40,50);
    objc.display_ab();
    objc.display_cd();
    objc.display();
    return 0;
}
```

Output-
B initialized
A initialized
C initialized
The value of a is :10
The value of c is:20 or 40 or 1347879076
The value of d is:30 or 120 or 30
The value of e is:40
The value of f is:50

Destructor in Derived Class – Destructor are invoked in the reverse order of the constructor invocation.

→
```
#include<iostream>
using namespace std;
class A
{
    protected:
        int a, b;
    public:
        A(int i, int j)
        {
            a=i;
            b=j;
            cout << " A initialized" << endl;
        }
    ~A()
    {
        cout << "\n Destructor in base class A" << endl;
    }
};
class B
{
    protected:
        int c, d;
    public:
        B(int i, int j)
        {
            c=i;
            d=j;
```

```
        cout << "\n B initialized" << endl;
    }
    ~B()
    {
        cout << "\nDestructor in base class B" << endl;
    }
};
class C: public B, public A    or: public B, virtual public A
{
    int e,f;
    public:
    C(int m, int n, int o, int p, int q, int r): A(m
    {
        e = q;
        f = r;
        cout << " C initialized;
    }
    ~C()
    {
        cout << "\nDestructor in derived class C" << endl;
    }
void display(void)
    {
        cout << "\nThe value of a is:" << a;
        cout << "\nThe value of b is:" << b;
        cout << "\nThe value of c is:" << c;
        cout << "\nThe value of d is:" << d;
        cout << "\nThe value of e is:" << e;
        cout << "\nThe value of f is:" << f;
    }
};
```

```
int main()
{
    C objc (10, 20, 30, 40, 50, 60);
    objc.display();
    return 0;
}
```

Output- B initialized or A initialized
A initialized or B initialized
C initialized
The value of a is : 10
The value of b is : 20
The value of c is : 30
The value of d is : 40
The value of e is : 50
The value of f is : 60
Destructor in derived class C
Destructor in base class A or B
Destructor in base class B or A

Q. Function Overloading – Function overloading refers to using the same thing for different purposes. This process of using two or more different purposes. This process of using two or more functions with same name but differing in the signature is called function overloading.

1- In terms of type of arguments
2- In terms of number of arguments

1- In terms of type of arguments

Ex-
- int add (int a, int b)
- double add (double a, double b)
- int add (int a, float b)

→
```
#include <iostream>
using namespace std;
int value(int);
double value(double);
char value(char);
double value(int, double);
int value(int, int);
int main()
{
    cout<<"\n Integer : "<<value(10);
    cout<<"\nDouble : "<<value(12.34);
    cout<<"\nChar : <<value('a');
    cout<<"\n Integer+ Double :"<<value(8, 12.2);
    cout<<"\n Integer + Integer :"<<value(2,3);
    return 0;
}
int value(int x)
{
    return x;
}
double value(double y)
{
    return y;
}
char value(char z)
{
```

or

or

2-

→

```
        return z;
}
double value (int m, double n)
{
    return (m+n);
}
int value (int a, int b)
{
    return (a+b);
}
```

Output- Integer: 10
Double : 12.34
Char : a
Integer + Double : 20.2
Integer + Integer : 5

Note- Compailer fun^n argument से datatype को identify करता है।

2- In terms of type of arguments-

Ex- • int add(int a, int b)
• int add(int a, int b, int c)

→
```
#include<iostream>
using namespace std;
int value (int);
int value (int, int);
int value (int, int, int);
int main()
{
    cout <<"\n Integer:" << value (10);
    cout <<"\n Integer+ Integer :" << value (20,30);
    cout <<"\n Int + Int + Int : "<<value(40,50,60);
```

```
    return 0;
}
int value (int m)
{
    return m;
}
int value (int a, int b)
{
    return (a+b);
}
int value (int x, int y, int z)
{
    return (x+y+z);
}
```

Output- Integer: 10
Integer + Integer: 50
Int + Int + Int : 150

Function Overloading Exact Match - When an overloaded function is called, C++ goes through the following process to determine which version of the function will be called:

1- C++ tries to find an exact match. This is the case where the actual argument exactly matches the parameter type of one of the overloaded functions.

→
```
#include<iostream>
using namespace std;
void value(int);
void value(char);
int main()
{
    value(10);          or value('a');
    return 0;
}
void value(int x)
{
    cout<<"Integer:"<<x;
}
void value(char m)
{
    cout<<"Char:"<<m;
}
```

Output- Integer: 10 or Char: a

2- If no exact match is found, C++ tries to find a match through promotion.

→
```
#include<iostream>
using namespace std;
void value(char*);
void value(int);
int main()
{
    char st='a';
    value(st);
    return 0;
```

```
}
void value (int m)
{
    cout << " Integer: " << m;
}
void value (char* x)
{
    cout << " Char: " << * x;
}
```

Output - Integer: 97 (ASCII Value of 'a')

3- If no promotion is found, C++ tries to find a match through standard conversion.

- Any numeric type will match any other numeric type, including unsigned (e.g. int to float)
- Enum will match the formal type of a numeric type (e.g. enum to float)
- Zero will match a pointer type and numeric type (e.g. 0 to char*, or 0 to float)
- A pointer will match a void pointer.

Function overloading Ambiguous -

→
```
#include <iostream>
using namespace std;
void value (unsigned int);
void value (float);
int main()
{
    value(32);                    or value (32.6f);
    return 0;
}
void value (float m)
{
    cout << " float: " << m;
}
void value (unsigned int x)
{
    cout << " Unsigned int : " << x;
}
```

Output - (Error आयेगा) or float: 32.6

Note - value (32.6 f) में f ~~यह~~ Canpailer को यह बताता है कि यह एक float value है। यदि हम f न लिखें तो यह by default double value माना जायेगा।

Decimal Number as double or float -

→
```
#include <iostream>
using namespace std;
void value (float);
void value (double);
int main()
{
    value (12.5);          or Value (12.5 f);
    return 0;
}
~~value~~ void value (float x)
{
        cout << "Float : " << x;
}
void value (double y)
{
    cout << " Double : " << y;
}
```

Output- Double : 12.5 or ~~value~~ Float : 12.5

Note- एफ f को uppercase या lowercase दोनों में लिख सकते हैं।

Operator Overloading -

1- Additional meaning of an operator
2- ~~Not~~ Normal function of operator still exists

एक operator को जो वह काम करता है उसके अलावा और भी काम करना।

List of Operators that can be Overloaded -

+	-	*	/	%	^	&	|	~	!
=	<	>	+=	-=	*=	/=	%=	^=	&=
|=	<<	>>	<<=	>>=	==	!=	<=	>=	&&
||	++	--	,	->*	->	( )	[ ]	new	delete
new[ ]	delete[ ]								

List of Operators that can't be overload -

.	.*	::	sizeof	?:

(member selection through pointer to function)

General rules for Operator Overloading -

- You cannot define new operators, such as **.
- You cannot redefine the meaning of operators when applied to built-in data types.
- Overloaded operators must either be a non static class member function or a globle function.
- Obey the precedence, grouping, and number of operands dictated by their typical use with built-in types. Therefore, there is no way to express the concept "add 2 and 3 to an object of type Point", expecting 2 to be added to the x coordinate and 3 to be added to the y coordinate.
- Unary operators declared as member functions take no arguments; if declared as global functions they take one argument.

- Binary operators declared as member functions take one argument; if declared as global functions, they take two arguments.
- If an operator can be used as either a unary operator (&, *, +, and -), you can ~~and~~ overload each use separately.
- Overloaded operators cannot have default arguments.
- All overloaded operators ~~exp~~ except ~~assg~~ assignment operator (=) are inherited by derived classes.
- The first argument for member-function overloadi operators is always of the

- Overloading + doesn't overload +=, and similarly for the other extended assignment operators.
- You may not redefine ::, sizeof, ?:, or . (dot).
- =, [], and -> must be member functions if they are overloaded.
- ++ and -- need special treatment because they are prefix and postfix operators.
- There are special issues with overloading assignment (=). Assignment should always be overloaded if an object dynamically allocates memory.

Defining Operator Overloading -

1- Outside class-

Syntax Return_type operator op(); // function prototype

