C++ Full tutorial

- **Rakesh**

 https://www.linkedin.com/in/rakesh-k-9ba214300

 rakeshofficial123456@gmail.com

 _ _rakesh_ gowda_

Features

1. **Simple**
2. **High level language**
3. **Case sensitive**
4. **Object oriented language**
5. **General purpose programming language**

   it is used to create system software as well as application software

   ex: OS, drivers, GUI apps ,business applications, compilers ,databases like oracle my SQL.

6. **Machine independent Platform dependent**
7. **Faster**
   Compiled language
   No automatic garbage collector
8. **Middle level language**
   It's the combination of both low level language application as well as high level language
9. **Extensible language**
   we can add new user defined features into this program

Application of C++

- Operating system and system programming linux-based OS (Ubuntu)
- Browsers (chrome & firefox)
- Database engines (Mysql, MangoDB)
- Cloud systems
- Graphics and game engines

History

Bjarne Stroustrup
Initially known as "C with classes"
Renamed as C++ in 1983 → "one higher than c"

Extension of C++ files

.cpp , .cxx , .cc

Difference between C and C++

First Program in C++

```cpp
//this is preprocessor directive tells the compiler to include
standard input and output library function
#include<iostream>
// std  is used to define the functions that are used in the program
//  cin and cout are declared based on std:: iostream instances
using namespace std;

// entry point of a program
int main(){
    // instance of std:: iostream class
    cout<<"welcome to c++";
    // indicates finishing of a function
    return 0;

}
```

Datatypes

Primitive datatypes

1. **Integer**
2. **Float**
3. **Character**
4. **Boolean**

Derived datatypes

1. **Function**
2. **Array**
3. **Pointer**
4. **Reference**

User-defined datatypes

1. **Class**
2. **Structure**
3. **Union**
4. **Enum**

1. **Int**
   Size = 4 bytes → 32 bits of memory allocated to store

2. **Float**
   Size = 4 bytes → 32 bits of memory allocated to store
   We can store up to 7 decimal digits

3. **DOUBLE**
   size =8 bytes → 64 bits of memory allocated to store
   we can store upto 15 decimal digits

4. **Char**

   Size = 1 byte → 8 bits of memory allocated to store

   Store the character based on the ascii value

   65 for 'A' and 97 for 'a'

   Int asci= 'a';

   Cout<< asci; // 97

5. **Boolean**

   Stores Boolean values

   True -1

   False -0

   Size = 1 byte

Code

```
#include<iostream>
Using namespace std;
Int main(){

Int a ;
a=12;
cout<< "size of int "<<sizeof(a)<<endl;

float b;
cout<< "size of float "<<sizeof(b)<<endl;

char c;
cout<< "size of char "<<sizeof(c)<<endl;

bool d;
cout<< "size of bool "<<sizeof(d)<<endl;

return 0;
}
```

```
Size of int 4
Size of float 4
```

Size of char 1
Size of bool 1

Type modifier

Used to modify the size occupied by the datatypes

Modifier	Storage Size	Range of Values
unsigned int	4 bytes	0 to 4,294,967,295
short int	2 bytes	-32,768 to 32,767
signed int	4 bytes	-2,147,483,648 to 2,147,483,647
long int	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Short int si; // 2
Long int li; // 8

Type conversion in C++

A type cast is basically a conversion from one type to another. There are two types of type conversion:

1. **Implicit Type Conversion Also known as 'automatic type conversion'.**
   - **Done by the compiler on its own, without any external trigger from the user.**
     .
   - **All the data types of the variables are upgraded to the data type of the variable with largest data type.**
   - `bool -> char -> short int -> int ->`
   - `float -> double`

Example of Type Implicit Conversion:

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

int main()
{
   int x = 10; // integer x
   char y = 'a'; // character c

   // y implicitly converted to int. ASCII   value of 'a' is 97
      x = x + y;

   // x is implicitly converted to float
   float z = x + 1.0;

   cout << "x = " << x << endl
       << "y = " << y << endl
       << "z = " << z << endl;

     return 0;
}
```

Output:

```
x = 107

y = a

z = 108
```

2. Explicit Type Conversion:

This process is also called type casting and it is user-defined. Here the user can typecast the result to make it of a particular data type. This can be also considered as forceful casting. Syntax:

```
(type) expression
```

where *type* indicates the data type to which the final result is converted.

Example:

```
// C++ program to demonstrate  explicit type casting

#include <iostream>
using namespace std;

int main()
{
   double x = 1.2;

   // Explicit conversion from double to int
   int sum = (int)x + 1;

   cout << "Sum = " << sum;

   return 0;
}
```

Output:

Sum = 2

Operators

Arithmetic operator

Binary operator

+ - * / %

Unary operator

++ --

Code:

```
Int i=1;
       1    +   3
i    =i++  + ++I;
cout<<i<<endl;
```

Output :: 4

```
Int i=1;
Int j=2;
Int k;
     1  2    1to2          2to3     3          4
K=  I + j  +  i++   + j++  +   ++i + ++j;
Cout<<k;
```

Output :: 13

```
Int i=0;
   0   -  0   +  1  -    1    =0
i=i++  - --i   +   ++I  - i--;
cout<<i;
```

output=0;

```
int i=1,j=2,k=3;
int m =  i--   -   j--   -  k--;
cout<<i<<endl;
cout<<j<<endl;
cout<<k<<endl;
```

op::   0

1
2
-4

Int i=10, j=20 , k;

10 9 19 20 9 20 10 19 j=20 I =10
K= i-- - i++ + --j - ++j + --i - j-- + ++i - j++;

Op ::
I=10
J=20
K = -20

Relational Operator

== , != , > , < , >= , <=

q) input two numbers and check the relation b/w them

Logical operators

&& , || ,!

!0 = true
5 &&1 true
5 &&0 false

Write a program to check whether a number is devisible by both 2 and 3 or devisible by only one of them

```
Int n;
Cin>>n;
If(n%2==0 &&n%3==0){
Cout<<"devisible by both"<<endl;
}
And continue…….
```

Bitwise Operator

& bitwise AND
| bitwise OR
^ (XOR) same =0 and different =1 1 1 → 0 -- 0 1→ 1
~ (ones complement)
<< left shift
>> right shift

&	|	^	~	<<	>>
0101	0101	0101		4<<1	4>>1
0110	0110	0110	0101	(0100)	(0100)
0100	0111	1100	1010	1000	0010

a<<n → a*2 pow n
a>>n → a/ 2 pow n

Assignment operator

= , += ,-= , *= , /= , %=

Miscellaneous operator

sizeof() → return the size of the variable

condition ? → x:y ternary operator

cast operator () → char ch = 'a' , cout<<(int)ch;

comma (,) → **causes a sequence of operation to be performed**

a=(2,3,4) a is stores last value 4

& → **return the address of the variable (in mem loc)**

Int a ; &(a)

(*) → **pointer to a variable**

Int a ;

Int * b =&a; b is pointing to the mem loc where a is stored

Addition of two numbers

```
#include <iostream>
using namespace std;
int main(){
    cout<<"hello world"<<endl;
    int num1,num2;
    cout<<"enter the numbers";
    cin>>num1;
    cin>>num2;
    int sum=num1+num2;
    cout<<"sum of two number "<<num1<<" "<<num2<<" is "<<sum<<endl;
    return 0;
}
```

Conditional Statements

If

If else

If else if

Nested if

Greatest of three number

```
#include <iostream>
```

```
using namespace std;
int main(){
    cout<<"start"<<endl;
    int a,b,c;
    cout<<"enter the value of a,b and c"<<endl;
    cin>>a>>b>>c;

    if(a>b){
        if(a>c){
            cout<<a<<" is greater"<<endl;
        }
        else{
            cout<<c<<" is greater"<<endl;
        }
    }
    else{
        if(b>c){
        cout<<b<<" is greater "<<endl;
    }

    else{
        cout<<c<<" is greater"<<endl;
    }
    }
    cout<<"end";

}
```

2Q even or odd

Loops in C++

for
while
do while

Arrays

List of variables of similar type

Array elements accessed with the help of index position

Initialization of Array in C++

