C++

compiler
Turbo C++
Dev C++
Code::Blocks

Basics:- ① C++ introduction ② Structure of C++ ③ Download
④ Run first C++ program ④ Compilation & execution process of C++
⑤ Datatype ⑥ Variables ⑦ Keyword ⑧ Constants ⑨ Indetifiers
⑩ operators

C++ - high level semi-objet oriented programming lang. developed by Bjarne Stroustrup in 1979, at "Bell lab".
(English + mathematical; procedural, object concept; class & concept के साथ)

Note:- (i) C++ is a case-sensitive lang.
(ii) In earlier the name of C++ was "C with classes"
(iii) C++ is a portable programming language. (computer can be used other; windows 7 32bit run platform)

Syntax:- Structure:-

```
#include <iostream.h>      → Headerfile section // compulsory
#include <conio.h>         class // optional but comp. in Turbo C
Preprocessor directive     extention. comment (detail discription to readable) statement
return-type main()         → main function.
{  open curly bracket
   // code; — execute
}  end
```

C++ input → cin >> (predefine obj. extraction operator)
output → cout << (point insertion operator)

How to Download "Turbo C++"
search engine - turbo C++ filehorse.com (website) → free Download - Start download C++ press [2.6MB] size [exe file] → install.
file → new.
downloads and snown folder → Turbo C++ → Double click → yes - Next → I accept → ...

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

save with .cpp

Keyword c++
syntax error

include iostream
int char float double public

Compilation & execution process

First.cpp (Save) — extension
↓
Preprocessor (link headerfiles)
↓
compiler
↓
First.obj file get
↓
Linker (add more than one obj files & libraries)
↓
First.exe file result
↓
Loader (loader loads .exe file into main memory in hard disk)
↓
program Run

Datatype:- type of value means what kind of value the variable will store

C++ has 3 types of datatype:-
1. Basic datatype | Primitive
2. Derived D.T
3. User-defined D.T

Size depend on compiler.

Basic D.T		Size	Range
1	int	4 byte	-32768 to 32767
2	char	1 "	-128 to 127 (0 to 255)
3	float (fraction value)	4 "	$-3.4E^{38}$ to $3.4E^{38}$
4	double	8 "	
5	bool	1 "	true or false
6	void	No size	

2. Derived D.T
(i) Array
(ii) function
(iii) Pointer
(iv) Reference

3. User-defined D.T
1. Structure
2. Union
3. Class
4. Enumeration
— take any value.

Variable:- name of memory location where we store value. value can be change using execution time.

Note:- 10 (container, 1245 - address)
int a = 10 — address

1. variables are case sensitive in C++
2. In C++ variable be starts with either (a-z, A-Z) or (_ (underscore))
3. we can't give extra spaces b/w the variable

num = 10 ✓
_a2 = 10; — variable can be change while running
a2 = 10
n_u_m = 10 ✓

```
#include<iostream.h>
#include <conio.h>
void main()
{
 int a=10; // local
 clrscr();
 cout<<a;
 getch();
}
```

Types of variables
1. Global, 2. Local, 3. static

Global declaration section.

// types of variables.

```
# include <iostream.h>
# include < conio.h>
  int b=20; //global   static int d=10; //static
  void main()
  {
     int a=10; //local
     static int c=30;   //static
     cout<<a<<" "<<b<<" "<<c
     getch();
  }
```

Keyword :- is nothing but reserved word, whose meaning already defined on the compilar

Note :- ① we can't use keyword as a variable & const. name

② keyword must be in lower case

int, char, float, static, inline, double, struct, class, template etc...

INT is not keyword

int float=10; ✗

Constant :- fixed value, which does not change in runtime.

① Const keyword is used to delare a constant

② Constants can be of an datatype

③ Constants are also called literals,

④ we can change the value of constant forcely using pointer.

```
const int a=10;
   a=20; //error
   cout<<a;
```

```
void main()
{
   const int a=10;
   class c();  - consider class
   //a=20;   //error
   cout<<a;
   getch();
}
```

Identifier :- refers a name used to indentify a function, class, module or any other user-defined item.

Note :- keywords can't be used as identifier name. ex :- int (a) identifier, void fun(), class A

Operators :- Symbol that is used to perform mathematical & logical task

C++ operators :- ① Arithmetic (+, -, *, /, %)

② Relational (>, <, >=, <=, ==, !=) comparison in condition

③ logical (&&, ||, !) and, or, not

④ Assignment (=, +=, -=, *=, /=) compound

⑤ Ternary (?:) → if else not use

⑥ Bitwise (&, |, ^, ~, <<, >>) and, or, xor, complement, left shift, right shift

Conditional Statement :- (i) if

(ii) if else

(iii) nested if-else

(iv) else-if ladder

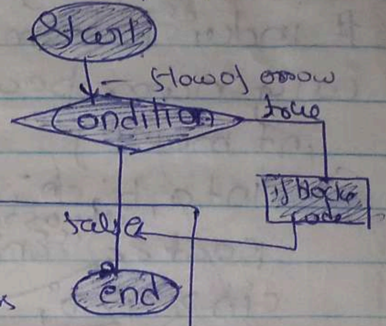
(v) switch statement

(i) if statement = If statement test, conditions, if condition is true, then if block code will be executed, not active in ...

syntax:- (write a program)

```
if (condition)
{
   // codes;
}
```

flowchart: graphical rep. of any program & condition



W.A.P to show, the simple example of if?

```
# include<iostream.h>
include<conio.h>
void main()
{ int atm;            clrscr();
  cout<<"Enter atm pin";
  cin>>atm;
  if (atm== 2731)
  {
    cout<<"you can access your balance";
  }
  getch();
}
```

atm balance access

(1) ATM Project.

Turbo C++

withdrawn deposit

Def. h not

(ii) if - else statement:- used to execute two statements for a single condition, if given condition, is true if block executes otherwise else block will be executed.

Syntax:-

```
if (condition)
{
  Statement 1;
}
else
{
  Statement 2;
}
```

ctrl copy, ctrl z undo, redo, execute, compile, run, console

*| max = (a>b) ? a : b;
cout << max;

W.A.P to give example of if-else?

def C++ compiler. File → New → Sourcefile

(2)

```
# include<iostream.h>
using namespace std;     └ Standard library
int main()
{ int a,b, max;
  cout<<"Enter two numbers:";
  cin>>a>>b; // a=20 b=15
  if (a>b) 20>15
  {
    cout<<"a="<<a;
  }
  else
  {
    cout<<"b="<<b;
  }
  return 0;
```

Headerfile, predefined access oops

ternary operator program

run time input

Greater no.