```
Return_type class_name :: operator op (argument)
{
        Body of function
}
```

2- Inside class-

Syntax Return_type operator op (argument)

```
{
        Body of function
}
```

→ For friend function-

1- Outside class-

Syntax- friend Return type operator op(class_name & object);

```
Return_type operator op(class_name & object)
{
        Body of function
}
```

2- Inside class-

Syntax friend Return type operator op (class_name & object)

```
{
        Body of function
}
```

List of Operators that can't be overloaded using friend function-

=	()	[ ]	->

- A member function has no arguments for unary operators and only one argument for binary operators.
- A friend function will have only one argument for unary operators and two for binary operators.

Calling Operator Function-

1- For unary operator - op m or m op

2- For binary operator - m op n

(where op - any operator, and m, n objects)

→ // unary operator overloading.

```
#include <iostream>
using namespace std;
class complex
{
    int real, imaginary;
  public:
      complex (int a, int b)
      {
        real = a;
        imaginary = b;
      }
      void operator -(); // unary operator overloading
      void display ()
      {
        cout << "Real value : " << real << endl;
```

```
        cout << " Imaginary value: " << imaginary << endl;
    }
};
void operator ~~(complex &o)~~ - (); // unary operator overloading
{
    real = - real;
    imaginary = - imaginary;
}
int main ()
{
    complex c1 (10,20);        or  c1(10,-20)
    cout << " Real and Imaginary value before operation: " << endl;
    c1.display();
    -c1;
    cout << " Real and Imaginary value after operation: " << endl;
    c1.display();
    return 0;
}
```

Output- Real and Imaginary value before operation:
Real value: 10
Imaginary value: 20 or -20
Real and Imaginary value after operation:
Real value: -10
Imaginary value: -20 or 20

```
// Unary Operator overloading using friend function
#include<iostream>
using namespace std;
class complex
{
    int real, imaginary;
  public:
      complex()
      {
      }
      complex(int a, int b)
      {
          real = a;
          imaginary = b;
      }
      friend void operator-(complex&c); //unary operator
      void display()
      {
          cout<<"Real value:" << real<< endl;
          cout <<" Imaginary value:" << imaginary << endl;
      }
};
void operator - (complex&c)
{
      c.real = -c.real;
      c.imaginary = -c.imaginary;
}
int main()
{
      complex c1 (10,20);      or (10,-20)
      cout <<" Real and Imaginary value before operation:" <<endl
```

```
    c1.display();
    -c1;
    cout<<"Real and Imaginary after operation:"<<endl;
    c1.display();
    return 0;
}
```

Output - Real and Imaginary value before operation:
Real value: 10
Imaginary value: 20 or -20
Real and Imaginary value after operation:
Real value: -10
Imaginary value: -20 or 20

Note - यदि by reference न करें तो ये changes नहीं हो पायेगा।

→ // Binary operator overloading

```
#include<iostream>
using namespace std;
class complex
{
    int real, imaginary;
    public:
        complex()
        {
        }
        complex(int a, int b)
        {
            real=a;
            imagnary=b;
        }
        void operator+(complex c);
};
```

```
void complex :: operator + ( complex c)
{
    complex temp;
    temp.real = real + c.real;
    temp.imaginary = imaginary + c.imaginary;
    cout << " Real sum is : " << temp.real << endl;
    cout << " Imaginary sum is :" << temp.imaginary << endl;
}
int main()
{
    complex c1 (10,20);
    complex c2 (20,30);
    c1 + c2;
    return 0;
}
```

Output- Real sum is: 30
Imaginary sum is : 50

→ // Binary Operator Overloading using friend function

```
# include < iostream >
using namespace std;
class complex
{
    int real , imaginary;
    public :
    complex ()
    {
    }
    complex (int a, int b)
    {
        real = a;
```

```
        imaginary = b;
    }
    friend void operator + (complex c, complex d);
};
void operator + (complex c, complex d)
{
    complex temp;
    temp.real = c.real + d.real;
    temp.imaginary = c.imaginary + d.imaginary;
    cout << "Real sum is : " << temp.real << endl;
    cout << "Imaginary sum is :" << temp.imaginary << endl;
}
int main()
{
    complex c1(10,20);
    complex d1(20,30);
    c1 + d1;
    return 0;
}
```

Output- Real sum is : 30
Imaginary sum is : 50

Note - इसे By value पास न करके By reference भी Pass कर सकते है।

More about Pointer-

1- Pointer to Pointer-

→
```
#include <iostream>
using namespace std;
int main()
{
    int a=100;
    int *p;
    int **p1;
    p=&a;
    p1=&p;
    cout<<"\n Value of a:"<<a;
    cout<<"\n Value of a using pointer p :"<<*p;
    cout<<"\n Value of a using pointer p1:"<<**p1;
    cout<<"\n Address of a :"<<&a;
    cout<<"\n Address of a using pointer p:"<<p;
    cout<<"\n Address of a using point *p1 : "<<*p1;
    cout<<"\n Address of pointer p :"<<&p;
    cout<<"\n Add. of point. p using point. p1:"<<p1;
    cout<<"\n Add. of point.**p1 using point. p1:"<<&p1;
    return 0;
}
```

Output- Value of a : 100
Value of a using Pointer p : 100
Value of a using Pointer p1 : 100
Address of a : 0x28fefc
Address of a using pointer p : 0x28fefc
Address of a using pointer *p1 : 0x28fefc
Address of pointer p : 0x28fef8
Add. of point. p using point. p1 : 0x28fef8
Add. of point. **p1 using point. p1 : 0x28fef4

2- Increment / Decrement in pointer and pointer at array-

→
```
#include <iostream>
using namespace std;
int main ()
{
    int a[] = {5,16,67,88};
    int *p;
    p = & a[0];   // p = a;
    cout <<" Value:" << *p;
    p++;   // increment to next address
    cout <<" \nValue : " <<*p;
    p--;   // decrement to previous address
    cout <<" \n Value:" << *p;
    return 0;
}
```

Output- Value : 5
Value : 16
Value : 5

3- Array of pointer -

→
```
#include <iostream>
using namespace std;
int main()
{
    int a[] = {10,30,50,70};
    int *p[3];
    int i;
    cout <<"Output via Pointer:" << endl;
    cout <<" Address \t Value";
    for( i=0; i<4; i++)
    {
        p[i] = & a[i];
        cout<<'\n' << p[i];
        cout <<"      " <<* p[i];
    }
    return 0;
}
```

Output- Output via Pointer:

Address	Value
0x28 fec	10
0x28 f1f0	30
0x28 f1f4	50
0x28 fef8	70

Pointer to Data Members -

- **Syntax for Declaration -**

datatype class_name :: *pointer_name;

Ex- int Laptop :: *pRam;

- **Syntax for Assignment -**

pointer_name = & class_name :: datamember name;

Ex- pRam = &Laptop :: Ram;

Declaration + Assignment Syntax -

datatype class_name :: *pointer_name = &class_name :: datamember_name;

Ex- int Laptop :: *pRam = & Laptop :: Ram;

→
```
#include <iostream>
using namespace std;
class Laptop;
{
    public:
        int Ram;
};
int main()
{
    int Laptop : *pRam;
    pRam = & Laptop :: Ram ; // int Laptop :: *pRam = &Laptop :: Ram;