```
//   Initialize Array

int main(){
    int arr[5] = {1,2,3,4,5};
```

```
//   Initialize Array with Values and without Size in C++

    int arr[]={1,2};// 1 2
```

```
// Initialize an array partially in C++
// remaining spaces filled with zeros

int main(){
```

```
    int arr[4]={1,2};// 1 2 0 0
```

```
//   Initialize Array with 0 it is like an empty array

int main(){
    int arr[5] = {0};//0 0 0 0 0
```

```
// printing array elements using for loop

#include<iostream>
#include<string>
using namespace std;
int main(){
    int arr[4];
    arr[0]=12;
    arr[1]=13;
    arr[2]=22;
    arr[3]=34;

    cout<<sizeof(arr)<<endl;
    cout<<sizeof(arr[0])<<"\n";
      for(int i=0;i<sizeof(arr)/sizeof(arr[0]);i++){
        cout<<arr[i]<<" ";
    }
}
```

Dynamic input from the user to an array

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

```
int main(){

    int n;
    cout<<"enter the size of the array"<<endl;
    cin>>n;

    int arr[n];
    cout<<"enter the array elements "<<"\n";
    for(int i=0;i<sizeof(arr)/sizeof(arr[0]);i++){
        cin>>arr[i];
    }
    cout<<"the entered array elements are "<<endl;

    for(int i=0;i<sizeof(arr)/sizeof(arr[0]);i++){
        cout<<arr[i]<<" ";
    }
}
```

To find smallest and largest element in an array

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


int main(){

    int n;
    cout<<"enter the size of the array"<<endl;
    cin>>n;

    int arr[n];
    cout<<"enter the array elements "<<"\n";
    for(int i=0;i<sizeof(arr)/sizeof(arr[0]);i++){
        cin>>arr[i];
    }
    cout<<"the entered array elements are "<<endl;
     for(int i=0;i<sizeof(arr)/sizeof(arr[0]);i++){
        cout<<arr[i]<<" ";
     }
      int minN=INT_MAX;
    int maxN=INT_MIN;
    for(int i=0;i<sizeof(arr)/sizeof(arr[0]);i++){
     // max
```

```
        if(arr[i]<minN){
            minN=arr[i];
        }
        if(arr[i]>maxN){
            maxN=arr[i];
        }
    }
   cout<<endl<<"smallest number in the array is
"<<minN<<endl;
   cout<<"largest number in the array is "<<maxN;
return 0;
}
 minN=min(minN,arr[i]);
 maxN=max(maxN,arr[i]);
```

Break and continue Statement in C++

➔ To control the flow of loops

continue ::

used to skip to the next iteration

Break ::

Used to terminate the loop

ex::

to print odd numbers from 1 to 10

```
int main(){
for(int i=1;i<=20;i++){

  if(i>10){
```

```
    break;
    }

  if(i%2==0){
    continue;
  }
cout<<i<< " " ;
}

Return 0;
}
```

Print the numbers from 1 to 50 except the numbers that are devisible by 5

```
Int main(){
for(int i=0;i<=50;i++){
  if(i%5==0){
    continue;
}
Cout<<i<< " ";

}
return0;
}
```

Check prime Number or not

```
int main() {
int   n;
cin>>n;
int i;
for(int i=2 ;i<n;i++){
    if(n%i==0){
        cout<< "non prime"<<endl;
        break;
   }
if(i==n){
  cout<< "prime "<<endl;
```

```
}
return 0;
}
```

Print all the prime number between a and b

```
Int main(){
Int a , b;

//outer for loop to traverse from a to b
// inner loop check the number is prime or not
for (int num= a ;num<=b;num++){
        int I;
    for(int i=2;i<=num;i++){
        if(num%i==0){
                break;
        }
  }
If(i==num){
    cout<<num<< " ";
}
return 0;
}
```

2D Arrays

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

int main(){

    int n,m;
    cout<<"enter the number of rows "<<endl;
    cin>>n;
    cout<<"enter the number of columns "<<endl;
    cin>>m;

    int arr[n][m];

    for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            cin>>arr[i][j];
        }
    }
     for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            cout<<arr[i][j]<<" ";
        }
        cout<<endl;
    }
    return 0;
    }
```

Searching in 2D arrays

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

int main(){

    int n,m;
    cout<<"enter the number of rows "<<endl;
    cin>>n;
    cout<<"enter the number of columns "<<endl;
    cin>>m;

    int arr[n][m];

    for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            cin>>arr[i][j];
        }
    }
     for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            cout<<arr[i][j]<<" ";
        }
        cout<<endl;
    }

    int x;
    bool flag=false;
    cout<<"enter the element to be searched in the matrix";
    cin>>x;
      for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            if(arr[i][j]==x){
                flag=true;
                cout<<i<<" "<<j;

            }
        }
      }
    if(flag){
        cout<<"element found";
    }
    else{
        cout<<"element not found ";
    }
    }
```

Character array

==========

Declaration

Char arr[n+1];
“Tom/0”
/0 → indicates the word is ended
This I the last character of a sentence

Program to print all the character in the character array

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

int main(){

    char arr[100]="apple";
    int i=0;

    while(arr[i]!='\0'){
        cout<<arr[i]<<endl;
        i++;
    }
    return 0;
    }
```

Palindrome of a string

==========

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

int main(){
  // input total number of charecter in a string
   int n;
   cin>>n;

   char arr[n+1];
   cin>>arr;

   bool isPalindrome= true;

   for(int i=0;i<n;i++){
    if(arr[i] !=arr[n-1-i]){
        isPalindrome=false;
        break;
    }
   }
   if(isPalindrome){
    cout<<"word is a palindrome"<<endl;
   }
   else{
    cout<<"word is not a palindrome"<<endl;
   }

    return 0;
    }
```

Longest word in a sentence

==========

Length

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

int main(){
  // input total number of charecter in a string
   int n;
   cin>>n;
   cin.ignore();

   char arr[n+1];
   cin.getline(arr,n);
   cin.ignore();// to clear the buffer

//    declare 2 variable to track length of the word untill we get
empty
//    space agin we initialize currLength to zero to find length of
the next word
//    and store the length of the longest word in maxLength.
//    maxLength is updating as we get the word of largest length

    int i=0;
    int currLen=0,maxLen=0;

    // iterate using while loop
    while(1){
        if(arr[i]==' ' || arr[i]=='\0'){
            if(currLen>maxLen){
                maxLen=currLen;
            }
            currLen=0;
        }
        else{
            currLen++;
        }
        if(arr[i]=='\0'){
            break;
        }
        i++;
    }
    cout<<maxLen<<endl;
    return 0;
    }
```

Note:

- Why we need to use cin.ignore() before and after cin.getline()
- When a user input something with cin they hit enter and '\n' character gets into the cin buffer.
- Then if you use getline() it gets the newline char instead of string we entered

Word with maxlength

```
#include<iostream>
using namespace std;
int main(){
  // input total number of charecter in a string
   int n;
   cin>>n;
   cin.ignore();
   char arr[n+1];
   cin.getline(arr,n);
   cin.ignore();// to clear rhe buffer
        int i=0;
    int currLen=0,maxLen=0;
    int start=0,endIndex=0;// start is track the word similar to currLen
    // iterate using while loop
    while(1){
        if(arr[i]==' ' || arr[i]=='\0'){
            if(currLen>maxLen){
                maxLen=currLen;
                endIndex=start;// because start is reaching the end of the
word next space next another word
            }
     currLen=0;
     start=i+1;//because space is present and one index after word is
begining
        }
        else{
            currLen++;
        }
        if(arr[i]=='\0'){
            break;
        }
        i++;
    }
    cout<<maxLen<<endl;
    for(int i=0;i<maxLen;i++){
        cout<<arr[i+endIndex];
    }
    return 0;
    }
```

Pointers in C++

1. Pointers in C++ are variables that hold memory addresses of other variables.

2. Pointers are declared by using the * symbol before the variable name. For example:

   int *ptr;

3. This declares a pointer variable named ptr that can point to an integer variable.

4. Pointers are used to access the value stored at the memory address they point to using the * operator. For example:

```
int x = 10;
int *ptr = &x;
cout << *ptr; // this will output 10
```

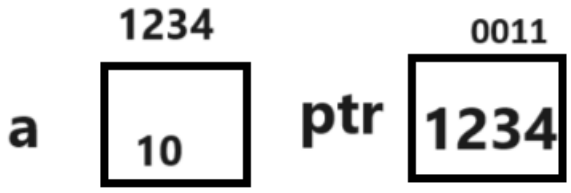
5. Pointers can also be used to dynamically allocate memory using the new keyword. For example:

```
int *ptr = new int;
*ptr = 20;
```

6. This dynamically allocates memory on the heap for an integer variable and stores the value 20 at that memory address.

7. It is important to note that pointers provide direct access to memory addresses and can be unsafe if not used correctly.

Pointer is the variable that holds the memory address of the another variable



Int a =10;

Int *ptr=&a;