(iii) else - if ladder statement - used when we have multiple statements (more than one conditions) Syntax:-

ternary operator multiline comment

```
if (condition1)
{
   Statement 1
}
else if (condition 2)
{
   condi stat2
}
else
{ stat 3;
```

W.A.P to show the example of if - else - if statement?
Calculator program x, -, *, +, /, %

```
#include<iostream>
using namespace std;
int main()
{ int a,b,ch,add,sub,multi,Div;
  cout<<"Enter 2 nos";
  cin>>a>>b;
  if(ch=1)
  {add=a+b;
   cout<<add;
  }
  else if(ch=2)
  { sub=a-b;
    cout<<sub;
  else if(ch==3)
  { multi=a*b;
    cout<<multi;
  else if(ch==4)
  { div=a/b;
    cout<<div;
  }
  else
  {cout<<"Invalid Task";
  }
  return 0;
}
```

calculator program.

Nested - if - else statement whenever we defined if-else block inside another if - else block called nested - if else statement. Syntax:- if (condition1)

```
{
   if(condition2)
   {statement1;
   }
   else(
   {statement2;
   }
else
{
   if(condition3)
   {statement3.
```

```
else
  { statement4;
  }.
}
```

Greater b/w 3 nos

```
#include<iostream>
using namespace std;
int main()
{
int a=10,b,c;
cin>>b>>c;
if(a>b)||10
{
  if(a>c)
  {cout<<"a="<<a;
  }
  else
  {cout<<"c="<<c;
  }
}
else
{ if(b>c)||20>30
  {
    cout<<"b="<<b;
  }
  else
  {cout<<"c="<<c;
  }
return 0;
}
```

run time input in compilte

assignment operator program

output c=30 (10)

(Greater Program)

```
#include<iostream>
using namespace std;
int main()
{
int a=10,b=2,c=3;
cout<<a<<endl<<b<<c;
return 0;
}
```

(11)

assignment operator program

Real life situation → exam

Switch statement : Switch statement used when we want to select only one case out of multiple cases.

Syntax:

```
Switch(exp)
{
Case 1: statement1;
        break;
Case 2: statement2;
        break;
Case 3: statement3;
        break;
default: statement;
}
```

(12)

switch body दोहर
बार execut.

Weekned name print

```
#include<iostream>
using namespace std;
int main()
{
    int ch;
    cout<<"Enter user choice:";
    cin>>ch;
    switch(ch)
    {
    case 0: cout<<"Sunday";
    break;
    case 1: cout<<"Monday";
    break;
    case 2: cout<<"Tuesday";
    break;
    case 3: cout<<"Wednesday";
    break;
    case 4: cout<<"Thursday";
    break;
    case 5: cout<<"friday";
    break;
    case 6: cout<<"Saturday";
    break;
    default: cout<<"Invalid choice";
    }
    return 0;
}
```

(13.) Day program

def ctt do not support void

Enter user choice:

and curly bracket

Logic operation Program

also || &&

relational operation Program < > >= != ==

```
#include<iostream>
using namespace std;
int main()
{
cout<<((10<5)&&(8<8)||(30<5))<<endl;
cout<<((10>=5)||(20<=5))<<endl;
cout<<((10!=5)&&(20==20))<<endl;
cout<<!(20>10);
return 0;
}
```

(14.) logic operator program

set hard logic

and or not

relational operator program

HW || logical -- leap year

|| Relational operator -> Tax calcalation.

10 thousand

11 - 20 10%

20 - 11 alc 20%

Looping statement

(i) while - entry control loop -> loop body से पहले condition.

(ii) Do-while - exit control loop -> loop के बाद condition check

(iii) for

(iv) nested - array (2D)

Increment & decrement operator

++ -- used to 1 value of variable

by 1

Pre increment (++a)

Post increment (a++)

used to d the value of variable by 1 Types — Pre decrement, Post Decr.

Loop:- Loop is nothing but iterative statement which allow block of code to be executed repeatedly.

Types of loop:- 1) while loop: Also known as "entry controlled loop". The statement will be executed continuosly until the given condition is no loger satisfied.

Syntax:- while (condition)
{ statement;
increment; }

W.A.P to show the simple example of while loop 1-10 natural no.

```
#include <iostream>
using namespace std;
int main()
{
int a=10;
cout<<a;
++a
cout<<a;
++a
cout<<a;
++a
cout<<a;
++a
cout<<a;
++a
cout<<a;
++a
cout<<a;
++a
cout<<a;
++a
cout<<a;
++a
cout<<a;
return 0;
}
```

(15.) without using loop

printing number program

```
#include <iostream>
using namespace std;
int main()
{
int a=1, b=2, c=3, d=4, e=5, f=6,
g=7, h=8, i=9, j=10; — 10 variable.
cout<<a<<b<<c<<d<<e<<f<<g<<h
<<i<<j;
return 0;
}
```

(16)

```
#include <iostream>
using namespace std;
int main()
{
int a=1;
while(a<=10) // For(a>=1)
{
cout<<a; //cout<<a<<endl;
++a; //--a;
}
return 0;
}
```

(17) using loop

execution & use-

```
#include <iostream>
using namespace std;
int main()
{
```

variable name

P.T.O

generally we use condition we don't know in data structure

no. of digits in no. input [1000 / 4]

```
#include<iostream>
using namespace std;
int main()
{
    int num, count=0;
    cout<<"Enter Number:";
    cin>>num; //1000
    while(num>0) // 1000>0   undefined structure (data structure)
    {
        num=num/10; // 100
        count++; 0-1
    }
    cout<<count;
    return 0;
}
```

no. of digit find (18) Hard

for loop - Unlike while loop, for loop performs all operations in a single line.

Syntax - for (initialization; condition; update incr/decr)
keyword

```
{
    //block of code;
}
```

loop body

generally used when we know the condition.

1 - 50 natural no. Print [1 50]

```
#include<iostream>
using namespace std;
int main()
{
    int num, i;
    cout << "Enter Number:";
    cin >> num;
    for (i=1; i<=num; i+1)
    {
        cout << i << endl;
    }
    return 0;
}
```

Printing no. program (19)

```
#include <iostream>
using namespace std;
int main()
{
    while(1)
    {
        cout<<"Ankit";
    }
    return 0;
}
```

Real life program execution

do-while loop - also known as "exit-controlled loop" because it tests condition at the end of loop body.