    Laptop c1;
    c1 . Ram = 1; // direct access
    cout << " Ram is " << c1 . Ram << endl;
    c1 . *pRam = 2; // access via pointer to member
```

```
    cout<<" Ram is "<<c1.Ram<<endl;
    return 0;
}
```

Output- Ram is 1
Ram is 2

Pointers to Member Function-

- Declaration-

Syntax- return_type(class_name::*ptr_name)(argument_type) = & class_name::function_name;

Ex- int(Laptop::*x)(int) = &Laptop::Ram;

- Function Call-

Syntax- (object.*object.ptr_name)(arguments);

Ex- (obj.*obj.x)(10);

- You can change the value and behavior of these pointers on runtime. That means, you can point it to other member function or member variable.

- To have pointer to data member and member functions you need to make them public.

→
```
# include <iostream>
using namespace std;
class Laptop
{
    int Ram (int m)
    {
        return m;
    }
public:
    int (Laptop :: * x) (int) = & Laptop :: Ram;
};
int main()
{
    Laptop a;
    cout <<(a.*a.x) (5);
    return 0;
}
```

Output - 5

Pointer to Object -

Syntax - class name object;
class name *pointer name;

• Initialization - pointer_name = &object;

Ex -

```
Laptop dell;        // Create object
Laptop *pdell;      // creating point object
pdell = & dell;     // initializing pointer object.
```

Accessing member using pointer object -

- Sytax - { pointer_name -> function_name ();
  pointer_name -> member_name ;
  or { (*pointer_name).function_name ();
  (*pointer_name).member_name ;

Ex -

```
{ pdell -> disp();        // member function
  pdell -> usb;           // data member
or { (*pdell).disp();     // member function
  (*pdell).usb;           // data member
```

→

```
#include <iostream>
using namespace std;
class Laptop
{
    int Ram;
  public:
      int usb;
      void getdata(int a)
      {
          Ram = a;
      }
      void disp()
      {
          cout << "RAM: " << Ram << endl;
          cout << "USB:" << usb << endl;
      }
};
int main()
{
```

```
Laptop dell;
Laptop *pdell;
pdell = &dell;
pdell -> usb = 3;            // (*pdell).usb = 3;
pdell -> getdata(20);        // (*pdell).getdata(20);
pdell -> disp();             // (*pdell).disp();
return 0;
}
```

Output- RAM: 20
USB : 3

⊛ this pointer - (this is keyward in c++)

It holds current object address.

Ex-
```
int main()
{
    cname obj;
    obj.add();   // this holds address of obj
}
```

Access using this-

```
m=10;
or this -> m = 10;
or (*this).m = 10;
```

→
```
#include <iostream>
using namespace std;
class mobile
{
    int m = 10;        or int m;
  public:
        void disp()
        {                or m = 10; or this->m = 10; or (*this).m = 10
            cout << " m = " << m << endl;
            cout << " this -> m = " << this->m << endl;
            cout << " (*this).m = " << (*this).m << endl;
        }
};
int main()
{
    mobile obj;
    obj.disp();
    return 0;
}
```

Output - m = 10
this -> m = 10
(*this).m = 10

→ // where use this pointer-

```
#include <iostream>
using namespace std;
class mobile
{
    int m;
  public:
      void getdata (int m)
      {
          this ->m = m;
      }
      void disp ()
      {
          cout <<" m = " <<m<< endl;
          cout <<" this->m =" << this->m << endl;
          cout <<" (*this).m = " <<(*this).m << endl;
      }
};
int main()
{
    mobile obj;
    obj.getdata (20);
    obj.disp();
    return 0;
}
```

Output - m = 20
this ->m = 20
(*this).m = 20

or garbagevalue आयेगा Compailor यह नही समझ पा रहा है कि value किस m में डालूं।

m = 1980567906
this->m = 1980567906
(*this).m = 1980567906

Returning object using this pointer -

```
#include<iostream>
using namespace std;
class exam.
{
    int m;
  public:
    exam max(exam ob, exam ob1)
    {
        ob.m = 30;
        ob1.m = 20;
        if (ob1.m > ob.m)
        {
            return obj1;
        }
        else
        {
            return ob;
        }
    }
    void disp()
    {
        cout << m;
    }
};
int main()
{
    exam obj, obj1, obj2;
    obj2 = obj.max(obj, obj1);
    obj2.disp();
```

}

Output - 30

Returning current Object using this -

Syntax - return * this;

```
int main()
{
    cname obj, obj1;
    obj1 = obj.add ();   // this hold address of this
}
```

→

```
#include <iostream>
using namespace std;
~~int main()~~ class exam
{
    int m;
    public:
        exam & maxi(exam &ob, exam &ob1)
        {
            ob.m = 30;
            ob1.m = 20;
            if (ob1.m > ob.m>
            {
                return ob1;
            }
            else
            {
                return *this
            }
        }
};
```

```
int main ()
{
    exam obj, obj1, obj2;
    obj2 = obj.max (obj, obj1);   or obj2 = obj1.max(obj, obj1);
    obj2.disp();
}
```

Output - 30 or 20

<u>Pointers in Derived class</u>-

Ex-
```
class A
{
    int a;
    void getdata();
}
class B: public A
{
    void fun();
}
int main()
{
    A obj;
    A *ptr;
    ptr = & obj;
    ptr -> getdata();  or (*Ptr.getdata();
    B obj1;
    ptr = & obj1;
    ptr -> fun();
}
```

```
{
    A obj;
    A *ptr;
    ptr = & obj;
    ptr -> getdata();
    B obj1;
    ((B*) ptr) -> fun();
or  B *ptr1;
    ptr1 = & obj1;
    ptr -> fun();
}
```

```
#include <iostream>
using namespace std;
class A
{
    int m;
  public:
    int n=20;
    void getdata (int x)
    {
        m=x;
    }
    void show ()
    {
        cout << "Base class Output: \n";
        cout << "Value of m=" << m << endl;
        cout << "Value of n=" << n << endl;
    }
};
class B: public A
{
    int p;
  public:
    void getdata1( int y)
    {
        p=y;
    }
    void disp ()
    {
        cout << " Derived class output: \n";
        cout << "Value of p=" << p << endl;
    }
};
```

(i)

```
int main ()
{
    A obj;
    A * ptr;
    ptr = & obj;
    ptr -> getdata(10);
    ptr -> show();
    B obj1;
    ptr = & obj1;
    ptr -> getdata1(20);
    ptr -> disp();
}
```

Output- Base class Output: or (Error आयेगा)
Value of m = 10
Value of n = 20

(ii)