```
#include<iostream>
using namespace std;
int main(){
    int a=20;
    int *ptr=&a;

    cout<<a<<endl;
    cout<<&a<<endl;
    cout<<ptr<<endl;
    cout<<*ptr<<endl; // value of pointer
}
```

Output

0

20

0x5ffe94

0x5ffe94

20

0x5ffe90

& →address of the operator
* →value at address operator

** → pointer to pointer

Pointer to pointer

```cpp
#include<iostream>
using namespace std;
int main(){
    int a=20;
    int *ptr=&a;

    // cout<<a<<endl;
    // cout<<&a<<endl;
    cout<<ptr<<endl;
    // cout<<*ptr<<endl;

    //address of the pointer
    cout<<&ptr<<endl;

    // pinter to pointer
    int **q=&ptr;
    cout<<q<<endl; // adress of p
    cout<<*q<<endl; //adress of a
    cout<<**q<<endl; // value
    cout<<&q<<endl;  // address of q
}
```

Output:

```
0x5ffe9c
0x5ffe90
0x5ffe90
0x5ffe9c
20
0x5ffe88
```

Analysis of size of pointer

```cpp
#include<iostream>
using namespace std;
int main(){
    int a=20;
    // & is address of operator
    cout<<"address of a is "<<&a;
    int *ptr=&a;

    cout<<"address of a is : "<<ptr<<endl;
    cout<<"value of a is   : "<<*ptr<<endl;

    double b=5.4;
    double *p2=&b;

    cout<<"address of b is  : "<<p2<<endl;
    cout<<"value of b is    : "<<*p2<<endl;

   // analysis on size of pointer
   cout<<"size of integer is     : "<< sizeof(a)<<endl;
   cout<<"size of pointer ptr is : "<<sizeof(ptr)<<endl;

   cout<<"size of double     : "<<sizeof(b)<<endl;
   cout<<"size of pointer p2 : "<<sizeof(p2)<<endl;
}
```

Output:

address of a is 0x5ffe8c
address of a is : 0x5ffe8c
value of a is   : 20
address of b is : 0x5ffe80
value of b is   : 5.4
size of integer is   : 4
size of pointer ptr is : 8
size of double   : 8
size of pointer p2 : 8

Note:

If a pointer is just declared not initialized to any value, then it is pointed to some garbage value. This is not a good practice

2 ways of declaring pointer

```
#include<iostream>
using namespace std;
int main(){
    int i=5;
    //case1
    int *ptr=0;
    ptr=&i;

    cout<<"address of pointer case1 : "<<ptr<<endl;
    cout<<"value of pointer         : "<<*ptr<<endl;

    //case2
    int *p2=&i;
    cout<<"address of pointer case2 : "<<ptr<<endl;
    cout<<"value of pointer         : "<<*ptr<<endl;
}
```

Output:

```
address of pointer case1 : 0xabaf1ff8ac
value of pointer         : 5
address of pointer case2 : 0xabaf1ff8ac
value of pointer         : 5
```

Incrementing a pointer

```cpp
#include<iostream>
using namespace std;
int main(){
    int num=10;
    int a =num;
    a++;

    cout<<num<<endl;

    int *p=&num;
    cout<<"before incrementing "<<num<<endl;
    (*p)++;
    cout<<"after               "<<num<<endl;
}
```

Output:

10
before incrementing 10
after 11

copying of pointer

==========

```cpp
#include<iostream>
using namespace std;
int main(){
  int i=5;
  int *p=&i;

  int *q=p;
  cout<<p<<"====> "<<q<<endl;    //0x5ffe8c====> 0x5ffe8c
  cout<<*p<<"====> "<<*q<<endl;  //5====> 5
}
```

// pointer arithmetic

```
#include<iostream>
using namespace std;
int main(){
  int i=5;
  int *p=&i;
  cout<<"  i before " <<*p<<endl;
 *p=*p+1;// (*p)++;
  cout<<"  i after " <<*p<<endl;
  cout<<"p before "<<p<<endl;
  p=p+1;
  cout<<"p after "<<p<<endl;
}

//output
//   i before 5
//   i after 6
// p before 0x2f647ffd44
// p after 0x2f647ffd48

Note:
When we try to increment the t (memory address)
It will incremented to next int size ie incremented by 4
bytes (44 → 48)
```

Pointers with arrays

==========

Note:

Arr[i]=*(arr+i)

i[arr]=*(i+arr)

Address of the first memory block is a array name itself

```
#include<iostream>
using namespace std;
int main(){
    int arr[10];

    // address of the first memory block is array name
itself
    cout<<"address of first memory block
is  ::"<<arr<<endl;
    //  address of first memory block is  ::0x5ffe70
    // now we check and compare with the first element
    cout<<"address of first memory block
is  ::"<<&arr[0]<<endl;
    // address of first memory block is  ::0x5ffe70

    int arr1[10]={0};
    cout<<"address of first memory block
is  ::"<<arr1<<endl;
}
```

*arr

```
// pointer to acess the element inside the array
// ie value at the zeron

#include<iostream>
using namespace std;
int main(){
     int arr[10]={2,4,6,8};
```

```
  cout<<*arr<<endl;

}
```

How to access the element at the first index

```
#include<iostream>
using namespace std;
int main(){
     int arr[10]={2,9,6,8};
  cout<<*arr<<endl;
  // when we try to add 1 to *arr directly without
paranthesis it add value at zeroth idx +1
  // ie arr[0]+1
    cout<<*arr+1<<endl;

    // if we want value at  index 1 we need to use
paranthesis *(arr+1)
    cout<< *(arr+1)<<endl;// it will move on to the next
memory laction by moving 4 bytes forward and print the
value
}
```

Note:

Arr[i]=*(arr+i)

i[arr]=*(i+arr)

```
int i=3;
    cout<<i[arr]<<endl; //8
```

Difference between array and pointer

```
#include<iostream>
using namespace std;
int main(){
   int a[20]={1,2,3,4};
   //whe we print a , &a or &a[0] it will give only memory
address
  cout<<a<<endl;
  //  a=a+1; // we get error we cant change the memory
address in case of arrays
   int *p=&a[0];
   cout<<p<<endl;
   p=p+1;// we can change the memory address in pointer
   cout<<p<<endl;
}
```

Pointer for character array

==========

In case of integer array when we try to print array name it will give the address of the first index

But in case of character array, it will give the actual content.

```
#include<iostream>
using namespace std;
int main(){
   int a[20]={1,2,3,4};
   //whe we print a , &a or &a[0] it will give only memory address
  cout<<a<<endl;// 0x5ffe50
  cout<<&a[0]<<endl;  // 0x5ffe50
   char ch[7]= "jerry";
   cout<<ch<<endl; // jerry

   // declare a char pointer and store address of array
   char *c=&ch[0];
   // we get full character instead of address while printing the pointer
   cout<<c<<endl; //jerry
   cout<<*c<<endl;// j
```

Functions in C++

Set of instruction to perform some task

- To avoid code redundancy

Syntax of a function

returnType functionName (parameter1 , parameter2……..){

}

Wtite a function to add 2 numbers

```
Int add ( int num1, int num2)
{
Int sum = num1+num2;
return sum;
}
// execution

Int main(){
Int a,b;
Cin>>a>>b;
Cout<<add(a,b)<<endl;
return 0;
}
```

1. **Only names of variables are passed and not their values and not their types while calling the functions**
2. **We pass the values, not variables themselves. Local variables are created in functions which are destroyed on returning from them**
3. **A function can be called from any other function or main function**

EXAMPLE : : Execution happens in the memory stack

```
void print (int num){
cout<<num<< endl;
return;
}

Int add(int num1,int num2){
```

```
print(num1);
print(num2);
int sum=num1+num2;
return sum;
)
Int main(){
Int a=2;
Int b=3;
cout<<add(a,b)<<endl;
return 0;
}
```

Pointer with variable with function

Passing pointer to a function

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

void print(int *p){
   cout<<p<<endl; // memeory address
   cout<<*p<<endl;// 5
}

    void update(int *p){
       // p=p+1;
       *p+=1;
    }
    int main(){
       int value=5;
       int *p=&value;
       print(p);
       cout<<"before "<<p<<endl;
       cout<<"before "<<*p<<endl;
       update(p);
       cout<<"after "<<p<<endl;
       cout<<"after "<<*p<<endl;
    }
```

Output:

0x5ffe94

5

before 0x5ffe94

before 5

after 0x5ffe94

after 6

Pointer array with function

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

void print(int *p){
   cout<<p<<endl; // memeory address
   cout<<*p<<endl;// 5
}
// A function takes array as a pointer argument
int getSum(int arr[],int n){ // internally getSUm(int
* arr,int n)
   cout<<sizeof(arr)<<endl;// it return the size of
pointer instead of size of the array
   int sum =0;
   for(int i=0;i<n;i++){
      sum+=arr[i];// i[arr] --> both are same
   }
   return sum;
}
int main(){
   int arr[5]={1,2,3,4,5};
   cout<<"sum is  : "<< getSum(arr,5)<<endl;
}
```

What is the advantage if a function takes array as pointer

We can pass part of array to get the sum

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

void print(int *p){
   cout<<p<<endl; // memeory address
   cout<<*p<<endl;// 5
}
// A function takes array as pointer argument
int getSum(int arr[],int n){ // internally getSUm(int
*arr,int n)
   cout<<endl<<"size :: "<<sizeof(arr)<<endl;// it return
the size of pointer
   int sum =0;
   for(int i=0;i<n;i++){
      sum+=arr[i];// i[arr] --> both are same
   }
   return sum;
}
int main(){
   int arr[5]={1,2,3,4,5};
   cout<< getSum(arr+3,5-3)<<"<-- Sum"<<endl; //n-3 if we
declare a array of size n
}
```

Pointer to pointer concept

int i=5;

int *ptr1 =&I;

int **ptr2=&ptr;

symbol table

I (710) → 5;

Ptr1 (810) →710

Ptr2 (910) →810

```
#include<iostream>
using namespace std;
int main(){
 int i=5;
 int *ptr1=&i;
 int **ptr2=&ptr1;

 cout<<"printing the values of i in different way "<<endl;
 cout<<"i :: "<<i<<" *ptr1 :: "<<*ptr1<<" **ptr2 ::
"<<**ptr2<<endl;

 cout<<"printing the address of i "<<endl;
 cout<<"&i --> "<<&i<<" ptr1 --> "<<ptr1<<" *ptr2-->
"<<*ptr2<<endl;

 cout<<"printing the address of ptr1 "<<endl;
 cout<<" &ptr1--> "<<&ptr1<<" ptr2--> "<<ptr2<<endl;

 cout<<"address of ptr2 or double pointer "<<endl;
 cout<<" &ptr2--> "<<&ptr2<<endl;
}
```

Output:

printing the values of i in different way

i :: 5 *ptr1 :: 5 **ptr2 :: 5

printing the address of i

&i --> 0x5ffe9c ptr1 --> 0x5ffe9c *ptr2--> 0x5ffe9c

printing the address of ptr1

&ptr1--> 0x5ffe90 ptr2--> 0x5ffe90

address of ptr2 or double pointer

&ptr2--> 0x5ffe88

Changes after Updating the double pointer in different manner

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

void update(int **p2){
   // case 1
// p2=p2+1;  // no changes in i,p1,p2 -)) i --> 5 p1--
>  0x5ffe94 p2--> 0x5ffe88
// *p2=*p2+1;   // changes in p1 value  p1-->  0x5ffe94
to  p1-->  0x5ffe98
// **p2+=1; // changes in the value of i  --)) i --> 6 p1-
->  0x5ffe94 p2--> 0x5ffe88
}
int main(){
 int i=5;
 int *p1=&i;
 int **p2=&p1;

cout<<"before"<<endl;
cout<<"i --> "<<i<<" p1-->  "<<p1<<" p2--> "<<p2<<endl;
update(p2);
cout<<"after"<<endl;
cout<<"i --> "<<i<<" p1-->  "<<p1<<" p2--> "<<p2<<endl;
}
```

Problems on pointers

1.

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

int main(){
int num1=12;
int num2=14;
int *ptr=&num1;
*ptr=5;
cout<<num1<<"  "<<num2<<endl;  // 5  14
}
```

2.

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

int main(){
int num1=12;
int *p=&num1;
int *q=p;
cout<<"address  &num1 "<<&num1<<" p-->"<<p<<" q--
>"<<q<<endl;
//address  &num1 0x5ffe8c p-->0x5ffe8c q-->0x5ffe8c

cout<<"before incrementing -->"<<*q<<endl; //12
(*q)++;
cout<<num1<<endl; //13
}
```

3. pointer based on pre and post increment

Post-Increment

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

int main(){
int num1=12;
int *p=&num1;
cout<<(*p)++<<endl; // 12// post increment first p get
printed
                    // then *p incremented by one
cout<<num1<<endl;   // 13
}
```

Pre-Increment

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

int main(){
int num1=12;
int *p=&num1;
cout<<++(*p)<<endl; // 13  // pre  increment first p get incremented
by one
                   // then *p get assigned
cout<<num1<<endl;   // 13
}
```

4. Null pointer

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

int main(){
int num1=12;
int *p=0;
p=&num1; // *p=num1; segmentation fault
cout<<p<<endl;  //0x5ffe94
cout<<*p<<endl; //12

}
```

5.

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

int main(){
int first=15;
int second=25;
int *third=&second;
first=*third;
*third+=2;
cout<<first<<" "<<second<<endl; // 25 27

}
```

6.

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

int main(){
float f= 22.5;
float g=15.5;
float *ptr=&f;
(*ptr)++;
*ptr=g;
cout<<*ptr<<" "<<f<<" "<<g<<endl; //15.5 15.5 15.5

}
```

7. Pointer's problems on arrays

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

int main(){
int arr[5];
int *p;
cout<<sizeof(arr)<<" "<<sizeof(p)<<endl; // 20 8

int arr1[]={1,2,3,4};
cout<<*(arr1)<<" "<<*(arr1+1)<<endl; // 1 2

int arr2[]={12,24,36};
cout<<arr2<<"  "<<&arr2[0]<<"  "<<&arr2<<endl;
// 0x5ffe64  0x5ffe64  0x5ffe64
cout<<(arr2+1)<<endl; // 0x5ffe68
}
```

8.

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

int main(){
int arr[6]={4, 8, 12};
int *p=arr;
cout<<p[2]<<endl;//12
cout<<p<<" "<<&arr<<endl;//0x5ffe80  0x5ffe80
cout<<*p<<endl;//4
}
```

9.

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

int main(){
int arr[6]={4, 8, 12};

cout<<*(arr)<<*(arr+3)<<endl; //4 0
}
```

10. We cant increment the array

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

int main(){
int arr[]={4, 8, 12};
int *ptr= arr++; // error we cant increment the array we can pointer
cout<<*ptr<<endl;
}
```

11.Char variable

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

int main(){
    char ch='a';
    char *ptr=&ch;
    ch++;
    cout<<*ptr<<endl; //b
}
```

12.char array vs int array pointer

```
using namespace std;

int main(){
   char charArray[]="hello";
   int intArr[10]={1,3,5,7,9};
   char *c=&charArray[0];
   int *i=&intArr[0];
// in case of char array it will print the whole char
array
// in case of integer array it will print the address of
the first index
   cout<<c<<"   "<<i<<endl; // hello   0x5ffe60


}
```

13.To print char array without first element

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

int main(){
    char arr[]="hello";
    char *p=&arr[0];
    p++;
    cout<<p<<endl; // ello
    return 0;
}
```

14. to print first element

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

int main(){
    char str[]="hello";
    char *p=str;
    cout<<p[0]<<" "<<str[0]<<endl; // h h
    return 0;
}
```

15.

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

void update(int *p){
    *p=(*p)*2;
}
int main(){
  int i=10;
  update(&i);
  cout<<i<<endl; // 20;
}
```

1. based on double pointer

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


int main(){
    int first=45;
    int *p=&first;
    int **q=&p;
    int second =(**q)++ +9;
    cout<<first<<"  "<<second<<endl; // 46  54

}
```

2.

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


int main(){
    int first=50;
    int *p=&first;
    int **q=&p;
    int second =++(**q);
    int *r=*q;
    ++(*r);
    cout<<first<<"  "<<second<<endl; //52 51

}
```

3.