Syntax:

```
do - keyword
{
    //statement; code
    incr/decr;
}
while(condition);
```

without depend on condn

Note: ① In executes at once even the given condition is true or false

WAP to show the simple example of do-while loop.

natural no 1-20

```
# include<iostream>
using namespace std;
int main()
{
  int a=1;
  do
  { cout<<a;
    a++;
  }
  while (a<=20);
  return0;
}
```

(20)

Statement एक बार execute होते हैं। Printing no program

```
# include <iostream>
using namespace std;
int main ()
{
   int a = 1;
   do
   { cout<<a;
     a++ //a=2
   }
   while(a>20); //2>20
   return0;
}
```

(21)

Jumping statement - working to jump control of program from one place to another

(i) break - loop terminate (remove from current iteration)
(ii) continue - skipe
(iii) goto
(iv) exit (0) - terminate the program
(v) return - functions call study, return value return call

break:-

```
# include<iostream>
using namespace std;
int main()
{ int i;
  for (i=1; i<=10; i++)
  { if (i==5)
     break;
  }
  return0; — loop terminate
}
```

(22)

1,2,3,4

break is used to leave us from current iteration

or

continue jump statement

```
# include <iostream>
using namespace std;
int main()
{
  int i;
  for(i=1; i<=10; i++)
  { if (i==10)
     break;
     cout<<i<<endl;
  }
  return0;
}
```

(22)

1,2..9

continue-

```
# include<iostream>
using namespace std;
int main()
{
 int i;
 for (i=1; i<=10; i++)
 { if (i==5)
   continue;  - skipe   skiping program
   cout<<i<<endl;
 }
 return0;  Printing no program
}
```

go-to:-

```
# include iostream
using namespace std;
int main()
{
 int age;
 cout<<"enter your age";
 cin>>age;
 if (age>=18)
   goto vote;
 else
   go-to notvote;
 vote: cout<<"eligible for vote..!";
```

(25)

- vote Programming

colon / semicolon

complete label

message pass

go-to — complete label

if else program

```
notvote:-
 cout<<"Not eligible for vote--!";
 return0;
}
```

exit'- program terminate

```
# include<iostream>
using namespace std;
int main()
{
  int age;
  cout << "Enter Your age";
  cin >> age; // age 67
  if (age >= 18)
    goto vote;
  else
    got notvote;
  vote:
    cout << "Eligible for vote...!";
    return 0;
  notvote:
    cout << "Not eligible for vote...!";
  return 0;
}
```

(26) vote program

```
# include<iostream>
using namespace std;
int main()
{
cout << "C language" << endl;
cout << "C++ Programming" << endl;
exit(0);
cout << "Java" << endl;
cout << "Python";
return 0;
}
```

(27) To check for exit.

Array:- is a derived Datatype which is constructed by the help of primitive datatype. It stores multiple values (same datatype) in single variable with continous memory location.

Derived Datatype:-

(i) Array
(ii) String
(iii) Pointer
(iv) Function

WAP to show the example of array

Syntax:- datatype arr. var [size]

int arr[5]; address location element

block

arr	10	20	30	40	50
	0	1	2	3	4

Index

Types of array → 1D / → 2D matrix form print array

0 possition index

1D

```
#include<iostream>
using namespace std;
int main()
{ // array hold multiple value
int arr(5) = {10,20,30,40,50};
int i; - declare
for (i=0; i<5; i++)   print array no.
{
cout << a[i] << endl;
}
return 0;
}
```

```
#include - -
{
int arr[5] - -
Cout << arr(0);
return 0;
}
```

or check

10

output

Enter array element 10
20
30
40
1020
3040

cout << endl

0 return

2D → array runtime value input

nested for loop syntax

row colum element input

```
# include<iostream
using namespace std;
int main()
{
  int a[2][2], i, j;
  cout << "Enter array Elements";
  for (i=0; i<=1; i++)
  {
    for (j=0; j<=1, j++)
    {
      cin >> a[i][j];
    }
  }
  cout << "\nArray Elements:";
  for (i=0; i<=1, i++)
  {
    for (j=0; j<=1; j++)
    {
      cout << a[i][j]
    }
  }
```

(29)

Matrix form:-

```
#include<iostream>
using namespace std;
int main()
{
  int a[2][2], i, j;
  cout<<"Enter Array Elements:";
  for(i=0; i<=1; i++)
  {
    for(j=0; j<=1; j++)
    {
      cin>>a[i][j];
    }
  }
  cout<<"\n Array Elements\n";
  for(i=0; i<=1; i++)
  {
    for(j=0; j<=1; j++)
    {
      cout<<a[i][j]<<ends;
    }
    cout<<endl;
  }
  return 0;
}
```

nested for loop

cin num input value

10 20 30 40

variable size

String:- 1 Dimensional array of characters terminated by null character

Syntax:- datatype str-var[size];

int str[5]; → cin>>

char str["ankit"];

Memory

a	n	k	i	t	\0
0	1	2	3	4	5

str. end

If we want to with null add 1 more char string in it or want to find length then,

String pre-defined functions

(1) str.len() - 5
(2) str.cpy() string copy
(3) str.rev() String rev.
(4) str.cat → 2 string concatenation join & so on.

String:

```
#include<iostream>
using namespace std;
int main()
{
  char str[] = "Ankit";
  cout<<str<<endl;
  int x = strlen(str);
  cout<<x;
}
```

header file

using function

parameter

concept data print

```
#include<iostream>
using namespace std;
int main()
{
  char str[] = "Ankit";
  cout<<str<<endl;
  strrev(str);
  cout<<str;
}
```

```
#include<iostream>
using namespace std;
int main()
{
  char str[] = "Ankit";
  char str[] = "Kumar";
  strcat(str, str2);
  cout<<str;
}
```

```
#include<iostream>
#include<string.h>
int main()
{
  char str[] = "Ankit";
  char str2[20];
  strcpy(str2, str);
  cout<<str2;
}
```

strlwr() upper to lower stream
strupr() = lower to upper

Pointer:- is a variable that can able to hold address of another variable

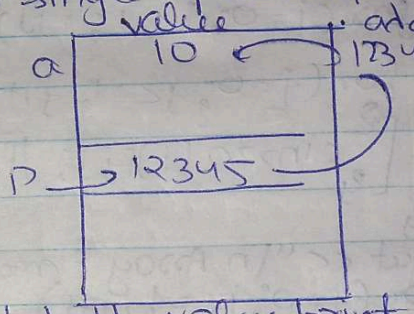
NOTE:- While working with pointer we need to required two unionary operators i.e:- work on single operand/variable