```
int main ()
{
    A obj;
    A * ptr;
    ptr = & obj;
    ptr -> getdata(10);
    ptr -> show();
    B obj 1;
    B * ptr1;
    ptr1 -> getdata(40);
    ptr -> disp();
}.
```

Output- Base class Output:
Value of m = 10
Value of n = 20
Derived class output:
Value of p = 40

→
```
#include<iostream>
using namespace std;
class A
{
    int m;
  public :
    int n=20;
    void getdata (int x)
    {
      m=x ;
    }
    void disp ()
    {
        cout <<" Base class output: \n";
        Cout <<" Value of m = " <<m<<endl;
        cout << "Value of n = " <<n <<endl;
    }
};
class : public A
{
    int p;
  public:
      void getdata1 ( int y)
      {
        p = y;
      }
      void disp ()
      {
        cout<<" Derived class output: \n";
        cout << " Value of p = " << p << endl;
      }
};
```

```
int main()
{
    A obj;
    A *ptr;
    ptr = &obj;
    ptr = &obj;
    ptr -> getdata(10);
    ptr -> disp();
```

or

```
    B obj1;
    B *ptr1;
    ptr1 = &obj1;
    ptr -> get data(40);
    ptr -> disp();
}
```

Output - Base class output:
Value of m = 10
Value of n = 20

or Derived class output:
Value of p = 40

Note - जब हमारे program में एक ही नाम दो function होते हैं और कि अलग - अलग class में हैं तो pointer जिस class का Create किया जाता है उसी class के function को call करेगा।

Polymorphism - Polymorphism is the ability to use an operator or method in different ways. Polymorphism gives different meanings or functions to the operators or methods.

Type of Polymorphism -

Polymorphism
- Compile-time Polymorphism
  - Function Overlading
  - Operator Overloading
- Runtime Polymorphism
  - Virtual Function

- **Compile Time Polymorphism / Static Binding / Static linking**

Compile Time पर ही Program को ही पता चलेगा की कौन सा Function or operator किस Task perform करेगा।

1. Function Overloading
2. Operator Overloading

- **Run Time Polymorphism / Dynamic Binding -**

Run time पर Program को पता चलेगा की कौन सा Function कब करेगा।

1. Virtual Function

Virtual Function- If there are member functions with same name in base class and derived class, virtual functions give programmer capability to call member function of different class by a same function call depending upon different context. It supports runtime polymorphism.

Definition of Virtual functions -

Syntax-
```
class class_name
{
    public:
        virtual return_type function_name(arguments)
        {
            .....
        }
};
class A
{
    public:
        virtual void disp()
        {
            cout << "I am Base Class";
        }
}
```

Note- Generally virtual function को Base class में लिखा जाता है।

Rule for Virtual Function -

- The virtual function must be the members of some class.
- Virtual Functions enable derived (sub) class to provide its own implementation for the function already defined in its base (super) class.
- Virtual Functions give power to the derived class functions to ~~overloaded~~ override the function in its base class with the same name and signature.

- Virtual Functions can't be static members.
- Only the functions that are members of same class can be declared as virtual that means we can't declare regular functions or friend functions as virtual.
- A virtual function can be a friend of another class.
- ~~A virtual function in a base class object then the function of the base class will be called for sure.~~
- If one will call the virtual function with the pointer having the reference to the base class object then the function of the base class will be called for
- A Virtual function in a base class must be defined, even ~~throw~~ though it may not be used.
- The corresponding function in the derived class must agree with the virtual function's name and signature that means both must have same name and signature.

Limitations of Virtual Function -

- The function call takes slightly longer due to the virtual mechanism, and it also makes it more difficult for the compiler to optimize because it doesn't know exactly which function is going to be called at compile time.
- In a complex system, virtual functions can make it a little more difficult to figure out

where a function is being called from.
- Virtual functions will usually not be inlined.
- Size of object increases due to virtual pointer.

→ // Need of virtual function

```
#include <iostream>
using namespace std;
class A
{
    public:
virtual or void disp () // Parent class function
        {
            cout<<"\n I am from Base Class A";
        }
};
class B: public A
{
    public:
        void disp() //Child class
        {
            cout <<"\n I am from Derived Class B";
        }
};
class C: public A
{
    public:
        void disp() // Child Class Function
        {
            cout <<"\n I am from Derived Class C
        }
};
```

खाली or

```
int main ()
{
    A *pa;          // Base class Pointer
    B objb;         // B class object
    C objc;         // C class object
    pa = & objb; // address of derived class B
    pa->disp(); // calling disp of class B
    pa = & objc; // address of derived class C
    pa -> disp(); // calling disp of class C
    return 0;
```

Output- I am from Class A
I am from Class A
or I am from Derived Class B
I am from Derived Class C
or/ Error आयेगा virtual function को define करना आवश्यक है चाहे ही उसका इस्तेमाल न हो।

Pure Virtual Functions and Abstract Base Class-
Pure virtual Functions are the specific type of virtual functions. A virtual function with no function body is called pure virtual function. ie. A pure virtual function purely exists in base class only to be overridden by the derived class functions. A base class only specifies that there is a function with this name and signature which will be implemented by any of derived class or some inheritance hierarchy. Such classes are also called Abstract base class.

Syntax of Pure Virtual Function

```
class class_name
{
    public:
        virtual return_type function_name(argument list);
};
class A
{
    public:
        virtual void disp () = 0;
};
```

Characteristics of Pure Virtual Functions –

- Pure virtual function itself do nothing but acts as a prototype in the base class and gives the responsibility to a derived class to define this function.
- As pure virtual functions are not defined in the base class thus a base class can not have its direct instances or objects that means a class with pure virtual function acts as an abstract class that can not be instantiated but its concrete derived classes can be.
- → We cannot have objects of the class having pure virtual function but we can have pointers to it that can in turn hold the reference of its concrete derived classes.
- It is possible, however, to ~~object~~ provide an implementations to a pure virtual function. The functions can then be called by objects

derived from the ADT, perhaps to provide common functionality to all the overridden functions.

- Pure virtual functions also implements run time polymorphism as the ~~appropriate objects~~ ~~here~~ normal virtual functions do as binding of functions to the appropriate objects here is also delayed up to the run time, that means which function is to invoke is decided at the run time.
- Pure virtual functions are meant to be overriden. Only the functions that are members of some class can be declared as pure virtual that means we cannot declare regular functions or friend functions ~~as~~ friend function as pure virtual.
- The corresponding functions in the derived class must agree be compatible with the pure virtual function's name and signature that means both must have same name and signature.
- For abstract class, pure virtual function is must.
- The pure virtual functions in an abstract base class are never implemented. Because no objects of that type are ever created, there is no reason to provide implementations, and the ADT (Abstract Data Type) works purely as the definition of an interface to objects which derive from it.

Abstract Classes - An abstract class is one that is not used to create any object of its own but it solely exists to act as a base class for the other classes that means the abstract class must be a part of some inheritance hierarchy.

- An abstract class can not be instantiated that means abstract classes can not have their own instances provided the child class itself is not an abstract class.
- Through objects of an abstract class cannot be created, however, one can use pointers and references to abstract class types.
- A class should contain at least one pure virtual function to be called as abstract.
- If a class is made abstract by giving a pure virtual function then it must be inherited by a child class of it that provides the implementation to the pure virtual function.
- If a class inherits an abstract class and does not provide the implementation of the pure virtual function then the child class itself should declare the function as pure virtual that means the child class will be an abstract class as well.
- The signature of the function declared as pure virtual in base class must strictly agree with the signature of the function in

Abstract Classes - An abstract class is one that is not used to create any object of its own but it solely exists to act as a base class for the other classes that means the abstract class must be a part of some inheritance hierarchy.

- An abstract class can not be instantiated that means abstract classes can not have their own instances provided the child class itself is not an abstract class.
- Through objects of an abstract class cannot be created, however, one can use pointers and references to abstract class types.
- A class should contain at least one pure virtual function to be called as abstract.
- If a class is made abstract by giving a pure virtual function then it must be inherited by a child class of it that provides the implementation to the pure virtual function.
- If a class inherits an abstract class and does not provide the implementation of the pure virtual function then the child class itself should declare the function as pure virtual that means the child class will be an abstract class as well.
- The signature of the function declared as pure virtual in base class must strictly agree with the signature of the function in

child class that implements the pure virtual function.

→
```
#include <iostream>
using namespace std;
class A
{
    public:
        virtual void disp () = 0; // Pure Virtual Function
};
class B : public A
{
    public:
        void disp () // Child Class Function
        {
            cout <<"\n I am from Derived class B";
        }
};
class C : public A
{
        void disp () // Child Class Function
        {
            cout <<"\n I am from Derived Class C";
        }
};
int main ()
{
        A obja;
        B objb;
        C objc;
        objb.disp ();
        objc.disp ();
}
```

(ii)
```
int main ()
{
    A *pa;      // Base class pointer
    B objb;     // B class object
    C objc;     // C class object
    pa = & objb; // address of derived class B
    pa->disp(); // calling disp of class B
    pa = & objc; // Address of derived class C
    pa -> disp(); // calling disp of class C
    return 0;
}
```

Output - (i) I am from Derived Class B
I am from Derived Class C

or Error आयेगा क्योंकि abstract class का object नहीं बनाया जा सकता है किन्तु उसका pointer बनाया जा सकता है।

Note - जिस class में कम से कम एक Virtual function हो उसे abstract class कहते हैं।

or