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

void increment(int **ptr){
    ++(**ptr);
}

int main(){
    int first=50;
    int *p=&first;
   increment(&p);
   cout<<first<<endl;

}
```

Inputs from input file to display result in output file

```
// to get input from input file and display output in
output file
#include<iostream>
using namespace std;

int main(){
    #ifndef ONLINE_JUDGE
        freopen("input.txt","r",stdin);
        freopen("output.txt","w",stdout);
    #endif

int a,b;
    cin>>a>>b;
    cout<<a+b;
    return 0;

}
```

Strings in C++

Input and output a string

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


int main(){
   string str;
   cout<<"enter the string"<<endl;
   cin>>str;
   cout<<"Name of the student is "<<str<<endl;

}
```

How to declare a string in different ways in C++

```
1.
#include<iostream>
#include<string>
using namespace std;
int main(){
   string str(10,'t');// ttttttttt
   cout<<str;
   return 0;
}
2.
#include<iostream>
#include<string>
using namespace std;

int main(){
   string str="hello";// hello
   cout<<str<<endl;
}
```

To accept full sentence from the user

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

// // to input single word
int main(){
//     string str;
//     cout<<"enter the sentence"<<endl;
//     cin>>str;
//     cout<<str<<endl;

// to accept sentence from the users
cout<<"enter str"<<endl;
string sentence;
getline(cin,sentence);
cout<<sentence<<endl;


}
```

Concatenation of string

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

int main(){
    string s1="tom ";
    string s2=" and  ";
    string s3="jerry";

    string newString=s1+s2+s3;
    cout<<newString<<endl;
}
```

Accessing each character in a string

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

int main(){
    string s1="tom and jerry";
    cout<<s1[6]<<endl; //d
    cout<<s1[0]<<endl; //t


}
```

Clear() → to clear string

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

int main(){
    string s1="tom and jerry";
    cout<<s1<<endl;
    s1.clear();
    cout<<s1<<endl;

}
```

Comparing two strings to

```cpp
#include<iostream>
#include<string>
```

```cpp
using namespace std;

int main(){
    string s1="tom";
    string s2="abc";
    cout<<s1.compare(s2)<<endl; // +1 lexicographically
greater
    cout<<s2.compare(s1)<<endl; // -1 lexicographically
smaller

    string s3="abc";
    string s4="abc";
    cout<<s3.compare(s4)<<endl; // 0


  if(!s3.compare(s4)){
        cout<<"strings are equal"<<endl;
    }
    else{
        cout<<"strings are not equal"<<endl;
    }

}
```

Empty function

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

```
int main(){
    string s1="tom";
    cout<<s1<<endl;
    cout<<s1.empty()<<endl; // 0
    s1.clear();
    cout<<"after clearing
isEmpty::"<<s1.empty()<<endl;//1

    if(s1.empty()){
        cout<<"string is empty"<<endl;
    }
    else{
        cout<<"string is not empty"<<endl;
    }


}
```

Erase function

```
#include<iostream>
#include<string>
using namespace std;
int main(){
    string s1="tomandjerry";
    s1.erase(2,5); // erase(begining index , no of
character)
    cout<<s1<<endl;

}
```

Find function

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

int main(){
    string s1="tomandjerry";
    // return begining index if the word exist
    cout<<s1.find("and")<<endl;//3
    cout<<s1.find("abc"); // some garbage value

}
```

Insert function

insert part of string into the original string at a given index

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


int main(){
    string s1="human";
    s1.insert(1,"aaa");// (beginingIndex,partofstr)
    cout<<s1<<endl;// haaauman
}
```

Length function (size() , length())

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


int main(){
    string s1="human";
    cout<<s1.size()<<endl;//5
    cout<<s1.length()<<endl;//5


}
```

Iterating each character of string using looping

```cpp
#include<iostream>
#include<string>
using namespace std;
int main(){
    string s1="human";
    for(int i=0;i<s1.length();i++){
        cout<<s1[i]<<" ";// h u m a n
    }
}
```

Substring of a string

```cpp
#include<iostream>
#include<string>
using namespace std;
int main(){
    string s1="human is a social animal";
    // substr(starting index, toatal num of character from
start index);
    cout<<s1.substr(4,4)<<endl; //n is
 }
```

Convert string to integer

```cpp
#include<iostream>
#include<string>
using namespace std;
int main(){
    string s="556";
    // converting string to integer
    int a = stoi(s);
    cout<<a+20<<endl;// 576

    // converting integer to string
    int c =242;
    cout<<to_string(c)+"2"<<endl;// 2422

}
```

Sorting a string

1. Import algorithm header file
2. Use sort function

```cpp
#include<iostream>
#include<algorithm>
#include<string>
using namespace std;
int main(){
    string s="qwerty";
    sort(s.begin(),s.end());
    cout<<s<<endl;//eqrtwy
}
```

Convert string to uppercase and lowercase

```
#include<iostream>
#include<algorithm>
#include<string>
using namespace std;
int main(){
    string s="qwerty";

   for(int i=0;i<s.size();i++){
    // convert the string to uppercase
        if(s[i]>='a' &&s[i]<='z'){
        s[i]-=32;
        }
    }
    cout<<s<<endl;

    for(int i=0;i<s.size();i++){
        if(s[i]>='A' &&s[i]<='z'){
            s[i]+=32;
        }
    }
     cout<<s<<endl;
}
```

With the help of inbuilt function

```
#include<iostream>
#include<algorithm>
#include<string>
using namespace std;
int main(){
    string s="qwerty";
    // toupper
    transform(s.begin(),s.end(),s.begin(),::toupper);
    cout<<s<<endl; //QWERTY
   // tolower
    transform(s.begin(),s.end(),s.begin(),::tolower);
    cout<<s<<endl;
}
```

Return max occurrence of character in string

abcacdade

a→3 times

```cpp
#include<iostream>
#include<algorithm>
#include<string>
using namespace std;
int main(){
    string s ="aaajdhehhdjehdkjwaffa";
    int freq[26];
    for(int i=0;i<26;i++)
        freq[i]=0;

    for(int i=0;i<s.size();i++){
        freq[s[i]-'a']++;
    }

    char ans ='a';
    int maxF=0;

    for(int i=0;i<26;i++){
       if(freq[i]>=maxF){
        maxF=freq[i];
        ans=i+'a';
       }
    }

    cout<<maxF<<" "<<ans<<endl;

}
```

Form a biggest number from the string

“634” →643

```cpp
#include<iostream>
#include<algorithm>
#include<string>
using namespace std;
int main(){
    string s="634251";
    sort(s.begin(),s.end(),greater<int>());
    cout<<s<<endl;
    sort(s.begin(),s.end());

}
```

Object oriented programming in C++

Class

Object

Object creation

Public access specifier

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

class Fruit{
    // we have to use public otherwise by defaultly it is
priovate
    //  and we can access the properties
    public:
    string color;
    string taste;

};
int main(){
    Fruit apple;
    apple.color="red";
    apple.taste="sweet";
    cout<<apple.color<<endl;
    cout<<apple.taste<<endl;


}
```

Object creation, constructors, this

```cpp
#include<iostream>
#include<algorithm>
#include<string>
using namespace std;

class Fruit{
    // we have to use public otherwise by defaultly it is
private
    //  and we can access the properties
    public:
    string color;
    string taste;

    public:
     Fruit(string color,string taste){
        this->color=color;
        this->taste=taste;
     }

    void display(){
        string color="orange";
        cout<<color<<endl;
        cout<<this->color<<endl;
        cout<<this->taste<<endl;
    }

};
int main(){

    cout<<"before reinitializing"<<endl;
    Fruit apple("green","sour");
    apple.display();
     cout<<"after  reinitializing"<<endl;
    apple.color="red";
    apple.taste="sweet";
    apple.display();


}
```

Object creation using new operator as pointer

Class-name *c =new Class-name();

Access using -> arrow operator

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

class Fruit{
    // we have to use public otherwise by defaultly it is
priovate
    //  and we can access the properties
    public:
    string color;
    string taste;
    public:
     Fruit(string color,string taste){
        this->color=color;
        this->taste=taste;
     }
    void display(){
        cout<<this->color<<endl;
        cout<<this->taste<<endl;
    }
};
int main(){

    cout<<"before reinitializing"<<endl;
     Fruit *apple =new Fruit("green","sour");
    apple->display();
     cout<<"after  reinitializing"<<endl;
    apple->color="red";
    apple->taste="sweet";
    apple->display();
    cout<<apple->color<<endl;// use arrow operator when we
use
}
```

Constructor overloading

```cpp
#include<iostream>
#include<algorithm>
#include<string>
using namespace std;

class Rectangle{
    public:
    int l;
    int b;

    // default constructor
    Rectangle(){
        l=0;
        b=0;
    }

    // parameterised constructor
    Rectangle(){
     // constructor overloading
    }
};
int main(){

    Rectangle r1;
    cout<<r1.b<<endl<<r1.l<<endl;
}
```

Default constructor

parameterized constructor

Copy constructor

```
#include<iostream>
#include<algorithm>
#include<string>
using namespace std;
class Rectangle{
    public:
    int l;
    int b;