(i) & → Address of operator

(ii) * Value of address operator

Syntax:- datatype *var-name;

```
int *P, a=10;
P=&a;   hold address of a
print *P; → 10
print P; → 12345
```



add-hold-value print

```
#include<iostream>
using namespace std;
int main()
{
  int *p;
  int a=10;
  p=&a;
  cout<<&a<<endl;
  cout<<p<<endl;
  cout<<"a="<<endl;
  cout<<*p="<<*p;
  return 0;
}
```

(35)

```
#include<iostream>
using namespace std;
int main()
{
  int *p;
  int a=10;
  p=&a;
  cout<<&a;||<<endl;
  cout<<p;
  return 0;
}
```

(36)

Types:

Function:- User defined:- ① main() { } → program starting & ending point, called by operating system when program run

use may line of code ② fun() { } → body open/close, called by main only

③ add()

Storage class:-

① auto

② static

③ register

④ extern

particular code piece executed no more

malu fn:- write code call particular fn automatic run

reusability facility

Pre-defined:- name + fix with not use for program

(i) strcpy(); no body & ;X

(ii) strcat(); end with semicolon

(iii) strlen(); no code written

Function is a block of codes that runs only when it is called

Syntax of user-defined fn:

```
return-type fun name()
{
  //your code;
}
```

Storage class :- defines the scope & lifetime of variable & functions.

Types: (1) auto (local)
(2) static
(3) register
(4) extern (Global)

	Storage class	Memory	Default value	Scope	Lifetime
1.	auto	RAM	Garbage	within block	still block is active
2.	static	RAM	0	within block	till the terminated
3.	extern	RAM	0	anywhere	"
4.	Register execution fast to	Register as compare to 1,2,3	Garbage	within block	still the block is activate

Practical Program of storage class using functions:-

```
#include<iostream>
using namespace std;
int a;//extern(global)
int main()
{ int b;//auto(local)
  static int c;//static
  register int d;//register
  cout<<a<<endl;
  cout<<b<<endl;
  cout<<c<<endl;
  cout<<d<<endl;
  return 0;
}
```

2nd defect output = 0 0 0 ... result

2nd Turbo output = 0 ... 0 ...

```
#include<iostream>
using namespace std;
void fun()//defination
{
  auto int a=10;//local variable
  static int b=10;//only declare in main fn
  cout<<a<<endl<<b<<endl;
  ++a; ++b;
}
int main()
{
  fun(); //calling
  return 0;
}
```

fun(); fun(); fun();

output =

10	10
10	11
10	12
10	13

Error program

```
using std;
int a=10;
void fun()
{cout<<a<<endl<<b<<endl;
}
int main()
{
  int b=10;
  fun();
  fun();
  fun();
  fun();
  return 0;
}
```

b was not declared in this scope

Global variable

but if not print b such that cout<<a<<endl; local variable then we get, output 10 10 10 10

```
#include<iostream>
using namespace std;
int a=10;
void fun()
{
cout<<a<<endl;
}
int main()
{
int b=20;
fun();
fun();
fun();
fun();
cout<<a<<endl<<b;
return 0;
}
```

output
10
10
10
10 20

(3)

we can also pass parameter in function

```
#include<iostream>
using namespace std;
int a=10;
void fun(int a, int b)
{
a=100, b=200;
cout<<a<<endl<<b;
}
int main()
{
fun(10,20);
return 0;
}
```

local variable
int b
formal parameter
actual parameter

(4)

output
100 200

but if
a=100;
b=200

output
10
20

variable can be change

user defined Datatype :-

1. enum
2. structure
3. Union
4. class

→ (1) File Handling
→ (2) Exception Handling

Structure:- user-defined datatype which are used to store, dis-similar types of value.

NOTE:- The size of structure is equals to sum of all structure number size.

Union:- **NOTE:-** The size of union is equal to its biggest member size.

Syntax:-
```
union union-name
{
//members;
};
```

Syntax:-
```
struct structure-name
{
//members; - value
}; - semicolon
```

```
#include<iostream>
using namespace std;
union stu
{
int roll;
char name[20];
float marks;
};
int main()
{
union stu s;
cout<<"Enter student roll number:";
cin>>s.roll;
cout<<"roll number "<<s.roll<<endl;
cout<<"Enter student name:";
cin>>s.name;
cout<<"student name "<<s.name<<endl;
cout<<"enter student marks:";
cin>>s.marks;
cout<<"student marks "<<s.marks<<endl;
return 0;
}
```

(44)

```
# include<iostream>
using namespace std;
struct stu
{   seperate memory found — execution fast
  int marks; // int 4bytes
  float avg; // 4bytes
  double salary; //8bytes
};
union stu2
{   memory share at diff time → execution slow, work done in small memory
  int marks; // 4 bytes
  float avg; // 4 bytes
  double salary; // 8 bytes
};
int main()
{
  struct stu s;
  union stu2 s2;
  cout<<"structure size="<<sizeof(s);  or <<endl
  cout << "Union size="<<sizeof(s2);
  return 0
}
```

structure size=16
union size=8

4+4+8 = 16

(41)

Using Union structure we record student data

↳ only for 1 number & side wise print

```
# include<iostream>
using namespace std;
union stu
{
  int roll;
  char name [20];
  float marks;
}; int main()
{
  union stu s;
  cout << "Enter student roll number:";
  cin >>s.roll;
  cout << "Enter student name:";
  cin>>s.name;
  cout<<"Enter student marks:";
  cin>> s.marks;
  cout<<s.roll<< endl << s.name << endl
      << s.marks<<endl;
} return0;
```

(43)

Output:
1136045448
457
starting two output

Using structure we record student data

```
# include<iostream>
using namespace std;
struct stu
{
  int roll;
  char name[20];
  float marks;
};
int main()
{
  struct stu s;
  cout << "Enter student roll number:";
  cin >>s.roll;
  return 0;
  cout<< "Enter student name:";
  cin>>s.name;
  cout<<"Enter student marks:";
  cin>>s.marks;
  cout<<s.roll<<endl<<s.name<<endl<<s.marks
      <<endl;
  return0;
}
```

(42)

Output:
ankit
71 28

What is e-num type?

Enumeration is a user defined name that consists of integral constants.

