1. **VECTOR**
2. **STACK**
3. **QUEUE**
4. **MAPS**
5. **SETS**
6. **DOUBLY LINKED LIST**
7. **FORWARD LINKED LIST**
8. **PAIR**
9. **TEMPLATE**
10. **BINARY SEARCH**
11. **SORT**
12. **NEXT_PERMUTATION**
13. **GCD**
14. **POW**
15. **SWAP**
16. **REVERSE**
17. **MIN/MAX**
18. **MEMSET**
19. **DYNAMIC BINDING**
20. **SIMPLE INHERITANCE**
21. **VIRTUAL FUNCTION**
22. **ENCAPSULATION**
23. **COPY CONSTRUCTOR**
24. **FUNCTION OVERLOADING**
25. **OPERATOR OVERLOADING**
26. **INTERFACE**

VECTOR

```
#include <iostream>
#include <vector>

using namespace std;

int main()
{
        vector<int> v1 {10,20,30,40};

        v1.push_back(1);  //insert 1 at the back of v1
        v1.push_back(2);  //insert 2 at the back of v1
        v1.push_back(4);  //insert 3 at the back of v1

        v1.pop_back();

        vector<int>::iterator it;

        for(it = v1.begin(); it != v1.end(); it++)
        {
                cout << *it <<" ";   // for printing the vector
        }
}
```

- vector_name.size() returns size of the vector.
- vector_name.front() retuns the element at the front of the vector (i.e. leftmost element).
- vector_name.back() returns the element at the back of the vector (i.e. rightmost element).

- size()
- empty()
- push_back()
- pop_back()
- end()
- begin()
- front()
- back()

<u>copy array to vector</u>

```
int myints[] = {1,2,3,4,5,4,3,2,1};
std::vector<int> v(myints,myints+9);
```

---

comman way to use find() in containers

```
bool status = container.find(element) != container.end();
```

STACK

```
#include <iostream>
#include <stack>

using namespace std;

int main ()
{
        stack<int> s;

        // pushing elements into stack
        s.push(2);
        s.push(3);
        s.push(4);

        cout << s.top();   // prints 4, as 4 is the topmost element

        cout << s.size();  // prints 3, as there are 3 elements in
}
```

```
}
```

- push()
- pop()
- size()
- empty()
- top()

QUEUE

```
#include <iostream>
#include <queue>

using namespace std;

int main ()
{
        queue <int> q;   // creates an empty queue of integer q

        q.push>(2);   // pushes 2 in the queue  , now front = back = 2
        q.push(3);     // pushes 3 in the queue  , now front = 2 , and back = 3

        q.pop() ;  // removes 2 from the stack , front = 3
}
```

front() returns the front element of the queue whereas back() returns the element at the back of the queue.

- Front()