    // default constructor
    Rectangle(){
        l=0;
        b=0;
    }

    // parameterised constructor
    Rectangle(int l,int b){
        this->b=b;
        this->l=l;
    }
    // // copy constructor
    Rectangle(Rectangle& r){
        l=r.l;
        b=r.b;
    }
};
int main(){
    Rectangle r1;
    cout<<r1.b<<endl<<r1.l<<endl;
    Rectangle r2(4,5);
    cout<<r2.b<<endl<<r2.l<<endl;
    Rectangle r3(r2);
    cout<<r3.b<<endl<<r3.l<<endl;
}
}
```

Destructor

Destructor is a function called when object deleted

We cannot pass any parameter inside the destructor

Destructor name is similar to class name

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

class Rectangle{
    public:
    int l;
    int b;

    // default constructor
    Rectangle(){
        l=0;
        b=0;
    }

    // parameterised constructor
    Rectangle(int l,int b){
        this->b=b;
        this->l=l;
    }

    // // copy constructor
    Rectangle(Rectangle& r){
        l=r.l;
        b=r.b;
    }
    ~Rectangle(){ // destructor function
    cout<<" destructor function is called"<<endl;
    }
};
```

```
int main(){

    Rectangle r1;
    cout<<r1.b<<endl<<r1.l<<endl;
    Rectangle r2(4,5);
    cout<<r2.b<<endl<<r2.l<<endl;
    Rectangle r3(r2);
    cout<<r3.b<<endl<<r3.l<<endl;
}

Output:

0
0
5
4
5
4
 destructor function is called
 destructor function is called
 destructor function is called
```

Note:

If we use new operator to create object we can use delete operator to delete the object and destructor function was called simultaneously.

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

class Rectangle{
    public:
    int l;
    int b;

    // default constructor
    Rectangle(){
```

```
        l=0;
        b=0;
    }

    // parameterised constructor
    Rectangle(int l,int b){
        this->b=b;
        this->l=l;
    }

    // // copy constructor
    Rectangle(Rectangle& r){
        l=r.l;
        b=r.b;
    }
    ~Rectangle(){ // destructor function
    cout<<" destructor function is called"<<endl;
    }
};
int main(){

    Rectangle *r1 = new Rectangle();
    cout<<r1->b<<endl<<r1->l<<endl;
    delete(r1);
    Rectangle r2(4,5);
    cout<<r2.b<<endl<<r2.l<<endl;
    Rectangle r3(r2);
    cout<<r3.b<<endl<<r3.l<<endl;
}

Output:

0
0
 destructor function is called
5
4
5
4
 destructor function is called
 destructor function is called
```

Encapsulation

Process of binding of methods and data members into a single entity/unit is called encapsulation

We can access the data within the class only through its method.

Advantage:

- Data abstraction[hiding]
  Hence class is a abstract datatype

```
class Student {
    int password; // password is private. Accessed within the class
only
};
int main(){
    Student s;
    cout<<s.password; // we cant directly access
```

```
class Student {
    int password;
    public:
    void setPassword(int p){
        password=p;
    }
     int getPassword(){
        return password;
     }
};
int main(){
    Student s;
    s.setPassword(1234);
    cout<<s.getPassword()<<endl;
}
Output: 1234
```

Abstraction

Process of hiding the implementation and showing only functionalities to the users

pow(x,y) → x^y

here we use pow function to find the power . but its implementation are hidden.

Similarly we use string function

Access Specifier and Inheritance

Public: Access everywhere in the code

Private: accessible within the class only

Protected: accessible within class and its child classes

Inheritance

Process of one class acquiring the properties of another class.

```cpp
#include<iostream>
#include<algorithm>
#include<string>
using namespace std;

class Parent {
    public:
    int x;

    protected:
    int y;

    private:
    int z;
};

class child1: public Parent{
    // x will remain public
    // y will remain procted
    // z will be inaccessible
};

class child2 :protected Parent{
    // x will be protected
    // y will be protected
    // z will be inaccessible
};
class child3 :private Parent{
    // x will be private
    // y will be private
    // z will be inaccessible
};


int main(){
    Parent p;
    p.x;
    // p.y; inaccessible as it is protected
    // p.z; inaccessible as it is private

return 0;
}
```

Types of inheritance

1.Single level inheritance

```
// Non parameterised constructor in super class

#include<iostream>
#include<algorithm>
#include<string>
using namespace std;

class Parent {
    public:
    Parent(){
        cout<<"parent class constructer"<<endl;
    }
};

class child: public Parent{
    public:
    child(){
        cout<<"child class constructor"<<endl;
    }
};

int main(){
   child c;

return 0;
}

Output:

parent class constructer
child class constructor
```

parameterized constructor in super class

```cpp
#include<iostream>
#include<algorithm>
#include<string>
using namespace std;

class Parent {
    public:

    Parent(int a){
        cout<<"parent class constructer with
value  "<<a<<endl;
    }
};

class child: public Parent{
    public:
    child():Parent(5){
    cout<<"child class constructor"<<endl;
    }
};

int main(){
   child c;

return 0;
}
```

2.Multilevel inheritance

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

class Parent {
    public:
    int x=33;
    Parent(int a){
        cout<<"parent class constructer with
value  "<<a<<endl;
    }
};

class child: public Parent{
    public:
    child():Parent(5){

        cout<<"child class constructor"<<endl;
    }
};

class grandChild: public child{
  public:
  grandChild(){
    cout<<"this is grand child"<<endl;
  }
};

int main(){
   grandChild c;
   cout<<c.x<<endl;

return 0;
}
```

Multiple Inheritance

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

class Parent1 {
    public:
    int x=22;
    Parent1(int a){
        cout<<"parent1 class constructer with
value  "<<a<<endl;
    }
};

class Parent2 {
    public:
    int y=33;
    Parent2(int a){
        cout<<"parent2 class constructer with
value  "<<a<<endl;
    }
};
class child: public Parent1,public Parent2{
    public:
    child():Parent1(5),Parent2(10){

        cout<<"child class constructor"<<endl;
    }
};


int main(){
    child c;
    cout<<"parent1 -> x -> "<<c.x<<" parent2 -> y ->
"<<c.y<<endl;

return 0;
}
```

```
Output:

parent1 class constructer with value  5
parent2 class constructer with value  10
child class constructor
parent1 -> x -> 22 parent2 -> y -> 33
```

Hierarchical inheritance

```
#include<iostream>
#include<algorithm>
#include<string>
using namespace std;
class Parent{
    public:
       Parent(){
        cout<<"parent class constructer with
value  "<<endl;
    }
};
class child1: public Parent{
    public:
    child1(){
    cout<<"child1 class constructor"<<endl;
    }
};

class child2: public Parent{
    public:
    child2(){

        cout<<"child2 class constructor"<<endl;
    }
};
int main(){
    child1 c;
    child2 c1;

return 0;
}
```

```
Output:
parent class constructer with value
child1 class constructor
parent class constructer with value
child2 class constructor
```

4.Hybrid inheritance

Diamond problem

Diamond problem arises in multiple inheritance when 2 classes inherit from one superclass and subclass derive from the both.

It complicates the code and makes it to more difficult to understand and maintain

In the below example super most class executes two times

Example:

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

```
class Parent{
    public:
       Parent(){
        cout<<"parent class constructer with
value  "<<endl;
    }
};
class child1: public Parent{
    public:
    child1(){
    cout<<"child1 class constructor"<<endl;
    }
};

class child2: public Parent{
    public:
    child2(){

        cout<<"child2 class constructor"<<endl;
    }
};

class grandchild : public child1,public child2{
    public:
    grandchild(){
        cout<<"grandchild "<<endl;
    }
};

int main(){
grandchild g1;
return 0;
}

output:
parent class constructer with value
child1 class constructor
parent class constructer with value
child2 class constructor
grandchild
```

problem with multiple inheritance

```
#include<iostream>
#include<algorithm>
using namespace std;
class Parent{
    public:
       Parent(){
        cout<<"parent class constructer with
value  "<<endl;
    }
};
class child1: public Parent{
    public:
    child1(){
    cout<<"child1 class constructor"<<endl;
    }
     void display(){
        cout<<"this is not a message"<<endl;
    }
};

class child2: public Parent{
    public:
    child2(){
        cout<<"child2 class constructor"<<endl;
    }
    void display(){
        cout<<"this is message"<<endl;
    }
};
class grandchild : public child1,public child2{
    public:
    grandchild(){
        cout<<"grandchild "<<endl;
    }
};
int main(){
  grandchild g1;
  g1.display(); // display() is ambiguous
return 0;
}
```

Polymorphism

1.compile time polymorphism

2.runtime polymorphism

1.Compile time polymorphism

Function overloading

Operator overloading