Syntax:- week 12, mouth 7

```
enum enum-name {value1, value2, ---- value N};
cout << value1; → 0
enum.name var. value2
cout << var2; → 1
```

error · value excess

```
#include<iostream>
using namespace std;
int main()
{
  enum gender{
    male,female,x
  };   0     1
  cout<<male;
  return 0;
}
```

:- if this true. if it is not then error

output = 0

(44)

```
#include<iostream>
using namespace std;
int main()
{
  enum gender{          → integer provide
    male, female, others
  };
  cout<<male<<endl;
  gender str = others; // "outlet" 10
  cout<<str;                  ↓ error
  return 0;
}
```

output = 0
2

(45)

Macro:- Piece of code in a program which is given some name. Wherever the name is used, it is replaced by the contents of the macro. a=10 save time

Note:- · Macro is define by the help of #define

Syntax:- #define macro-name content

Types:- → (i) object like macro (ii) fn like macro
variable, value

use in graphics data & for gaming purpose

(i)

```
#include<iostream>
#define num 10
using namespace std;
int main()
{
  int i,r;
  for(i=1; i<=10; i++)
  {
    r=num*i;
    cout<<num<<"*"<<i<<"="<<r<<endl;
  }
  return 0;
}
```

(46)

Table of 10
output
10×1 =
2 =
3 =
4 =
5 =
6 =
7 =

```
#include<iostream>
#define num(a,b) ((a>b)?a:b)
using namespace std;
int main()
{
  cout<<num(236,167)<<endl;
  cout<<num(564,1675)<<endl;
  cout<<num(342,167)<<endl;
  cout<<num(236,453)<<endl;
  cout<<num(5643,167)<<endl;
  return 0;
}
```

10 no. एक साथ check

output
236
1675
342
5643

(48)

(ii)

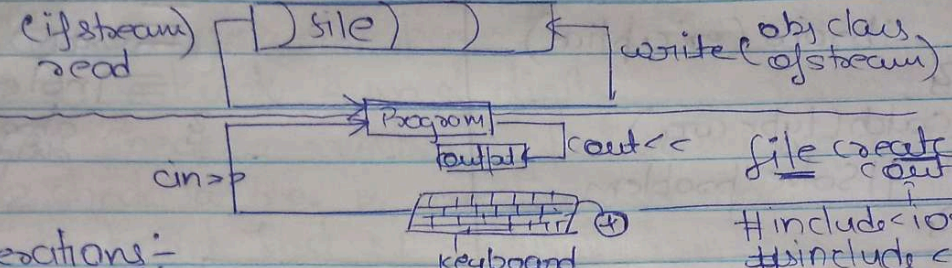
```
#include<iostream>
#define num(a,b) ((a>b)?a:b)
using namespace std;
int main()
{
  cout<<num(236,167);
  return 0;
}
```

(47)

output
236

secondary storage.

What is file Handling:- file handing is a mechanism so that we can store the output of the program in the file & we can perform many operations on the data present in a file.



RAM

file create
cout cin

File operations:-

create
write — store > offstream — see program
read — if stream

close a file

(49)

```
#include<iostream>
#include <fstream>
using namespace std;
int main()
{
  ofstream newfile
  ("C:\\Users\\WIN10\\Desktop
  \\CPP.txt");
  cout <<"file created...";
  newfile. close();
  return0;
}
```

run create output हमें पता चलना चाहिए की file create हो चुकी है Bank console se

Background run unnessary space cover.

Data write or insert

(50)

```
#include <iostream>
# include <fstream>
using namespace std;
int main()
{
  ofstream newfile("C:\\Users\\
  WIN10\\Desktop
  \\CPP.txt");
  cout <<"file created...";
  newfile. close();
  return0;
}
```

goto file
write

> I want to do some new thing
> ↓
> now save.

work by if stream

Data read by C++ program

(51)

```
#include <iostream>
include< fstream>
using namespace std;
int main()
{
string str;
ifstream newfile("C:\\Users\\WIN10
\\Desktop\\CPP.txt");
while (getline(newfile, str))
{
  cout<<str;
}
newfile.close();
return0;
}
```

output
I want to do some new thing

for successful termination of program.

Exception Handling :- An exception is unexpected / unwanted abnormal situation that occured at runtime called e...

Syntax :- try (risky code)

```
{
    throw (exception)
}
catch(type arg)
{
    // Solve problem
}
```

main()
{
≡
— exp ④
≡] not execute.
}
abnormal
तरीके से
terminate.

Program :-

```
#include <iostream>
using namespace std;
int main()
{
    cout<<"Execution Starting--"<<endl;
    int a,b,c;
    cout<<"Enter Two numbers:";
    cin>>a>>b;
    c=a/b;
    cout<<"Result:"<<c;
    cout<<"Execution ended--"<<endl;
    return 0;
}
```

output
10
0
Terminate program
no output
exception

52

```
{
    cout<<"Execution starting--"<<endl;
    int a,b,c;
    cout<<"Enter Two numbers:";
    cin>>a>>b;
    try
    {
        if (b==0)
        throw b;
        c=a/b;
        cout<<"Result:"<<c;
    }
    catch (int x)
    {
        cout<<"Can't divide by"<<x;
    }
    cout<<"Execution ended--"<<endl;
    return 0;
}
```

output
10
0
can't divide by 0

Questions DIY:-

1. WAP to Add two numbers
2. WAP to check a number is odd or even
3. WAP to make calculator using switch case
4. WAP to reverse a number
5. WAP to calculate sum of digits
6. WAP to check a number is palindrome or not.
7. WAP to check a no. is armstrong or not.
8. WAP to print first N natural numbers.
9. WAP to check a no. is perfect or not.
10. WAP to check a no. is prime or not.
11. WAP to find factorial of a no.
12. WAP to print fibonacci & tribonacci series.
13. WAP to find prime numbers blw two ranges.
14. WAP to calculate area of circles, square, rectangle & Δ.
15. WAP to check a year is leapyear or not.
16. WAP to swap two numbers.
17. WAP to sort array elemnts in ascending & descending order.
18. WAP to insert element at beginning, ending & any position of array
19. WAP to print array element in reverse order.
20. WAP to find maximum & minimum elemnt of array.
21. WAP to print transpose matrix.
22. WAP to print mirror matrix
23. WAP to add two matrix.
24. WAP to swap two matrix.
25. WAP to print palindrome string.
26. WAP to convert characters upper to lower & vice-versa.
27. WAP to convert temperature celcius to fahrenheight & vice-versa
28. WAP to add two no. using pointer.
29. WAP to print 5 student records using structure & union.
30. WAP to calculate total & average marks of 5 students.

Imp.