- back()
- empty()
- size()
- push()
- pop()
- push_back()*
- pop_front()*

MAPS

Maps are used to replicate associative arrays. Maps contain sorted key-value pair, in which each key is unique and cannot be changed, and it can be inserted or deleted but cannot be altered.

```
#include <iostream>
#include <map>

using namespace std;

int main ()
{
        map<int,int> m{ {1,2} , {2,3} , {3,4} };
        /* creates a map m with keys 1,2,3 and their corresponding values 2,3,4 */

        map<string,int> map1;
        /*  creates a map with keys of type character and values of type integer */
        map1["abc"]=100;    // inserts key = "abc" with value = 100
        map1["b"]=200;
        map1["c"]=300;
        map1["def"]=400;

        map<char,int> map2 (map1.begin(), map1.end());
        /* creates a map map2 which have entries copied from map1.begin() to map1.end() */

        map<char,int> map3 (m);
        /* creates map map3 which is a copy of map m */
}
```

```
#include <iostream>
#include <map>

using namespace std;

int main ()
{
        map<int,string> m{ {1,”fayas”} , {2,”farwees”} , {3,”peerkalaikadu”} };


        cout << m.at(1) ;  // prints value associated with key 1 ,i.e fayas
        cout << m.at(2) ;  // prints value associated with key 2 ,i.e farwees

        /* note that the parameters in the above at() are the keys not the index */

        cout << m[3] ; // prints value associated with key 3 , i.e majeed



        m.at(1) = "shamshad";   // changes the value associated with key 1 to “shamshad”
        m[2] = "salmaa";   // changes the value associated with key 2 to “salmaa”

        m[4] = "majid";
        /* since there is no key with value 4 in the map, it insert a key-value pair in map with
           key=4 and value = “majid” */

        m.at(5) = "mannaiyah";
        /* since there is no key with value 5 in the map , it throws an exception  */
}
```

```
#include <iostream>
#include <map>

using namespace std;
```

```
int main ()
{
        map<int,int> m{{1,2} , {2,3} , {3,4} };

        m.insert( pair<int,int> (4,5));
        /* inserts a new entry of key = 4 and value = 5 in map m */

        /* make_pair() can also be used for creating a pair */
        m.insert( make_pair(5, 6));
        /* inserts a new entry of key = 5 and value = 6 */


        map::iterator i , j;
        i = m.find(2);    // points to entry having key =2
        j = m.find(5);    // points to entry having key =5

        map<int,int> new_m;

        new_m.insert(i,j);
         /* insert all the entries which are pointed by iterator i to iterator j*/

        m.insert( make_pair(3,6));
         // do not insert the pair as map m already contain key = 3 */

        m.insert_or_assign( make_pair(3,6));  // assign value = 6 to key =3
}
```

```
/* this program demonstrates the find the value */

#include <iostream>
#include <map>

int main ()
{
  std::map<char,int> mymap;
  std::map<char,int>::iterator it;

  mymap['a']=50;
  mymap['b']=100;
  mymap['c']=150;
  mymap['d']=200;

  it = mymap.find('b');
  if (it != mymap.end())
    mymap.erase (it);

  // print content:
  std::cout << "elements in mymap:" << '\n';
  std::cout << "a => " << mymap.find('a')->second << '\n';
  std::cout << "c => " << mymap.find('c')->second << '\n';
  std::cout << "d => " << mymap.find('d')->second << '\n';

  return 0;
}
```

- begin()
- end()
- at()
- size()
- empty()
- insert()
- find()
- erase()

SETS

Sets are containers that store unique elements following a specific order.

Sets values cannot modifed but insert and remove operations can be done.

```
#include <iostream>
#include <set>

int main ()
{

        std::set<std::string> s;
        std::cout << "Adding 'Hello' and 'World' to the set twice" << std::endl;

/*
         to use find()
         bool status = s.find('g') != s.end() ;
```

```
 */

	s.insert("Hello");
	s.insert("World");
	s.insert("Hello");
	s.insert("World");

	std::cout << "Set contains:";
	while (!s.empty())
	{
		std::cout << ' ' << *s.begin();
		s.erase(s.begin());
	}

  return 0;
}
```

```
#include<bits/stdc++.h>

using namespace std;

int main()
{
set<int> s;

for(int i=0;i<10;i++)
	s.insert(i);

int x;
cin >> x;

bool status = s.find(x) != s.end();
if(status)
	cout << "present";
else
	cout << "Not present";
cout<<endl;

s.insert(x);
status = s.find(x) != s.end();
if(status)
            cout << "present";
else
            cout << "Not present";
cout<<endl;

while( !s.empty() )
{
            cout << *s.begin() << endl;
            s.erase( s.begin() );
}

return 0;
}
```

- size()
- insert()
- begin()
- end()
- find()
- erase()

LIST [DOUBLY LINKED LIST]

```
#include <iostream>
#include <list>

int main ()
{
  int myints[] = {75,23,65,42,13};
  std::list<int> mylist (myints,myints+5);

  std::cout << "mylist contains:";

            for (std::list<int>::iterator it=mylist.begin(); it != mylist.end(); ++it)
            std::cout << ' ' << *it;

  std::cout << '\n';

  return 0;
}
```

some more functions :

- list::empty()
- list::end()
- list::begin
- list::front()
- list::back()
- list::insert()
- list::size()
- list::reverse()
- list::sort()
- list::swap()
- list::pop_back()
- list::pop_front()

FORWARD_LIST [SINGLY LINKEDLIST]

```
// forward_list::begin example

#include <iostream>
#include <forward_list>

int main ()
{
  std::forward_list<int> mylist = { 34, 77, 16, 2 };

  std::cout << "mylist contains:";

        for ( auto it = mylist.begin(); it != mylist.end(); ++it ) // note : auto keyword used here
        std::cout << ' ' << *it;

  std::cout << '\n';

  return 0;
}
```

some more functions :