```
#include<iostream>
#include<algorithm>
#include<string>
using namespace std;
    class Student{
        public:
        void display(){
        cout<<"this is a message"<<endl;
        }

        void display(int age){
            cout<<"age of the student :: "<<age<<endl;
        }


        void display(string name){
            cout<<"name  :: "<<name<<endl;
        }
    };
int main(){

    Student s;
    s.display();
    s.display(22);
    s.display("tom");
    return 0;
}
```

Operator overloading

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

int main(){
int a=22,b=12;
string s1="be",s2=" cool";

cout<<a+b<<endl;
cout<<s1+s2<<endl;
}
```

Runtime polymorphism

using function overriding

virtual keyword used in parent class method to achieve function overloading.

```cpp
#include<iostream>
#include<algorithm>
#include<string>
using namespace std;

class parent{
    public:
    virtual void display(){
        cout<<"parent class display method"<<endl;
    }
    void print(){
        cout<<"parent class print method"<<endl;
    }
};

class child:public parent{
    public:
    void display(){
        cout<<"child class message"<<endl;
    }
    void print(){
        cout<<"child class print method"<<endl;
    }
};
int main(){
    parent *p;
    child c;
    p=&c;
    p->print();
    p->display();
}
Output:
parent class print method
child class message
```

Friend function

Non member function which can access the private member of the class

Here we cant access the private member directly

```
class student{
    int x;
};

int main(){
  student s;
  s.x;
}
```

Solution: use friend function

```
#include<iostream>
#include<algorithm>
using namespace std;
class student{
    int x;
    public:
    student(int y){
        x=y;
    }

    friend void print(student &obj);
};
void print(student &obj){
    cout<<obj.x<<endl;
}
int main(){
  student s(6);
  print(s);
  return 0;

}  // output  6
```

Static keyword in C++

We can use static keyword with data members and member functions.

There is only one copy of static data members

It shared memory for all objects in the class

Static function

It accesses only static members Without object creation we can access static function

```
#include<iostream>
using namespace std;
class student{
    int x;
    static int y;
    public:
    student(int a,int b){
        x=a;
        y=b;
    }
    // non static member function
    void display(){
        cout<<x<<"  "<<y<<endl;
    }
    // static member function
    static void displayNum(){
        // cout<<x<<endl; // non static member cant access
inside static function
        cout<<y<<endl;
    }
};
int student::y=0;
int main(){
 student obj1(3,6);
 obj1.display(); // 3  6
 student::displayNum(); //6
 student obj2(30,60);
 obj1.display();  // 3   60
}
```

Local variable and global variable

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

    int a=200;
    void display(int y){
        cout<<"accessing local variable "<<y<<endl;
        cout<<"accessing global variable <<a<<endl;
    }

int main(){
    cout<<a<<endl;
    display(66);

}
```

C++ Standard Template Library

Arrays

Vector

Deque

List

Stack

Queue

Priority_queue

Set

Map

1. Arrays

```cpp
#include<iostream>
#include<array>
#include<algorithm>

using namespace std;
int main(){
    // declaring normal array
    int normalArray[3]={1,2,3};

    // declaring array from stl
    array<int,4> arr={1,2,3,4};
    int size=arr.size();

    for(int i=0;i<size;i++){
        cout<<arr[i]<<endl;
    }

    cout<<"element at second index"<<arr.at(2)<<endl;
    cout<<"empty or not "<<arr.empty()<<endl;
    cout<<"first element  "<<arr.front()<<endl;
    cout<<"last element  "<<arr.back()<<endl;
}
```

Output:

1

2

3

4

element at second index3

empty or not 0

first element 1

last element 4

Vector

```cpp
#include<iostream>
#include<vector>
using namespace std;
int main(){
    vector<int> v;
    // capacity of vector
    cout<<" Capacity "<<v.capacity()<<endl;

    v.push_back(1);
    cout<<" Capacity "<<v.capacity()<<endl;

    v.push_back(2);
    cout<<" Capacity "<<v.capacity()<<endl;

    v.push_back(3);
    cout<<" Capacity "<<v.capacity()<<endl;

    cout<<"Size : "<<v.size()<<endl;

    // to find element at the particular index
    cout<<" element at the 2nd index "<<v.at(2)<<endl;
    cout<<"front "<<v.front()<<endl;
    cout<<"back "<<v.back()<<endl;

    cout<<"before pop "<<endl;
    for(int i:v){
        cout<<i<<" ";
    }cout<<endl;

    v.pop_back();
      cout<<"after pop "<<endl;
    for(int i:v){
        cout<<i<<" ";
    }cout<<endl;
}cout<< " before cleraing the size "<<v.size()<<endl;

   v.clear();

    cout<< " before cleraing the size "<<v.size()<<endl;
```

Capacity 0
Capacity 1
Capacity 2
Capacity 4
Size : 3
element at the 2nd index 3
front 1
back 3
before pop
1 2 3
after pop
1 2
before cleraing the size 2
before cleraing the size 0

Copying from one vector to another

```
#include<iostream>
#include<vector>
using namespace std;
int main(){
    vector<int> a(5,5);

    for(int i:a){
        cout<<i<<" ";
    }cout<<endl;

    vector<int> newOne(a);


    for(int i:newOne){
        cout<<i<<" ";
    }cout<<endl;

}

Output:
5 5 5 5 5
5 5 5 5 5
```

Dequeue

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

void printEle(deque<int> g)
{
    deque<int>::iterator it;
    for (it = g.begin(); it != g.end(); ++it)
        cout << *it<<" ";
    cout << '\n';
}
int main(){
    deque <int> d;
   d.push_back(1);
   d.push_front(2);

    printEle(d);// 2 1

    // d.pop_back();
    // printEle(d);//2

    // print element at the particular index
    cout<<"element at first index "<<d.at(1)<<endl;
    cout<<"front element "<<d.front()<<endl;
    cout<<"back element "<<d.back()<<endl;

    cout<<"empty or not "<<d.empty()<<endl;
    cout<<"after erase size "<<d.size()<<endl;
    d.erase(d.begin(),d.begin()+1);
    cout<<"after erase size: "<<d.size()<<endl;
    printEle(d);

}
```

Output:

2 1

element at first index 1
front element 2
back element 1
empty or not 0
after erase size 2
after erase size: 1
1

List

```cpp
#include<iostream>
#include<list>
using namespace std;
int main(){
    list<int> l;
    cout<<"sie of list "<<l.size()<<endl;
    l.push_back(1);
    l.push_front(2);

    cout<<"size  of list after push "<<l.size()<<endl;
    for(int i:l){
        cout<<i<<" ";
    }cout<<endl;

    cout<<"size of list before erasing "<<l.size()<<endl;
    l.erase(l.begin());
    cout<<"after erase"<<endl;
     for(int i:l){
        cout<<i<<" ";
    }cout<<endl;
    cout<<"size of list after erasing "<<l.size()<<endl;

    // declaring another list
    list<int> n(5,500);
    for(int i:n){
        cout<<i<<" ";
    }cout<<endl;
}
```

QUEUE

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

void printElements(queue<string > q){
    while(!q.empty()){
        cout<<q.front()<<" ";
        q.pop();
    }cout<<endl;

}
int main(){
    queue<string> q;
    q.push("tom");
    q.push("and");
    q.push("jerry");

     cout<<"size of the queue before pop
"<<q.size()<<endl;
    printElements(q);
    q.pop();
    printElements(q);
    cout<<"size of the queue after pop "<<q.size()<<endl;

    cout<<"first element "<<q.front()<<endl;
    cout<<"last element "<<q.back()<<endl;

}
```

Output:

size of the queue before pop 3
tom and jerry
and jerry
size of the queue after pop 2
first element and
last element jerry

Stack

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

void print(stack<string> s){
    while(!s.empty()){
        cout<<s.top()<<" ";
        s.pop();
    }cout<<endl;
}
int main(){
    stack<string> s;
    s.push("tom");
    s.push("and");
    s.push("jerry");

    print(s);
    cout<<"the size of the stack is "<<s.size()<<endl;
    cout<<"after pop operation the stack elements
are"<<endl;
    s.pop();
     cout<<"the size of the stack after pop is
"<<s.size()<<endl;
    print(s);
    cout<<"element at the peak or top "<<s.top()<<endl;
}
```

Output:

jerry and tom
the size of the stack is 3
after pop operation the stack elements are
the size of the stack after pop is 2
and tom
element at the peak or top and