31. WAP to calculate tax :- Condition <= 1000 ⟶ "No tax"
> 100088 <= 100000 ⟶ "10% tax"
> 1000 ⟶ "25% tax".

32. WAP to add array elements.
33. WAP to search array elements with appropriate location.
34. Print pattern.

(i) :-
*
* *
* * *
* * * *
* * * * *

(ii) :-
* * * * *
* * * *
* * *
* *
*

(iii) :-
*
* * *
* * * * *
* * * * * * *
* * * * * * * * *

(iv) :-
①
② ③
④ ⑤ ⑥
⑦ ⑧ ⑨ ⑩
⑪ ⑫ ⑬ ⑭ ⑮

OOPs :-
- 500-600 code
- for security of program
- complex not
- we divide program in diff diff module/function

(1) class & object
(2) Access specifiers — secure of data in class
(3) Constructor & Destructor
(4) friend function
(5) Friend class

reusability maintain — define — data members — no memory — name declare

Class & object :- Class is a user-defined datatype & it has its own data-members & member functions which are used by creating an instance of class. actual use of photo — us

Syntax:
class class name
{
// data members;
public:
// member function;
};

by default private in class — Encapsulation, Data abstraction

by default public
stu union member

NOTE: By default the access specifier of class is "private"

Obj:- Object is a instance of class that have state & behaviour. (state → class data member variable)

Syntax:- Class name object name;

WAP to print a message using class & object?

```
#include<iostream>
using namespace std;     — Header file / Global section
class Print
{ public:
  void show()
  { cout << "Welcome to learn coading";
  }
};                        — function body
int main()
{
  Print obj;
  obj.show();             — behavior
}
```

access specifier
except data member value obj

Real life ex

```
#include<iostream>
using namespace std;
class Person        — logical part
{
  private:
  int run=10;
  string msg;
};
int main()
{
  Person obj;
  obj.play();
  obj.walk();
  return 0;
}
```

obj ← identity, state, behavior

today I scored 50 runs, today I walked

```
public:
  void play()
  {
    run=50;
    cout<<"today i scored "<<run<<"run"<<endl;
  }
  void walk()
  { msg="today I walked 3 kilometers";
    cout<<msg;
  }
```

by default private

obj form memory for call use

without fn we can access data member.

no of d.m & obj can be created in class

not give value
logical section part → variable | not set value (default)

for security purpose → inverted comma

Aprotected

inheritance

Accesibility of access specifiers → accessibility restrict
DM & MF

private → i) class itself
ii) friend of class

public → i) Anywhere accessible
any class main f^n other member class

protected → i) class itself
ii) inherited class member, of class
if class use

```
#include<iostream>
using namespace std;
class A
{
  private:        // data member
    int a;
  protected:
    int b;
  public:
    int c;
};
int main()
{
  A obj;
  //obj.a=10;
  //obj.b=20;
  obj.c=30;
  //cout<<obj.a<<endl; not allowed
  //cout<<obj.b<<endl; not allowed
  cout<<obj.c<<endl; //allowed
}
```

DM cube

class is logical entity

output
30

output: 10 20

class A
constructor call

```
int main()
{
  A obj;
  return 0;
}
```

Constructor:- special type of f^n which is automatically called at the time of obj creation

class name same as constructor

Note:- The main purpose of constructor is used to "initialize the obj".

Syntax:-

```
class A
{
  A()
  {
    Body
  }
};
```

Types:- Default
Parameterized
Copy

Default constructor

```
class A
{
  int a,b;
  public:
    A()  // default.
    {
      a=10; b=20;
      cout<<a<<" "<<b;
    }
};
```

Sn के बाद में paranthesis.

Parametrised constructor

```
class A
{
public: double parametrised
A(int a, int b) // x
{
cout<<a<<endl<<b;
}
};
int main() 56.5 call
{
A obj(100,200); parantisis value
return 0;
}
```

To show constructor is used to initialize the value obj

```
# include<iostream>
using namespace std;
class A
{
public:
A() // - -
{
cout<<"learn coding"<<a;
}
};
int main()
{
A obj = A(); of class 56.5
  assignment operator
return 0;   use to initilisation
}
```

output
learn coding

copy constructor

```
# include<iostream>
using namespace std;
class A
{ int x,y; — class variable
public
A(int a, int b) // par
{
x=a; y=b;
cout<<x<<endl<<y<<endl;
}
A(A & ref) // reference
{
x=ref.x;
y=ref.y;
cout<<x<<endl<<y;
}
};
int main()
{
A obj(10,20); → const 2 obj
A obj2 = obj; → initization
return 0;
}
```

Output	
10 20	P.C
10 20	C.C

Destructor - Special type of member functions that is used to dealloacate the memory by allocated by the constructor.

Syntax :-
```
class A
{
public:
A()
{
  memory
}
~A()   tilde sign
{   memory dealocted
}
};
```

friend function :- is a fn which is not the member of class instead of that it can access private & protected member of class.

Note :- friend fn is declared with keyword friend. Using friend fn we can write work with two diff classes.

Syntax :- friend return-type function-name (class ref); variable

int, void   Keyword

Manipulator endl setw()
setwith :: scope resolⁿ oper
Lib func
ios namespace

program

Destructor

```
#include<iostream>
using namespace std;
int count=0;   default section   // variable -> private -> Global
class A
{ public:
  A()
  {cout<<"object"<<++count<<"created"<<endl;     // obj create, memory allocated
  }
  ~A()
  {
  cout<<"object"<<count--<<"deleted"<<endl;      // free destructor
  };
int main()
{
  A obj, obj2, obj3;
  return0;
}
```

output
obj1 created
obj 2 created
obj 3 created
obj 3 deleted

(59)

WAP to add two nos using friend fn. 2 class private D.M.

```
#include <iostream>
using - - ;
class ankit                      // Real life problem
{ private:  bydefault private
  string tv;  — D.M
  public: — fn. call class
  void show()
  {
   tv="IPL final 2021";
  }
friend void ankush(ankit r);
};
void ankush(ankit r)
{
 cout<<"I am ankush now, Watching"<<r.tv;
}
int main()
{
 ankit a;   obj store
  a.show();
  ankush(a);
return0;
}
```

(60)

```
#include<iostream>
class akhilesh;    — Declare
class Ankit
{ private:
   int a;
public:
   void ihave()
   {
    a=10;
   }
  friend void ankush(ankit p, akhilesh p2);
};
void ankush(ankit p, akhilesh p2)
{ cout<<"I'm ankush now, watching"<<r.tv;
}
  — — — same; int c;
      c=p.a+p2.b;
      cout<<c;
class akhilesh
{ private
   int b;
 public
   void ihave()
   {b=10;
   }
 friend void ankush(ankit p, ankit p2);
```

output: 20

(70)

ch

→ copy
→ cut

friend class :- If it is a class become a friend class to a other class then it can access the all private & protected member of that class.