- empty()
- end()
- begin()
- push_front()
- pop_front()
- reverse()
- swap()
- sort()

PAIR

The pair container is a rather basic container. It can be used to store two elements, called first and second, and that's about it.

```
#include<bits/stdc++.h>

using namespace std;

int main()
{
        pair<int,string> pr;

        pr = make_pair(1,"fayas");
        pr = make_pair(2,"shamshad");
        pr = make_pair(3,"majeed");
```

```
            cout << pr.first << " " << pr.second << endl;

return 0;
}

/* output */

3 majeed


 #include<bits/stdc++.h>

typedef std::pair<int,char[100]> intPair;
typedef std::vector<intPair> vectorPairs;

int main()
{
    int numEntries;
    std::cin >> numEntries;

    vectorPairs pairs(numEntries);

    for(vectorPairs::iterator itor = pairs.begin(); itor != pairs.end(); ++itor)
    {
        std::cin >> itor->first >> itor->second;
    }

    for(vectorPairs::iterator itor = pairs.begin(); itor != pairs.end(); ++itor)
    {
        std::cout << itor->first << " " << itor->second << std::endl;
    }

    return 0;
}

/* output */

5
1 aaa
2 bbb
3 ccc
4 ddd
5 eee
```

some more function :
make_pair() ,
move(),
begin(),
end().

TEMPLATE

```
#include<bits/stdc++.h>
using namespace std;

template <class T> // keyword -> template //  T variable accepts any type
T MaxFun(T a, T b)
{
	return (a>b ? a:b);
}

int main()
{
int x = 10;
int y = 133;

cout<<MaxFun(x,y)<<endl<<endl;

return 0;
}
```

sample program converting from integer to string using stringstream and converting from string to c-style char array.

```
#include<bits/stdc++.h>
using namespace std;

int main()
{

	int num = 123;
	string str;
	stringstream sss;  // string buffer

	sss << num;
	str = sss.str();
	const char *cptr = str.c_str();

	char ch[100];
	strcpy(ch,cptr);
cout<<"-> "<<str<<"\n\n"<<ch<<" \n\n";
return 0;
}
```

BINARY_SEARCH

Returns true if any element in the range [first,last] is equivalent to val, and false otherwise.

Note : it must be a sorted array.

```
std::cout << "looking for a 3... ";
if (std::binary_search (v.begin(), v.end(), 3))
std::cout << "found!\n"; else std::cout << "not found.\n";

std::cout << "looking for a 6... ";
if (std::binary_search (v.begin(), v.end(), 6, myfunction))
std::cout << "found!\n"; else std::cout << "not found.\n";
```

SORT array

```
#include<bits/stdc++.h>

using namespace std;

int main()
{
        int arr[] = {3,2,1,6,5,4,9,8,7,6};

        for(int i=0;i<10; i++)cout<<arr[i] << " " ;

        cout << endl;

        sort(arr,arr+10);
        for(int i=0;i<10;i++)cout<< arr[i] << " ";

return 0;
}
```

SORT vector

Sorts the elements in the range [first,last] into ascending order.

```
sort(arr,arr+size);

sort(vec.begin(),vec.end());
```

to print vector :

