STL (Standard Template Library)

Template:

- Template is keyword.
- It's used to define function template & class template. To make our class and function is generic (Generalized).

Syntax for function template: (it's a generic (generalized) function i.e. can use in string, int, double, class etc any type of value……)

Traditionally: int print (int a, int b) { }

template <class X> X print (X a, X b) {…..}

template <class X, class Y> X print (X a, Y b) {...}

here class mean not actual class

X = placeholder (generalized type)

Syntax for class template: (it's a generic (generalized) class i.e. can use in string, int, double, class etc any type of value……)

Traditionally: class demo {….};

template <class X> class demo {……};

X = placeholder (generalized type)

Ex:

```
Template <class X> class Demo {
    X arr[10];
};
int main(){
```

Demo <int> obj1;

Demo <double>obj2;

Demo <student> obj3;

}

```
//.....2.template function.......

#include <iostream>
using namespace std;
    template <class X, class Y> X bigger(X a, Y b) {            //X
& Y = placeholder(generalised type)
        if(a>b) {
            return a;
        }
        else {
            return b;
        }
    }
    int main() {
        cout<<"float: "<<bigger(371.2f, 311)<<endl;
        cout<<"char: "<<bigger( 69,'C')<<endl;
    return 0;
    }
```

Output:

float: 371.2

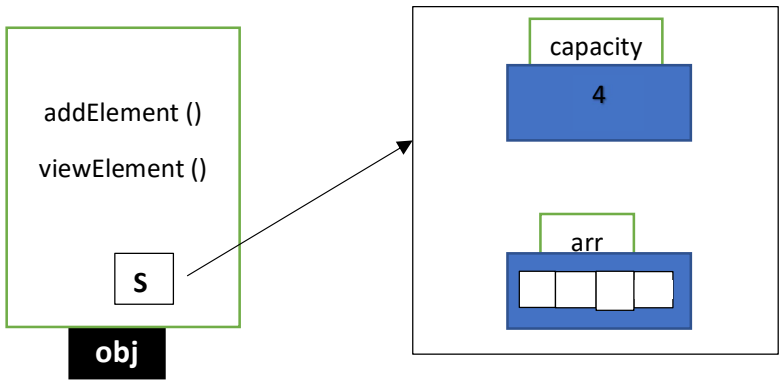
char: 69

```
//.........3.template class......any type value can store in
class........

#include <iostream>
using namespace std;
    template <class X> class Demo {             //template class
        private:
```

```
        struct ControlBlock {
            int capacity;
            int *arr;
        };
        ControlBlock *s;

        public:
        Demo(int capacity) {
            s = new ControlBlock();             //obj of struct
            s->capacity = capacity;
            s->arr = new X(s->capacity);         //arr[capacity]
here dynamically
        }

        void addElement(int index, X data) {
            if(index>=0 && index<=s->capacity-1)
                s->arr[index] = data;
            else
                cout<<"Array index not valid"<<endl;
        }

        void viewElement() {
            for (int i=0;  i<s->capacity;  i++)
                cout<<" "<<s->arr[i];
        }
    };

    int main()
    {
        Demo <int>obj(4);
        obj.addElement(0, 22);
        obj.addElement(1, 33);
        obj.addElement(3, 44);

        obj.viewElement();
    return 0;
    }
```

Output:

22 33 garbage_value 44

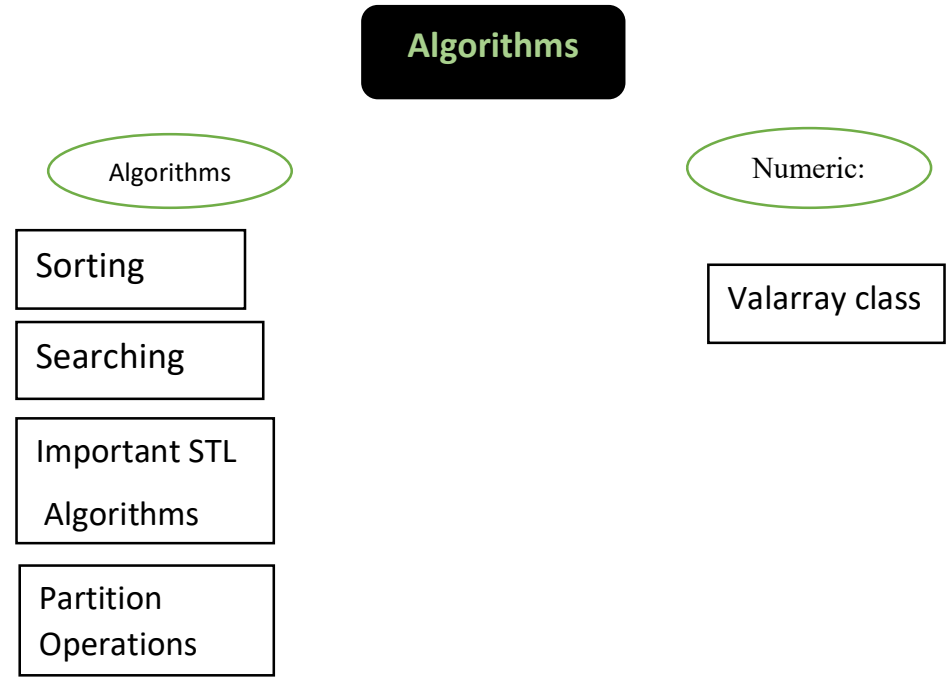


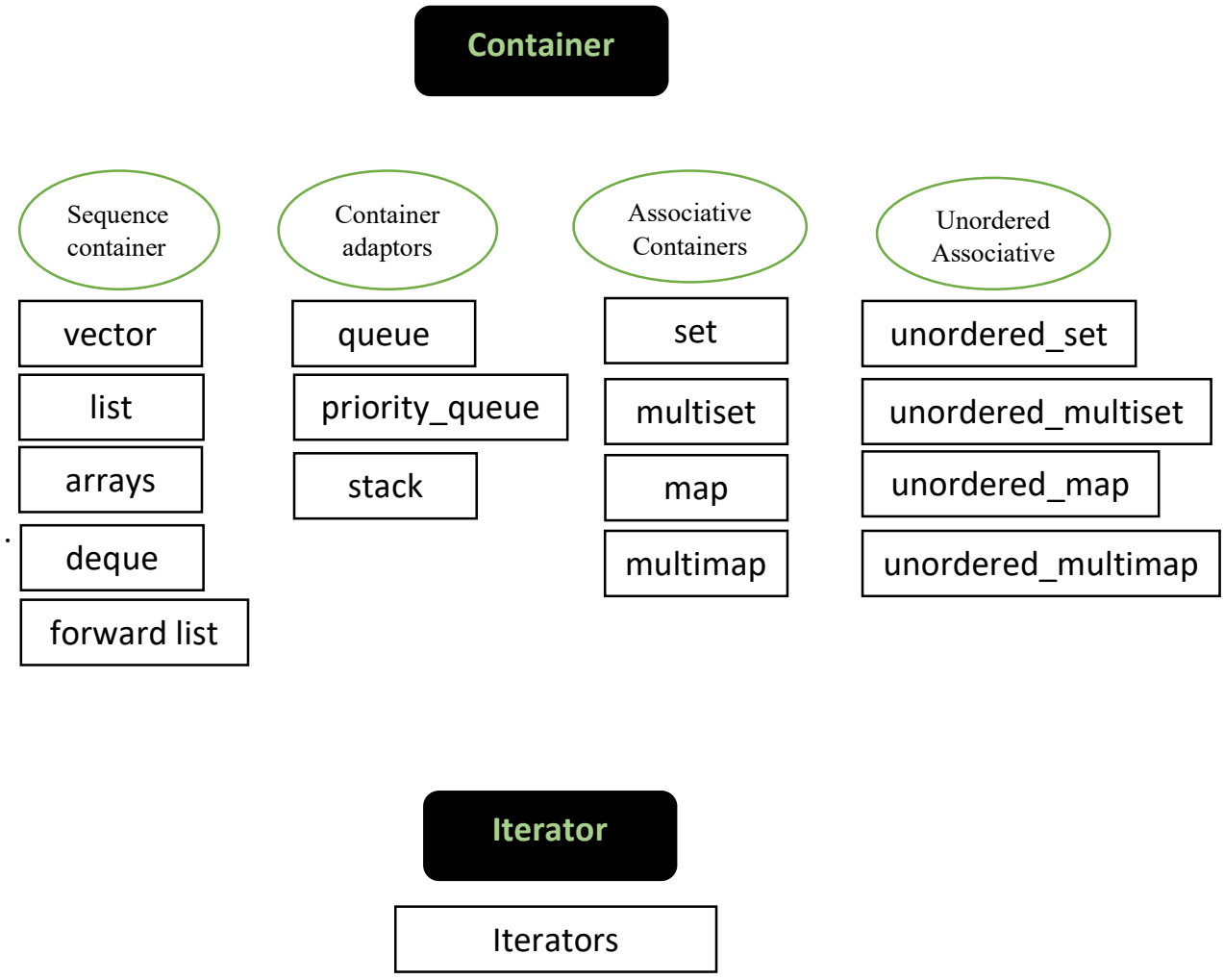
****
****

STL:

- It is set of template classes which is by made template keyword internally..i.e. these class and functions are generic (generalized)..
- STL has 3 Component:

1. **Algorithms- Specially design for various operation perform on elements in the container.**
2. **Containers- Container class store data or object. and it's generic i.e., any type of data.**
3. **Iterators- just like pointer.**
4. **Functors (function object)- classes which can act as functions.**





Algorithms:

Methods- sort, search, upper bound, lower bound, sort, comparator, max_element, accumulate, find, count, next permutation etc.

Containers:

Container is a collection of classes. and class are internally template classes i.e., generic.

Mainly generic classes (containers) are:-

1. vector (arrays)
2. queue (queues)
3. stack (stack)
4. priority_queue (heaps)
5. list (linked list)

6. set (trees)
7. map (associative arrays)

Eg:

1) list<int> obj;
2) vector<char> obj;
3) map<student> obj;

```
#include <iostream>
#include <list>
using namespace std;
 class demo{

 };
int main()
{
    // list obj;           //error
    list<int> obj;
    list<demo> obj2;
return 0;
}
```

Nested Containers:

1) vecter < vecter < int > >
2) map < int, pair < int > >