```
A *pa, *paa;
B objb;
C objc;
pa = & objb;
paa = & objc;
pa -> disp();
paa -> disp ();
return 0;
```

or

```
A *pa[2];
B objb;
C objc;
pa[0] = &objb;
pa[1] = &objc;
(pa[0]) -> disp();
(pa[1]) -> disp();
return 0;
```

Output - (ii) I am from Derived Class B
I am from Derived class C

Dynamic Memory Allocation -

~~Dynamic Memory Allocation~~

new and delete -

- new allocates Memory dynamically.
- delete is used to free memory allocation which created by using new

New, Access and Delete -

New -

Syntax Pointer name = new data type;

Ex- int *p;

p = new int; // p is pointer name

or

int *p = new int;

Access -

*p = 10;

int *p = new int (10); // *p = 10

cout << *p; // print value

Delete -

delete p;

One D Dynamic Memory Allocation-
One -D Array

New-

Syntax- Pointer_name = new data_type [size];

Ex- int *p;

p = new int [size]; // return base address of array

or

int *p = new int [size]

Access-

```
for (i=0; i<size; i++)
{
    cin >> p[i];   // *(p+i);
}
```

Delete-

```
delete []p;
```

Note- पुराने C++ के Librery में size भी लिखना पड़ता था

→
```
#include <iostream>
using namespace std;
int main ()
{
    int a[5], n, i, total = 0;
    cout << "Enter no. of Elements:";
    cin >> n;
    cout << "Enter Elements:\n";
    for (i=0; i<n; i++)
    {
```

```
        cin >> a[i];
    }
    cout << " Elements are: ";
    for (i=0; i<n; i++)
    {
        cout << '\t' << a[i];
        total = total + a[i];
    }
    cout << "\nTotal: " << total;
    return 0;
}
```

Output- (i) Enter no. of Elements: 4
Enter Elements:
1
2
3
4
Elements are: 1 2 3 4
Total: 10

(ii) Enter no. of Elements: 8
Enter Elements:
1
2
3
4
6
7
8
Elements are: 1 2 3 4 6 23 6 1967938
92
Total: 1967938544

Note- Static Memory Allocation में जितना declare किया गया है उतनी ही Memory Allocate होती है। इस लिये Garbage Value आ रहा है।

```
-> #include <iostream>
using namespace std;
{
    int n, i;
    int *p , total = 0;
    cout << "Enter No. of Elements:";
    cin>>n;
    p = new int [n];
    cout<<"Enter" <<n <<"Elements:"<< endl;
    for (i=0; i<n; i++)
    {
        cin >>p[i];
    }
    cout <<"\n Elements are :";
    for (i=0; i<n; i++)
    {
        cout <<'\t'<< *p(p+i);    // p[i]
        total = total +p[i];
    }
    cout << "\n Total:" << total;
    return 0;
}
```

Output- Enter No. of Elements:4
Enter 4 Elements:
1
2
3
4
Elements are: 1 2 3 4

Dynamic Memory Allocation in 2D Array -

New-

Syntax-
```
data_type **pointer_name = new data_type *[row_size];
for (int i=0; i<row_size; i++)
{
    pointer_name[i] = new data_type[col_size];
}
```

Ex-
```
int **p = new int *[row_size];
for(i=0; i<row_size; i++)
{
    p[i] = new int [col_size];
}
```

Access-

Syntax-
```
for (i=0; i<row_size; i++)
{
    for(j=0 j<col_size; j++)
    {
        cin >> p[i][j]
    }
}
```

Delete-

Syntax-
```
for (i=0; i<row_size; i++)
{
    delete [] p[i];
}
delete[]p;
```

```
#include <iostream>
using name space std;
int main()
{
    int i, j, row, col;
    cout << "Enter No. of Rows:";
    cin >> row;
    cout << "Enter No. of Columns :";
    cin >> col;
    cout << "Enter 1st Matrix Elements : \n";
    // Dynamic Allocation for 1st Matrix.
    int **p = new int *[row];
    for (i=0; i<row; i++)
    {
        p[i] = new int [col];
    }
    // Strong Value for 1st Matrix
    for (i=0; i<row; i++)
    {
        for (j=0; j<col ; j++)
        {
            cin >> p[i][j];
        }
    }
    cout << "Enter 2nd Matrix Elements : \n";
    // Dynamic Allocation for 2nd Matrix.
    int **p1 = new int*[row];
    for(i=0; i<row ; i++)
    {
        p1[i] = new int [col];
    }
```

```
// Storing Value for 2nd Matrix
for (i=0; i< row;i++)
{
    for(j=0; j<col; j++)
    {
        cin >> p1 [i][j];
    }
}
// Printing of 2nd Matrix
cout <<"1st Matrix: \n";
for(i=0; i<row, i++)
{
    for (j=0; j< col; j++)
    {
        cout << p[i][j] << '\t';
    }
    cout <<endl;
}
// Printing of 2nd Matrix
cout <<"2nd Matrix: \n";
for(i=0; i<row; i++)
{
    for(j=0; j<col; j++)
    {
        cout<< p1[i][j] <<'\t';
    }
    cout << endl;
}
// Dynamic Allocation for Total Matrix
int **sum = new int* [row];
for (i=0; i< row; i++)
```