sample vector : vector<int> vec = {1,2,3,4,5};

```
for(vector<int>::iterator i = 0; i != vec.end(); i++)cout << vec[i];

/* sort in ASC order - default */
std::sort(arr,arr+n) ;

/* sort in DESC order */
std::sort(arr,arr+n,std::greater<int>()) ;


#include <bits/stdc++.h>
using namespace std;
```

```
int main()
{

int n;
cin >> n;

int arr[n];

for(int i=0;i<n;i++)cin>>arr[i];

/* sort in ASC order - default */
std::sort(arr,arr+n);

for(int i=0;i<n;i++)cout << arr[i] <<" " ;

cout << endl << endl;

/* sort in DESC order */
std::sort(arr,arr+n,std::greater<int>());

for(int i=0;i<n;i++)cout << arr[i] <<" " ;

return 0;
}

/*

this program helps to sort the struct containers by calling the 3rd parameter in stl::sort()
functions.

Also this program demostrates the sorting by modifying the calling functions. (i.e ASC / DESC
order)

*/


#include <iostream>
#include <algorithm>
#include <vector>
#include <string>

using namespace std;

struct Person
{
   string name;
   int age;
   string favoriteColor;
};

bool sortByName(const Person &lhs, const Person &rhs) { return lhs.name < rhs.name; }
bool sortByAge(const Person &lhs, const Person &rhs) { return lhs.age < rhs.age; }
bool sortByColor(const Person &lhs, const Person &rhs) { return lhs.favoriteColor <
rhs.favoriteColor; }

const unsigned numberOfPeople = 2;

int main()
{
    vector<Person> people(numberOfPeople);

    for (vector<Person>::size_type i = 0; i != numberOfPeople; ++i)
    {
        cout << "Person #" << i + 1 << " name: ";
        cin >> people[i].name;

        cout << "Person #" << i + 1 << " age: ";
        cin >> people[i].age;

        cout << "Person #" << i + 1 << " favorite color: ";
        cin >> people[i].favoriteColor;
    }

    cout << "\n\n";

    // Sort by name
    sort(people.begin(), people.end(), sortByName);
    for (Person &n : people)
        cout << n.name << " ";

    cout << endl;

    // Sory by age
    sort(people.begin(), people.end(), sortByAge);
    for (Person &n : people)
        cout << n.age << " ";

    cout << endl;

    // Sort by color
    sort(people.begin(), people.end(), sortByColor);
    for (Person &n : people)
        cout << n.favoriteColor << " ";

    return 0;
}
```

NEXT_PERMUTATION

```
std::string moves = "xxxxxoooo";
sort(begin(moves), end(moves));

while (std::next_permutation(begin(moves), end(moves)))
{
    std::cout << moves << std::endl;
}
```

GCD

The libstdc++ algorithm library has a hidden gcd function (I'm using g++ 4.6.3).

```
#include <iostream>
#include <algorithm>

using namespace std;

int main()
{
  cout << std::__gcd(100,24);
  return 0;
}
```

POW / POWL

```
 std::pow(2,10);
```

SWAP

```
std::swap(x,y);
```

REVERSE

```
std::reverse(myvector.begin(),myvector.end());
```

MIN / MAX

```
  std::cout << std::min(2,1) << '\n';
  std::cout << std::min('a','z') << '\n';
  std::cout << std::min(3.14,2.72) << '\n';
```

MEMSET

Fill block of memory
Sets the first num bytes of the block of memory

```
#include <stdio.h>
#include <string.h>

int main ()
{
  char str[] = "almost every programmer should know memset!";
  memset (str,'#',6);
  puts (str);
  return 0;
}
```

Input :

```
almost every programmer should know memset!
```

Output:

```
###### every programmer should know memset!
```

Dynamic Binding

Linking a procedure call to the code that will be executed only at a run time.

```
#include<bits/stdc++.h>

using namespace std;

int square(int sq)
{
	return sq*sq;
}

int cube(int cb)
{
	return cb*cb*cb;
}

int main()
{

int ch;
int x = 5;

cout << "Enter 0 - square , 1 - cube : " << endl;
cin >> ch;

int (*ptr)(int);

if( ch == 0 )
{
	ptr = square;
}
else if(ch == 1)
{
	ptr = cube;
}

cout << ptr(x) << endl;
return 0;
}
```

Output :

```
Enter 0 - square , 1 - cube :
0
25
```

Simple Inheritance

```
#include<bits/stdc++.h>

using namespace std;

class Value
{

private :

protected :
int x;

public :

void setX(int val)
{
        x = val;
}

};

class SqValue : public Value
{

private :

public :

void show()
{
        cout << "--- > " << x*x << endl;
}

};


int main()
{
```

```
SqValue s;
s.setX(5);
s.show();

return 0;
}
```

Output :

```
--- > 25
```

Virtual Function ;

Virtual keyword is used in superclass to call the sub class member functions.

```
/*
Reqd :
inheritance and virtual keyword
*/

#include<bits/stdc++.h>
using namespace std;

class SuperClass
{
private:
public:
virtual void area()
{
        cout << "SuperClass area function" << endl;
}
};

class SubClass:public SuperClass
{
private :
public :
virtual void area()
{
        cout << "SubClass area function " << endl;
}

};

int main()
{
SuperClass *SCptr;
SubClass objSubClass;

SCptr = &objSubClass;
SCptr -> area();

return 0;
}
```

```
Output :
SubClass area function

/* if we remove “virtual” keyword  then output will be :

SuperClass area function
 */
```

Encapsulation (also known as Data Abstraction)