3) vecter < map < int, set < int > > >

Iterator:

vector < int > :: iterator x;

methods- begin(), end()

1. ***array:*** implement the static array.

* #include < array >

* array < type, size > arr_name;

* array <type, size> arr_name = {val_1, val_2, ……val_n};

```
//.......2 .at() .front() .back()  .size()  .empty()..........

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

    int main()
    {
        array<int,5> arr = {11,22,33,44};
        cout<<arr.at(4)<<endl;            //o/p: 0
        cout<<arr[2]<<endl;
        cout<<arr.front()<<endl;
        cout<<arr.back()<<endl;
        cout<<arr.size()<<endl;
        bool x = arr.empty();              //return: 0
        cout<<boolalpha<<(x)<<endl;        //convert to string
    return 0;
    }
```

Output:

0

33

11

0

5

False

```
//.......5 .begin() .endl()..........

#include <iostream>
```

```cpp
#include <array>
#include <algorithm>
using namespace std;

    int main()
    {
        array<int,5> arr1 = {11, 4, 13, 44, 5};
        sort(arr1.begin(), arr1.end());

        // for (int i = 0; i < arr1.size(); i++)
        //     cout<<arr1.at(i)<<"  ";

        for(auto x : arr1)
             cout<<x<<' ';          //' ' only single space

    return 0;
    }
```

Output:

4 5 11 13 44

2. ***vector:*** implement the Dynamic array. And access random element.