```
        sum[i] = new int [col];
    }
    // Addition of Matrix
    for( i=0; i<row; i++)
    {
        for(j=0; j<col ; j++)
        {
            sum [i][j] = p[i][j]+ p1[i] [j];
        }
    }
// Printing of Total Matrix
cout << "Total:\n";
for (i=0; i<row; i++)
{
    for(j=0; j<col; j++)
    {
        cout << sum [i][j] <<'\t';
    }
    cout <<endl;
}
// free Memory of First Matrix
for (i=0; i<row; i++ )
{
    delete [ ] p[i];
}
delete [ ] p;
// Free Memory of 2nd Matrix
for( i=0; i<row ; i++)
{
    delete[ ] p1 [i];
}
```

```
delete [] p1;
// Free Memory of Total Matrix
for (i=0; i<row; i++)
{
    delete [] sum[i];
}
delete [] sum;
return 0;
}
```

Output- Enter No. of Rows: 3
Enter No. of Colume: 3
Enter 1st Matrix Elements:
1
2
3

4
5

6
7

8
9

Enter 2nd Matrix Elements:
11
22
33

44
55
66
77
88
99

1st Matrix:

1	2	3
4	5	6
7	8	9

2nd Matrix:

11	22	33
44	55	66
77	88	99

Total:

12	24	36
48	60	72
84	96	108

Dynamic Memory Allocation in 3D or Multi D Array.

New-

Syntax-
```
data_type *** pointer_name = new data_type **[f_size];
for (i=0; i<f_size; i++)
{
    pointer_name[i] = new data_type * [s_size];
    for ( j=0; j<s_size; j++)
    {
        pointer_name[i][j] = new data_type [t_size];
    }
}
```

Ex-
```
int ***p = new int ** [f_size];
for (i=0; i<f_size; i++)
{
    p[i] = new int *[s_size];
    for (j=0; j<s_size; j++)
    {
        p[i][j] = new int [t_size];
    }
}
```

Access -

Syntax:

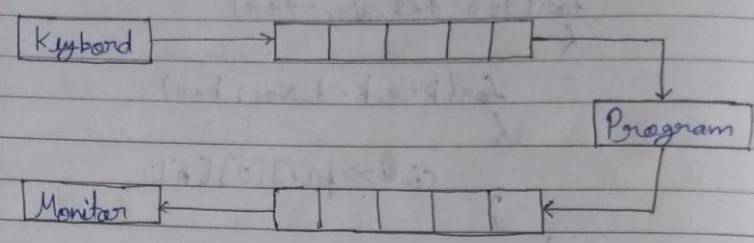
```
for (i=0;i<f_size;i++)
{
    for(j=0;j<s_size;j++)
    {
        for(k=0;k<t_size;k++)
        {
            cin>>p[i][j][k];
        }
    }
}
```

Delete -

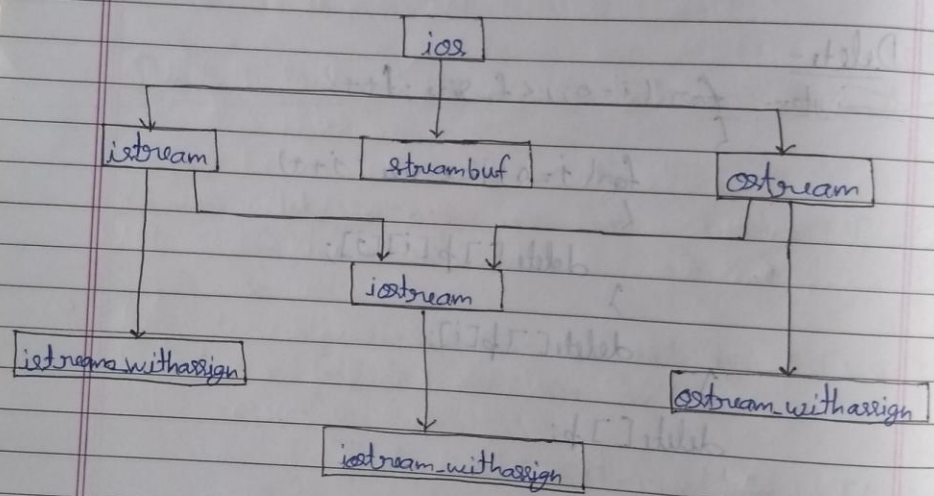
Syntax:

```
for (i=0;i<f_size;i++)
{
    for(j=0;j<s_size;j++)
    {
        delete[]p[i][j];
    }
    delete[]p[i];
}
delete[]p;
```

Streams- A stream is a sequence of byte data stream or data flow stream-



Stream Class-



Formatted I/O-

- Width()
- precision()
- fill()
- setf()
- Unsetf()

Unformatted I/O -

- << (insertion operator)
- >> (extraction operator)
- put()
- get()
- getline()
- ignore()
- putback()
- peek()
- read()
- write()
- gcount()

get() and put()

- get(char *), get(void)

get() character by character read करता है और put write करता है।

→
```
#include<iostream>
using namespace std;
int main()
{
    char str;
    cout<<"Enter a character:";
    cin.get(str);
    cout<<"You Entered:";
    cout.put(str);
    return 0;
}
```

Output- Enter a character: r
You Entered: r

(Note- one character read और write करने के लिये।)

→
```
#include<iostream>
using namespace std;
int main()
{
    char str;
    cout<<"Enter characters:";
    while(str!='\n')
    {
        cin.get(str);
        cout.put(str);
    }
    return 0;
}
```

Output- (i) Enter characters: Greekyshows
Greekyshows
(ii) Enter characters: Greeky shows
Greeky shows

Note- यह Space को भी read करता है जबकि cin space को read नहीं करता है

→
```
#include<iostream>
using namespace std;
int main()
{
    char c;
    cout <<"Enter a character:";
    c = cin.get();
    cout<<"You Entered:"<<c;
    return 0;
}
```

Output- Enter a character: g
You Entered: g

→
```
#include <iostream>
using namespace std;
int main()
{
    char line[30];
    cout<< "Enter Text:";
    cin.get(line,30);
    cout << line;
    return 0;
}
```

Output- (i) Enter text: abcdefghijkl mnop qrst afdezxy
abcdefghijkl mnop qrst afdezxy
(ii) Enter text: abcdefgh ijkl mn op qur stuvewxy
abcdefgh ijkl mn op qur stuvewxy

Note- '/0' turnary operator get count होता है।

→
```
#include <iostream>
using namespace std;
{
    char line[30];
    cout <<"Enter Text and terminate by t:";
    cin.get(line,30,'t');
    cout << line;
    return 0;
}
```

Output- (i) Enter Text and terminate by t: hjgh
mnop
rep
t
hjgh
mnop
rep

(i) Enter Text and terminate by t: abcdefghijklmn
opqr stuvwxyz e mnop qrst
abcdefghijklmn
opqr stuvwxyz e

Note- get() का bydefault terminating character new line होता है।

getline() - से line by line read करता है।
से terminating character को से साथ string में store नहीं रखता है जबकि get() रखता है।

```
→ #include <iostream>
using namespace std;
int main()
{
    char str[20];
    cout << "Enter Text:";
    cin.getline(str, 20);    or (str, 20, 't');
    cout << str;
    return 0;
}
```

Output- (i) Enter Text: New Delhi
New Delhi
(ii) Enter Text: I am from New Delhi an I
I am from New Delhi

or (i) Enter Text: New delhi
Happy
Was
New delhi
Happy
was
(ii) Enter Text: new delhi
yes
t
new delhi
yes

Ignore function-