NOTE:- friend class is declared always with friend keyword.

Syntax:- friend class classname;

Q WAP to show the example of friend class?

```
#include<io---
class A
{ int a,b;
  public: friend class B;
};
class B
{ void add(A &o)
  { int c; public:
    c=a+b;
    cout<<c;
  }
};
int main()
{ A a; a.input();
  B obj;
  obj.add(a);
  return 0;
}
```

output = 0 garbage value

```
void input()
{ a=10;
  b=20;
}
```

Topics:- 1. Encapsulation
2. Abstraction
3. Inheritance
4. Polymorphism
5. Abstract class
6. Template
7. Namespace

Practical Program

Abstraction:- Data abstraction is a technique by which only necessary data is shown to the user & unecessary data is hidden

WAP to show the simple example of data abstraction

```
int main()
{ int a;
  a.have();
  while(a>2);
  a2.have();
  answish(a,a2);
  return 0;
}
```

In Banking Real life example

```
#include--
class myBank
{ private: int atmPIN, Balance;
  string bName, IFSC;
  public:
  int accNumber;
  int atmPIN, Balance; //not share
  void input()
  { atm PIN = 4598;
    Balance = 674589;
    bname = "Paytm";
    IFSC = "paytm0123456";
    acc Number = 45236789;
  }
  void output() //bank details
  { cout<<"My atmPIN"<<endl;
    cout<<atmPIN<<endl;
    " " Balance " "
    " " bName " "
    " " IFSC " "
    " " atmPIN " "
    " " atmPIN
  }
};
```

```
int main()
{
  myBank m;
  m.input();
  m.output();
  cout<<"saying to access my account--"<<endl;
  cout<<m.atmPIN<<endl;
  return 0;
}
```

calling

same

Encapsulation: concept of oops that is used to wrap the data members & membes function into a single unit.

NOTE: The main purpose of encapsulation is to secure the data (100%)

Syntax: class A
{ private:
// datamembers
public
// membesfn:
};

WAP to show the example of encapsulation

```
#include <iostream>
using namespace std;
class thief
{ private:
  string name, address;
  int mob;

public:
  void input()
  {
   name = "Raj";
   address = "old Purulia Road";
   mob = 5634745;
  }
  void output()
  { cout << name << endl;
    " address "
    " mob "
  }
};
int main()
{ thief t;
  t input();
  " output();
return 0;
}
```

output
Raj
old Purulia Road
5634745

Police find data of thief — inherit

chore
police

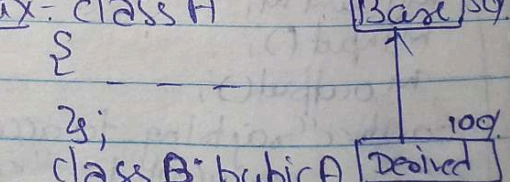
```
int main()
{ policep;
  p.input();
  p.output();
 return 0;
}
```

Inheritance: nothing but when one class access the property of another class is called inheritance.

Types:

(1) Single/Simple inheritance

Syntax: class A
{ ---
};
class B: pubic A
{ ---
};



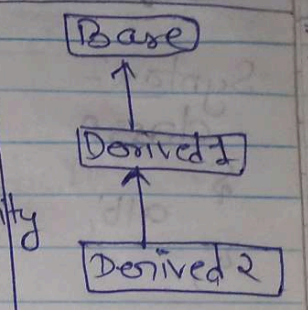
main fn & other class cannot use it.

```
class D
{
int protected amount;   } by default private, D.M
public:
void Sapit()
  {
    amount = 10000;
  }
};
class Son: public Dad
{
 int money;
  public:
   void show()
   {
    money = 45674;
    cout << "Son Money" << money << endl;
    cout << "Dad amount" << amount;
   }
};
int main()
{
  Son S;
  S.input();
  S.show();
return 0;
}
```

Multi-Level - inheritance

Syntax:-

```
class A
{
  a+b;
};
class B: public A
{
  a+b अपने में
  a-b अपना
};
class : public B
{  a+b a-b
   a*b a/b
};
```



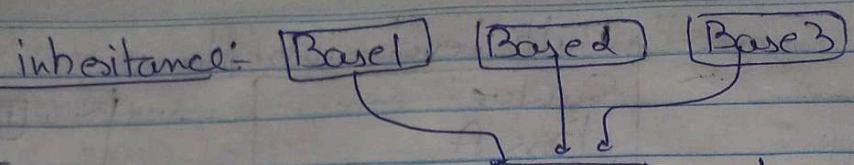
reusability

calculator program

```
#include<iostream>
using namespace std;
class A
{
 int a,b;
  public:
   void input()
   {
    cout << "Enter two numbers:";
    cin a>>b;
   }
};
class B: public A
{
 int c;
 public
   void add()
   {
    c = a+b;
    cout << "Addition" << c;
   }
  void sub()
  {
    c = a-b;
    cout << "subtraction" << c;
  }
};
class C: public B
{
 int c;
 public:
   void multipli()
   {
    c = a*b;
    cout << "Multiplication" << c;
   }
   void multi()
   {
    c = a/b;
    cout << "division" << c;
   }
};
int main()
{
 C obj;
 obj.input();
 obj.add();
 obj.sub();
 obj.multi();
 obj.div();
 return 0;
}
```

output
Enter two nos - -

Multiple inheritance:-



Syntax:-

```
class A          class B          class C
{                {                {
   a+b;             a-b;             a+b
};               };                  a-b
                                     a*b
                                     a/b;
};
```

```
class D: public A, public B, public C
{ a+b  a-b  a*b  a/b
  a%b  a>b  a<b  a==b
};
```

```
#include <iostream>
using namespace std;
class A
{ int a,b;
  public:
    void input()
    {
      cout<<"Enter two Numbers:
    }
};
class B
{
};
class C
{
};
```

```
class A
{ int a,b;
  public
    void input();
};
A void :: input()
{
  cout<<"Enter two numbers:";
  cin>>a>>b;
};
A void :: add()
{ int c;
  c=a+b;
  cout<<"Addition"<<c;
};
```