Priority queue (max heap and min heap)

```cpp
#include<iostream>
#include<queue>
using namespace std;
int main(){
    // declaring max heap
    priority_queue<int> maxHeap;
    maxHeap.push(1);
    maxHeap.push(2);
    maxHeap.push(3);
    maxHeap.push(0);

    cout<<"size --> "<<maxHeap.size()<<endl; // size --> 4
    int size=maxHeap.size();
    for(int i=0;i<size;i++){
        cout<<maxHeap.top()<<" "; // 3 2 1 0
        maxHeap.pop();
    }cout<<endl;

    // declaring a minheap
    priority_queue<int,vector<int>, greater<int> >
minHeap;
    minHeap.push(1);
    minHeap.push(0);
    minHeap.push(7);
    minHeap.push(5);
    minHeap.push(4);

    int size1=minHeap.size();
       cout<<"is minheap empty -->
"<<minHeap.empty()<<endl;
    for(int i=0;i<size1;i++){
        cout<<minHeap.top()<<" ";
        minHeap.pop();
    }cout<<endl;

    // min heap is empty or not
    cout<<"is minheap empty --> "<<minHeap.empty()<<endl;
}
```

Output:

```
size --> 4
3 2 1 0
is minheap empty --> 0
0 1 4 5 7
is minheap empty --> 1
```

Set

```cpp
#include<iostream>
#include<set>
using namespace std;
int main(){
    set<int> s;
    s.insert(1);
    s.insert(2);
    s.insert(4);
    s.insert(2);
    s.insert(5);
    s.insert(5);
    s.insert(2);
    s.insert(0);

    for(auto i:s){
        cout<<i<<" "; //0 1 2 4 5
    }cout<<endl;

    set<int>:: iterator it=s.begin();
    it++;
     cout<<"value present at it -->"<<*it<<endl;
    s.erase(it);
      for(auto i:s){
        cout<<i<<" ";  //0 2 4 5
    }cout<<endl;
    // element  is present or not
    cout<<"-1 is present or not --> "<<s.count(-1)<<endl;
    cout<<"4 is present or not --> "<<s.count(4)<<endl;
    // to print element after the given element
    set<int>::iterator itr=s.find(2);
    for(auto it=itr;it!=s.end();it++){
        cout<<*it<<" ";
    }cout<<endl;
}
```

Map

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

int main(){
    map<int,string> m;
    m[1]="tom";
    m[2]="and";
    m[3]="jerry";

    m.insert({5,"tim"});

    cout<<"before erase "<<endl;
    for(auto i:m){
        cout<<i.first<<" "<<i.second<<endl;
    }

    //find any element
    cout<<"search 5 "<<m.count(5)<<endl;
    cout<<"search 12 "<<m.count(12)<<endl;


    m.erase(2);
    cout<<"after erase "<<endl;
    for(auto i:m){
        cout<<i.first<<" "<<i.second<<endl;
    }

    // find function // print all element after a
particular key
    auto it=m.find(3);
    for(auto i=it;i!=m.end();i++){
        cout<<(*i).first<<endl;
    }
}
```

Output:

before erase
1 tom
2 and
3 jerry
5 tim
search 5 1
search 12 0
after erase
1 tom
3 jerry
5 tim
3
5

STL algorithm

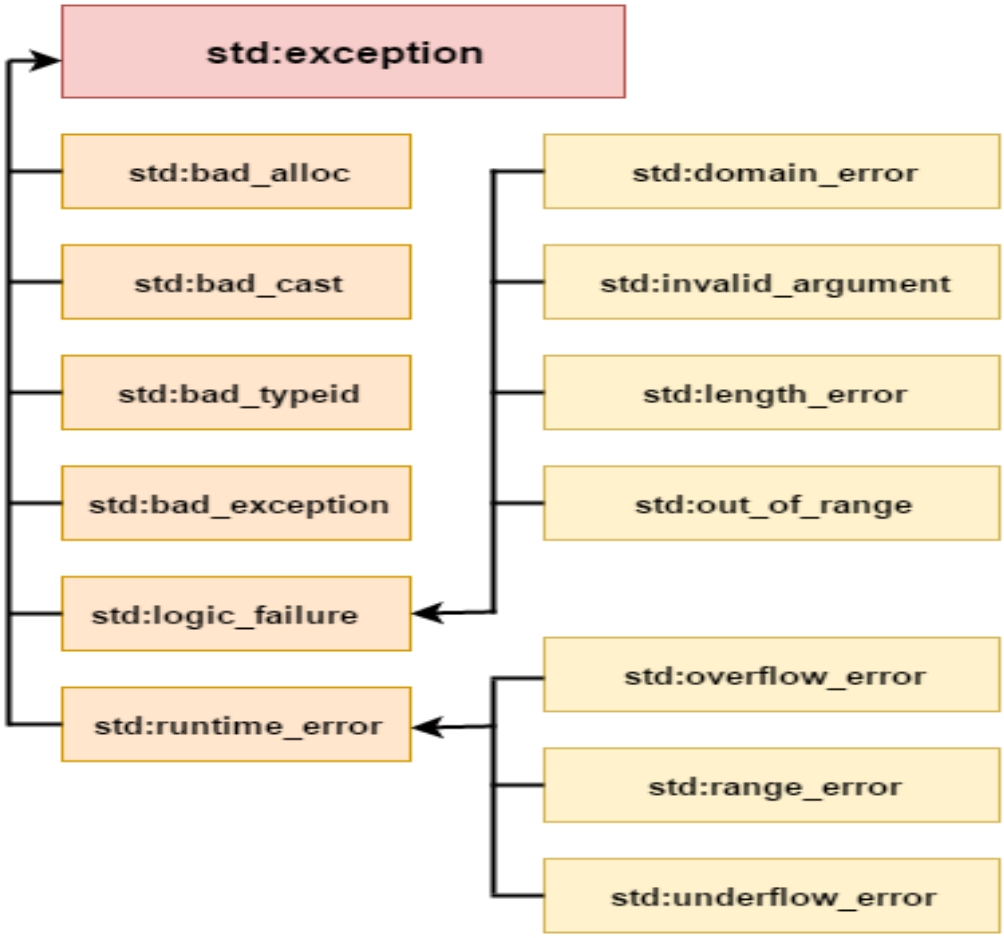
Binary search, swap, reverse of string

Max, min

```cpp
#include<iostream>
#include<algorithm>
#include<vector>
using namespace std;
int main(){
    vector<int> v;
    v.push_back(1);
    v.push_back(3);
    v.push_back(6);
    v.push_back(7);

    // binary search
    cout<<"finding 6
"<<binary_search(v.begin(),v.end(),6)<<endl;
// max and min
int a=5;
int b=12;
cout<<" max -->"<<max(a,b)<<endl;
cout<<"min -->"<<min(a,b)<<endl;
// swap
swap(a,b);
cout<<" a=5,b=12 then swap now a is "<<a<<endl;
// reverse
string str ="abcd";
cout<<" before reversing -->"<<str<<endl;
reverse(str.begin(),str.end());
cout<<" after reversing -->"<<str<<endl;
//rotate right
rotate(v.begin(),v.begin()+1,v.end());
cout<<"after rotaing "<<endl;
for(int i:v){
    cout<<i<<+" ";
}cout<<endl;
// sort the vector
// introsort combination of heap sort , quick sort and
insertion sort
sort(v.begin(),v.end());
cout<<"after sorting --> "<<endl;
for(int i:v){
    cout<<i<<+" ";
}cout<<endl;
}
```

Exception Handling




```cpp
#include<iostream>
#include<exception>
using namespace std;
int main(){
    int a =10;
    int b=0;
    try{
        if(b==0){
        throw "invalid denominator entered";
        }
    cout<<a/b<<endl;
    }
    catch(const char *e){
        cout<<"invalid denominator"<<endl<<e;
    }
}
```

```
#include<iostream>
#include<exception>
using namespace std;
int main(){
    int a =10;
    int b=0;
    try{
        if(b==0){
        throw runtime_error("invalid denominator
entered");
        }
    cout<<a/b<<endl;
    }
    catch(runtime_error e){
        cout<<"invalid denominator"<<endl<<e.what();
    }
    return 0;
}
```

Note:

We can have multiple catch block for a single try block

```
#include<iostream>
#include<exception>
using namespace std;
int main(){
    int a =10;
    int b=0;
    try{
        if(b==0){
       throw runtime_error("invalid denominator
entered");;
        }
        cout<<a/b<<endl;
    }
    catch(int e){       // throw 10
        cout<<e<<endl;
```

```
    }
    catch(const char *e){ // string is char array   throw
"hii";
        cout<<e<<endl;
    }
    catch(runtime_error e){ //
        cout<<"exception occured"<<endl<<e.what();
    }
    return 0;
}
```