```
→ #include<iostream>
using namespace std;
int main()
{
    char str[20];
    int a;
    cout<<"Enter any number:";
    cin>>a;
    cout<<"Enter Text:";
or  or cin.getline(str,20);   or   (str,20,'\n');
    cout<<str;
    cout<<a;
    return 0;
}
```

Output - Enter any number: 23
Enter Text: 23
or Enter any number: 23
Enter Text: geeky showe
geeky showe 23
or " " "

Note - (str, 20, '\n')
str → string
20 → 20 character को ignore करेगा।
'\n' → \n को ignore करेगा।

Formatted I/O -

- ios class functions and flags
- Stream manipulators
- User-defined output function

ios class functions and flags -

Function	Purpose
width()	To specify field size for display an output value.
precision()	To specify the number of digits to be displayed after the decimal sign.
fill()	To specify a character that fills the unused portion of an output data field.
setf()	To specify format flags such as left-justify, right-justify etc.
unsetf()	To clear / reset defined flags.

width ()-

```
# include <iostream>
int main ()
{
    int item [4]={1,10,100,1000};
    int price[4] = {12, 231, 2, 13};
    int i;
    cout.width(6);                    or
    cout<<" Item";
    cout.width(6);                    or
    cout<<" Price";
    cout << endl;
    for(i=0; i<4; i++)
    {
        cout.width(6);                or
        cout<<item[i];
        cout.width(6);                or
        cout<< price[i];
        cout<<endl;
    }
    return 0;
}
```

Output-

Item	Price	or Item Price
1	12	112
10	231	10231
100	2	1002
1000	13	100013

Note- हर item के लिये बार-बार width का use करना होगा।
जितना width देते है उतने Block create होते है और उसमें Data R to L fill किया जायेगा।

Always ahead...

precision () :- यह दशमलव के पहले से ही count करता है। यह Value को round up भी कर देता है।

→
```
#include <iostream>
using namespace std;
int main()
{
    float a = 51.123879;        or double a = 51.123879
    cout.precision(6);          or cout.precision(8);
    cout << a;
    return 0;
}.
```

Output- 51.239 or 51.123879

→
```
#include<iostream>
using namespace std;
int main()
{
    float a = 51.1204;
    float b = 23.7632;
    double c = 45.32423;
    cout.precision(4);
    cout << a << endl;
    cout << b << endl;
    cout << c;          or cout.precision(6);
    return 0;
}
```

Output- 51.12
23.76
45.32

or 51.12
23.76
45.32 42

Note- यह दशमलव के बाद यदि 0 होतो उसे नही print करता है (1.20000) 1.2

→ // width plus precision

```
# include< iostream>
using namespace std;
int main ( )
{
    float a = 51.1204;
    cout. precision(4);
    cout << a << endl;
    cout. width (7);
    cout . precision(4);
    cout << a << endl;
    return 0;
}
```

Output - 51.12
51.12

→

```
#include< iostream>
using namespace std;
int main ( )
{
    float a = 22.00/7.00;
    int i;
    for( i=1; i<8; i++)
    {
        cout. width (i+1);
        cout. precision(i);
        cout << a << endl;
    }
    return 0;
}
```

Output -
3
3 . 1
3 . 1 4
3 . 1 4 3
3 . 1 4 2 9
3 . 1 4 2 8 6
3 . 1 4 2 8 5 7

// यह 3.0 है पर precision 0 print नहीं करता इस लिये से ऐसे दिख रहा है।

fill() - इसका use width() के साथ किया जाता है।

→
```
#include <iostream>
using namespace std;
int main()
{
    float a = 12.231;
    cout.fill('x');
    cout.width(10);
    cout << a;
    return 0;
}
```

Output - ***12.231

Note - दशमलव भी count होता है width() में।

Setf() - इसका पूरा नाम set flag है।

Syntax - cout.setf(arg1, arg2)

Ex - cout.setf(ios::left, ios::adjustfield);

Format Required	Flag (arg1)	Bit-field (arg2)
Left justified output	ios::left	ios::adjustfield
Right justified output	ios::right	ios::adjustfield
Padding after sign or base indicator (like +## 30)	ios::internal	ios::adjustfield
Scientific notation	ios::scientific	ios::floatfield
Fixed point notation	ios::fixed	ios::floatfield
Decimal base	ios::dec	ios::basefield
Octal base	ios::oct	ios::basefield
Hexadecimal	ios::hex	ios::basefield

or Syntax- cout.setf (arg)

Ex- cout.setf(ios::showpoint);

Flag	Purpose
ios::showbase	Use base indicator on output
ios::showpos	Print "+" before positive numbers
ios::showpoint	Show trailing decimal point and zeroes
ios::uppercase ~~ios::showupp~~	Use upper case letters for hex output
ios::skipws	Skip blank space on input
~~ios::unitbuf~~	
ios::unitbuf	Flush all streams after insertion
ios::stdio	Flush stdout and stderr after insertion

Note- यह एक बार declare करने के बाद पूरे program में काम करेगा जब तक कि उसे reset न किया जाये। एक साथ एक से ज्यादा flag का उपयोग एक ही Program में कर सकते है।

→
```
#include<iostream>
using namespace std;
int main()
{
    cout.setf(ios::left , ios::adjustfield);
    cout.width(10);
    cout<<"Geeky";          or cout<<"Geeky"<<endl;
    cout.setf(ios::right, ios::adjustfield);
    cout<<"Shows"<<endl;
    return 0;
}
```

Output- Geeky shows (10 box) or Geeky shows (5 box)

```
#include <iostream>
using namespace std;
int main()
{
    float a=12.500;
    cout.precision(5);
                    or cout.setf(ios::showpoint);
    cout << a;
    return0;
}
```

Output- 12.5 or 12.500

Stream manipulators- इसे use करने के लिये <iomanip> का use करना होगा।

Manipulator	Purpose	Equivalent
setw(int w)	Set the field width to w.	width()
setprecision(int d)	Set the floating point precision to d	precision()
setfill(int c)	Set the fill charater to c	fill()
setiosflags(long f)	Set the format flag f	setf()
resetiosflags(long f)	clear the format flag f	unsetf()
endl	Insert a new line and flush stream	"\n"

Ex-