*** Size grow during execution: 1,2,4,8,16.32,64……

*a) vector<int> a; //static

b) vector<int> *v = new vector<int>(); //dynamic

c) vector<string> v {“pawan”, “harray”}; (or) v.push_back(value);

d) vector<string> v(4); //all null

d.1) vector<string> v(4,”hey”); //all values are ‘hey’

* operators with vector: ==, !=, >, <, >=, <=

```cpp
//.........vector.....always reference will try for copy.......

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

template<class X>
```

```
X print(vector<X> &a) {            //void print(vector<int> &a){
    cout<<"print: ";
    for (auto x : a)    {
        cout<<x<<' ';
    }
}
int main() {
    int element, size;
    vector<int> a;
    cout<<"vector size: ";
    cin>>size;
    cout<<"Enter elements: ";
    for (int i = 0; i < size; i++) {
        cin>>element;
        a.push_back(element);
    }
   print(a);
return 0;
}
```

Output:

vector size: 4

Enter elements: 1

2

3

4

print: 1 2 3 4

```
//.....3.  .erase(element)  iterator .insert(index, value).........

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

int main()
{
    vector<int> v2 {30,20,40,20,10};
    v2.erase(v2.begin());
```

```
    for(auto x : v2)
        cout<<x<<" ";
    cout<<endl;

    vector<int> :: iterator i = v2.begin();
    v2.insert(i+2, 99);            //v2.insert(i+2, 5, 9999)   //5
copy insert at index 2

    for(auto x : v2)
             cout<<x<<" ";

return 0;
}
```

Output:

20 40 20 10

20 40 99 20 10

3. ***pair***: 2 different types of value.

* pair <string, int> p;

* also can compare two pair: ==, !=, <, >=

```
//.........1.pair<firstValue, secondValue>.............
//...........    .first  .second   ..............

#include <iostream>
// #include <pair>
using namespace std;
int main()
{
    pair<string, int> p;
    // p = make_pair("Naveen", 22);               //inserting value
    p = {"Naveen", 22};                          //insert value
    cout<<p.first<<' ' <<p.second;               //first value &
second value
return 0;
}
```

Output:

Naveen 22

4. ***tuple:*** just like a pair.

* tuple < type1, type2, type3 >

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

int main()
{
    tuple <string, int, float> t;
    // t = make_tuple("Dhoni", 7, 14.233f);
    t = {"Dhoni", 7, 14.233f};
    cout<<get<0> (t)<<endl;
    cout<<get<2> (t);
return 0;
}
```

Output:

Dhoni

14.233

5. ***list:*** support internally doubly linear list. Vector support random access but list sequentially only.

Can't use 'at()' function. Use iterator or loop.

* list < int > t;

* list < int > t {12, 23, 34, 45};

```
//.........3.simple code....

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

void print(list<int> x) {
    cout<<"print: ";
    // x.reverse();
    list<int> :: iterator i;
```

```
    for(i = x.begin(); i != x.end(); ++i)
        cout<<*i<<' ';
}

int main()
{
    list<int> li;
    int size, element;
    cout<<"Enter size: ";
    cin>>size;
    cout<<"Insert: ";
    for (int i = 0; i < size; i++)
    {
        cin>>element;
        li.push_front(element);
    }
    print(li);

return 0;
}
```

Output:

Enter size: 5

Insert: 23

56

34

8

4

print: 4 8 34 56 23

6. ***map:*** It is an associative array. Always arrange its sorted order.

* map < int, string > player;

* map < int , string > c {{101, “Dhoni” }, {88, “rohit”}, {99, ”naveen”}};

Associative array: Contain key-value pair. Which can't be change (unique) only can insert and delete. [Key: index_no, value: it's value]

Eg.

Dhoni	kohli	raina	rohit	sahvag
110	22	33	44	55