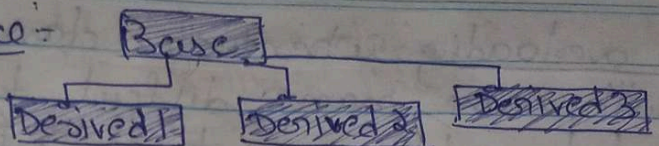
```
class B int a,b;
{ public:
  void sub();
};
B void :: sub()
{ c=a-b;
  cout<<"Subtraction"<<c;
};
```

```
class C: public A, public B
{ c=a*b;
  cout<<"Multiplication"<<c;
};
C void :: div()
{ c=a/b;
  cout<<"Division"<<c;
};
```

```
int main()
{ C obj;
  obj.input();
  obj.add();
  obj.sub();
  obj.multi();
  obj.div();
  return 0;
}
```

```
{
  void multi();
  void div();
};
```

Hierarchical inheritance:



Syntax:

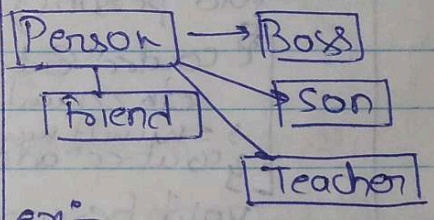
```
class A  - Base class
{
};
class public A          class public A          class public A
{                       {                       {
    a+b                     a+b                     a+b
    a-b                     a*b                     a/b
};                      };                      };
```

```
#include <iostream>
using namespace std;
class animal
{ public:
      void eat()
      { cout << "animal eating..." << endl;
      }
};
class cat: public animal
{ public:
      void voice()
      { cout << "meow meow..." << endl;
      }
};
class dog: public animal
{ public:
      void bark()
      { cout << "dog gi. woof..." << endl;
      }
};
int main();  —— obj
{ cat c;  - each and every obj have diff behaviour but in same obj fn is same here.
  dog d;
  c.eat(); c.voice();
  d.eat(); d.bark();
return 0;
}
```

96

Polymorphism:-

Poly + morphism
many form
many form
greek word whose meaning is "same object having diff behaviours"



ex:-

```
void person (Boss)
void person (friend)
void person (Teacher)
```

Types:-

(1) Compile time polymorphism (function overloading)

(2) Run-time polymorphism (function overloading)

Function overloading: whenever a class contains more than one method with same name diffent types of parameters

Not: i) This polymorphism exists at the time of compilation

ii) Compile time polymorphism is also known as early binding or static polymorphism.

Syntax: class A
{ public:
void add()
{
}
void add(int a) diff para
{
}
void add(int a, int b)
{

W.AP to show the example of function overloading?

```
#include<iostream>
using namespace std;
class A
{ int num1, num2;
  public:
    void person()
    { cout<<"Enter two numbers:";
      cin>>num1>>num2;
      s=num1+num2;
    cout<<"addition"<<s<<endl;
    void person(int a, int b)
    {
      m=a*b;
    cout<<"multiplication"<<m<<endl;
};
int main()
{ A obj;
  obj.person();
  obj.person(10,20);
  return 0;
}
```

fn-overlaping problem: In the case we can't call the base class fn using the derived class obj is known as fn overriding problem.

Function overriding:
Whenever we write a method in base & derived classes in such a way that fn name parameters must be same
NOTE: This polymorphism exists of the time of execution of program is called runtime polymorphism. Runtime polymorphism is also known as late binding or dynamic polymorphism.
Syntax:
class A
{ void add()
{
}
};
class B: public A
{ void add()
```
```

WAP to show the example of fn overlapping?
point rider.

```
#include <iostream>
using namespace std;
class A
{ public:
  void person()
  {
    cout<<"good morning";
  }
};
class B: public A
{
};
int main()
{
  B obj;
  obj.person()
}
```

Good morning

```
class A
{ public:
  void person()
  {
    cout<<"good morning";
  }
};
class B: public A
{
  public:
  void person()
  {
    cout<<"Good night";
  }
};
int main()
{ B obj;
  obj.person()
  return 0;
}
```

Good night

```
int main()
{
  A *p;
  B obj;
  p = &obj;
  p->person();
  return 0;
}
```

Good night
Good morning

class A
{ public:
  void person()
  {
    cout<<"Good morning"<<endl;
  }
};
class B: public A
{
  public:
  void person()
  {
    cout<<"Good night";
  }
};
int main()
{ B obj;
  obj.person();
  obj.A::person();
  return 0;
}

Good night
Good morning

class A
{ public:
  virtual void person()
```
```

normal pure virtual . templat

Abstract class:- are such classes that you are defined to inherit only by other classes. The purpose of abstract classes is to provide a abstract structure to other classes which you can inherit

Note:(1) We can't create obj for abstract classes
(2) A class which contain at least 1 pure virtual fu called abstract class.

Syntax:- virtual return-type function-name() = 0;

```
#include <iostream>
class animal
{
    void void sound() = 0;
};
class dog : public animal
{
    public: voidsoud()
    {
        cout << " woof woof..";
};
int main()
{
    dog d;
    d.soud();
    return 0;
}
```

woof woof..

```
class animal
{ public:
    virtual void sound = 0;
    void eat()
    {
        cout << "animal eating..";
    }
};
class dog: public animal
{
    public:
    void sound()
    {
        cout << "woof woof..." << endl.
    }
};
int main()
{
    dog d;
    d.sound();
    d.eat();
    return 0;
}
```

woof woof...
animal eating

Templat:- frame which defines its actual meaning in a C++ programming. The main purpose of templat is to "it accept any type of value" at the time of program execution.
we can use templat in C++ by two ways
(i) function templat
(ii) class Templat.

function templat also known as generic function. A normal f* works only one type of value at a time but "function templat" work with diffent-different type at a time

Syntax:- templat < class type >
return type functi name (parameter-list)
{
    //code;
}

WAP to show the example of fun template?

```
#include<iostream>
using namespace std;
template<class A>
void print(A x, A y)
{ cout<<x<<endl<<y<<endl;
}
int main()
{ print(10,20);
  print('a','b');
  print(45.3,676.7);
  return 0;
}
```

Class template: also known as generic class. We use class template when user does not know what kind of value to pass from the parameter.

Syntax:- template<class type>
class class-name
{
// body
};

WAP to show the example of class template?

```
#include<iostream>
using namespace std;
template<class A>
{ public:
    print(A x, A y)
    { cout<<x<<endl<<y<<endl;
    }
};
int main()
{ print<int>obj(100,200); //<char obj('x','1')
  return 0;                //<double>obj(45.4, 346.4)
}
```

Mainspace

```
#include<iostream>
int main()
{ std::cout<<"learn coding";
  return 0;
}
```

assignment operator

```
#include<...
namespace A
{ int a;
  void print()
  { a=10;
    cout<<a;
  }
}
int main()
{ A::print();
  B::print();
  return 0;
}
```

1020

namespace B
a=20
std::