```
setw(10)
setiosflags(ios::showpoint);
```

Note- एक line में एक से ज्यादा manipulator का use कर सकते है।

→
```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
        cout << setw(20)<< "Geekyshows" << endl;
        cout << setprecision(4) << 12.239 << endl;
        cout << setw(10) << setprecision(4) << 12.239;
}
```

Output -
```
                    Geekyshows
12.24
          12.24
```

User-defined Manipulator -

Syntax- ostream& manipulator_name(ostream & output, args...)
```
{
    // body of user defined manipulator return output;
}
```

Ex-
```
ostream & currency(ostream & output)
{
    output << "Rs";
    return(output);
}
int main()
{
    cout<< currency <<3234.4;
}
```

Output - Rs 3234.4

Note- इसे main() के ऊपर लिखते हैं।
इसका use करने से बार-बार width या flag को नहीं लिखना पड़ता है।

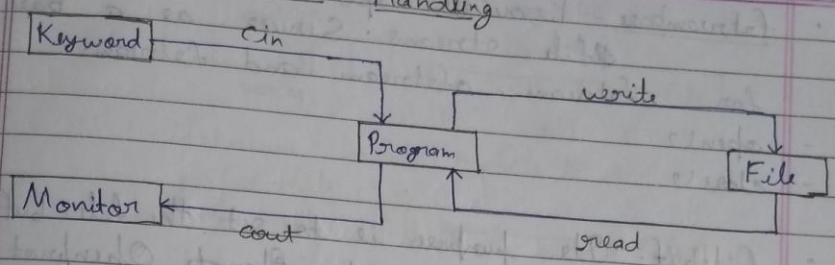
→
```
#include<iostream>
#include <iomanip>
using namespace std;
ostream & geekyshow (ostream & output)
{
    output << "Rs.";
    return output;
}
int main()
{
    cout<<geekyshow << 12.2000;
}
```

Output- Rs.12.2

→
```
#include<iostream>
#include <iomanip>
using namespace std;
ostream & geekyshow (ostream & output)
{
    output.setf (ios::showpoint);
    output<< setprecision(4);
    output << setfill('#');
    output.width (10);
    return output;
}
int main()
{
    cout <<geekyshow << 12.2000 <<endl;
    cout<< geekyshow << 40.2000;
}
```

Output- ######12.20
######40.20

File Handling



<fstream.h>

Classes-

- ifstream - Provides functions for input operations. It inherits all the functions from istream class.
  - get()
  - getline()
  - read()
  - seekg()
  - tellg()
  - open() with default input mode

- ofstream - Provides functions for output operations. It inherits all the functions from ostream class.
  - put()
  - write()
  - seekp()
  - tellp()
  - open() with default output mode

- fstream - Provide support for input and output both operations. It inherits all the functions from istream and ostream classes through iostream.

- fstreambase - Provides operations common to the file streams. Serves as a base class for ifstream, ofstream and fstream.
  - open()
  - close()

- filebuf - Its purpose is to set the file buffers to read and write. ~~Operati~~ Openprot constant used in the open() of file stream classes. Also contain close() and open() as members.

Opening Method -

- Using constructor function of the class.
  - Only one file in the stream.

- Using the member function open() of the class.
  - Multiple file using one stream.

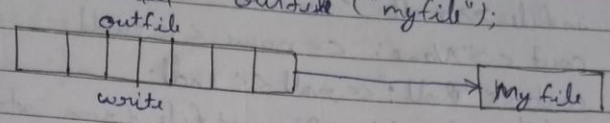
Opening and closing file using Constructor -

Syntax - file_stream_class_name stream_object_name("file_name");

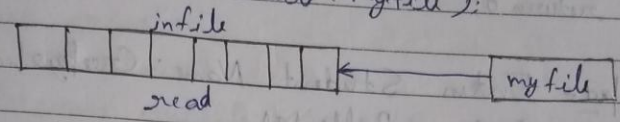
Note - File के Name में हम उसका extantion भी लिख सकते हैं।

- stream class 3 हैं। -
  1. ifstream
  2. ostream
  3. fstream.

Ex-(i) ofstream outfile ("myfile");



(ii) ifstream infile ("myfile");



Disconnect stream-

Syntax- stream object name.close();
Ex- outfile.close();

```
# include <iostream>
# include <fstream>
using namespace std;
int main()
{
    char name [50];
    int roll;
    ofstream outfile ("Student"); //Connect Student file to stream
    cout << "Enter Student Name:";
    cin >> name;
    Cout << "Enter Roll ~~Student Name~~:";
    cin >> ~~name~~ roll;
    outfile << name << endl; //write to file student
    outfile << roll << endl; //write to file student
    outfile close(); //Disconnect file from stream
    ifstream infile ("Student"); // Connect file to stream
    infile >> name; // read from file
```

```
    infile >> roll; // read from file
    cout << "Name:" << name << endl;
    cout << "Roll:" << roll << endl;
    infile.close(); // Disconnect file from stream
    return 0;
}
```

Output- Enter Student Name: Greekyshouse
Enter Roll: 101
Name: Greekyshouse
Roll: 101

In folder file- Greekyshouse
101

Opening & Closing file using open() -

Syntax- file_stream_class_name stream_object_name;
stream_object_name.open("file-name");

Ex-

```
ofstream outfile;
outfile.open("my file");
- - - - - ;
outfile.close();
outfile.open("myfile");
- - - - - - - ;
outfile.close();
```

Note- इसमें एक ही stream create होता है

```
→ #include <iostream>
  #include <fstream>
  using namespace std;
  int main ()
  {
      char name[50];
      int roll,mark;
      ofstream outfile; // creating output stream
      outfile.open ("Student1"); // connecting 1st file to output stream
      cout<< "Enter Student Name:";
      cin >> name;
      cout<< "Enter Roll:";
      cin >> roll;
      outfile<< name << endl; // write to file student
      outfile<< roll << endl; // write to file student
      outfile.close(); // Disconnect file from stream
      outfile.open ("Student2"); // connecting 2nd file to output stream
      cout<<" Enter Student Name:";
      cin >> name;
      cout <<" Enter Roll:";
      ~~cout~~ cin >> name;
      cout <<"Enter Roll:";
      cin >> roll;
      cout<<"Enter Marks:";
      cin >> mark;
      outfile << name<< endl; // write to file student
      outfile<<roll< endl; // write to file student
      outfile<< mark<< endl; // write to file student
      outfile.close(); // Disconnect file from s
      ifstream infile; // create input stream
      infile.open("Student1"); // Connecting 1 file to input stream
```

```
infile >> name; // read from file ~~student1~~
infile >> roll; // read from file ~~student1~~
cout << " Name :" << name << endl;
cout << "Roll :" << roll << endl;
infile.close (); // Disconnect file from stream
infile.open ("Student 2"); // connecting 2nd file to input stream
infile >> name; // read from file
infile >> roll; // read from file
infile >> mark; // read from file
cout << " Name : << name << endl;
cout << "Roll :" << roll << endl;
cout << "Mark :" << mark << endl;
infile.close (); // Disconnect file from stream
return 0;
}
```

Output- Enter Student Name : Greeky
Enter Roll : 101
Enter Student Name: Shouvi
Enter Roll: 102
Enter Marks: 222
Name: Greeky
Roll: 101
Name : Shouvi
Roll: 102
Mark: 222

In folder file - (i) Student 1-

Greeky
101

(ii) Student 2-

Shouvi
102
222

-5 //Write direct into file

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    char site [50];
    int subscriber;
    ofstream outfile;
    outfile.open("data.txt");
    outfile <<" Greekyshows "<<endl;
    outfile<<" 1000"<<endl;
    outfile.close();
    ifstream infile;
    infile.open("data.txt");
    infile >> site;
    infile >>subscriber;
    cout <<" Website Name:" << site << endl;
    cout <<" Total Subscriber : "<< subscriber <<endl;
    infile.close();
    return 0;
}
```

Output- Website Name : Greekyshows
Total Subscriber : 1000

In folder file - Greekyshows
1000

Note- infile & outfile Object name है इसका नाम इच्छा अनुसार कुछ भी रख सकते हैं।

- यदि एक Name का file पहले से है और उसी Name का file फिर create करे तो पुराने file की सभी data नये file से replace हो जायेंगे। पर file का extention change होने पर यह New file बना देगा।