Method of combining the data and function inside the class. Hides data from being accessed outside the class directly. Two types : Data abstraction , Function Abstraction – It is used without showing how it is implemented.

```
#include<bits/stdc++.h>

using namespace std;

class Sample
{

private:

int var;

public:

int addition(int a,int b)
{
        var = a+b;
}

void show()
{
        cout << var << endl;
}

};

int main()
{

Sample s;

s.addition(4,6);
s.show();
return 0;
}
```

Copy constructor

To “reproduce” a identical copy of an original existing object

```
#include<bits/stdc++.h>

using namespace std;

class Sample
{
private:

int varA, varB;

public:

void setValues(int a,int b)
{
        varA = a;
        varB = b;
}

void show()
{
        cout << varA << "  "<< varB << endl;
}

};

int main()
{

Sample One , Two;

One.setValues(2,3);

/* copy values from object One to Two */
Two = One;
Two.show();

return 0;
}
```

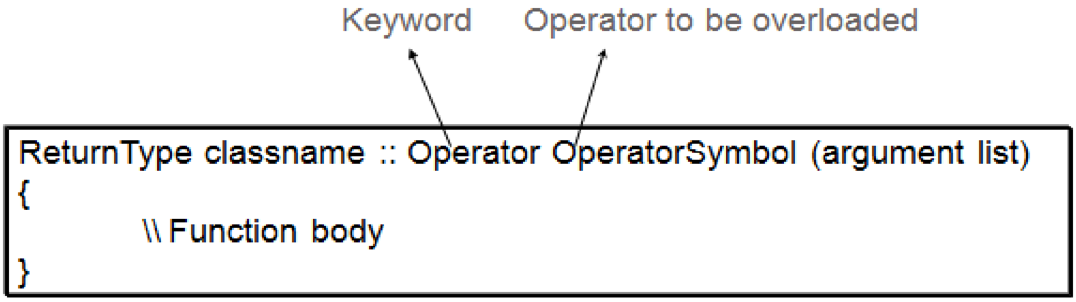
Function overloading

```
class Sample
{
private:
public:
/* function with no parameters */
void show()
{
        cout << "show I/O" << endl;
}

/* function with integer parameters */
void show(int i)
{
        cout << i << endl;
}

/* function with float parameters */
void show(double f)
{
        cout << f << endl;
}

/* function with character parameters */
void show(char *c)
{
        cout << c << endl;
}
};

int main()
{
Sample s;

s.show();
s.show(123);
s.show(333.33);
s.show("zaza");

return 0;
}
```

```
Output
display show I/O
123
333.33
zaza
```

Operator overloading

programmer can use operator with user-defined as well.



```
ReturnType classname :: Operator OperatorSymbol (argument list)
{
        \\ Function body
}
```

```
#include <bits/stdc++.h>

using namespace std;

class Sample
{
private :

int x,y;

public :

void setXY(int a, int b)
{
        x = a;
        y = b;
}

void show()
{
        cout << x << " " << y <<endl;
}

/* add two objects */
Sample operator + (const Sample &s)
{

Sample obj;

obj.x = this->x + s.x;
obj.y = this->y + s.y;

return obj;
}

};

int main()
{

Sample s1, s2;
s1.setXY(2,2);
s2.setXY(3,3);

Sample s3 = s1+s2;
s3.show();

return 0;
}
```

Output :

```
5  5
```

Interfaces :

Interface descrobes the capability of a class without implementation of class. A class is made abstract by declaring atleast one function is pure virtual function. (Placing “=0” in virtual function) Abstract class provide a base class in which other class can inherit It cannot be used for instantiate for objects.

```
class Shape
{

protected: // protected

int x,y;

public:
/* PURE VIRTUAL FUNCTION PROVIDE INTERFACE FRAMEWORK  *
virtual int getArea() = 0;

void setXY(int a, int b)
{
        x = a;
        y = b;
}
};

class Rectangle : public Shape
{

private:

public:
int getArea()
{
        return x*y;
}

};

class Triangle : public Shape
{

private:

public:

int getArea()
{
        return 0.5*x*y;
}

};

int main()
{
Triangle t;
Rectangle r;

t.setXY(5,5);
r.setXY(5,5);

cout << "Triangle area :" << t.getArea() << endl;
cout << "Rectangle area : " << r.getArea() << endl;

return 0;
}
```