```cpp
//.........2.insert() in map using pair<t1, t2>

#include <iostream>
#include <map>
// #include <iterator>
using namespace std;

int main()
{
    map<int, string> mp;
    mp[77]="naveen";
    mp[69]="harry";
    mp[23]="pawan";
    mp[19]="neeraj";
    mp[99]="neeru";

    mp.insert(pair<int, string>(55, "god"));

    map<int, string> :: iterator i;
    i = mp.begin();
    while(i != mp.end()) {
        cout<<i->second<<' ';
        i++;
    }

return 0;
}
```

Output:

neeraj pawan god harry naveen neeru

7. ***deque():***

* deque < int > d;

8. ***stack():***

* stack < string > s;

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

int main()
{
    stack<string> name;
    name.push("Naveen");
    name.push("Balveer");
    name.push("Kamal");
    name.push("Lokesh");
    name.push("Uraj");

    for (int i = name.size(); i > 0; i--)
    {
        cout<<name.top()<<' ';
        name.pop();
    }

return 0;
}
```

Output:

Uraj Lokesh Kamal Balveer Naveen

9. ***queue():*** FIFO

* queue<float> q;

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

void print(queue<int> x, int s) {
```

```
    cout<<"print element: ";
    for(int i=0; i<s; i++){
        cout<<x.front()<<' ';
        x.pop();
    }
}

int main()
{
    queue<int> q;
    int size, element;
    cout<<"enter size: ";
    cin>>size;
    for(int i=0; i<size; i++) {
        cin>>element;
        q.push(element);
    }

    print(q,size);

return 0;
}
```

Output:

enter size: 5

35

23

56

27

11

print element: 35 23 56 27 11

10. priority_queue(): heap (sorted max element return first)

* priority_queue < int > p;

*

```
///........max element return first.......

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

void print(priority_queue<int> x, int s) {
    cout<<"print element: ";
    for(int i=0; i<s; i++){
        cout<<x.top()<<' ';
        x.pop();
    }
}

int main()
{
    priority_queue<int> q;
    int size, element;
    cout<<"enter size: ";
    cin>>size;
    for(int i=0; i<size; i++) {
        cin>>element;
        q.push(element);
    }

    print(q,size);

return 0;
}
```

Output:

enter size: 5

23

56

34

11

43

print element: 56 43 34 23 11

11. ***set:*** implement BST(binary search tree), shorted order.

Same value print once.

* set < string > ss;

```
///......also same value print once....

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

void print(set<int> x) {
    cout<<"print element: ";
   for(int i : x){
        cout<<i<<' ';

    }
}

int main()
{
    set<int> q;
    int size, element;
    cout<<"enter size: ";
    cin>>size;
    for(int i=0; i<size; i++) {
        cin>>element;
        q.insert(element);
    }

    print(q);

return 0;
}
```

Output:
print element: 33 44 88 99
no exist or not: true

****

- Algorithm library provides abstraction.
- Insert, find, erase, count - O(n)
- Size, begin, end, empty - O(1)
- for(auto variableName : arrName){} loop
- *boolalpha<<(x) return ‘true’ or ‘false’ with string
- auto use in for loop
- begin() returns an iterator to the start element(I)
- end() returns an iterator to the end element(I)
- reverse((.begin(), .end())
- binary_search(.begin(), .end(), element)
- lower_bound(……)
- max()
- min(variable1, variable2)
- swap(……..)
- .sort() sort the element {Quick, heap, insertion}
-
- rotate((.begin(), .end())

array:

- .at(index) value at index
- arrayName[index] value at index
- .size(); size of array
- .front() return first value
- .back() return last value
- .fill(value) all values are given value
- *1starray.swap(2ndarray) swap two similar equal size array
- *sort(arr.begin(), arr.end()); but include <algorithm>
- .empty() return Boolean

vector:

- .push_back(value) insert element at last index
- .pop_back() remove element from last index
- arrayName[index] value at index
- .capacity() return capacity not size
- .size() return element
- .clear() remove all element but capacity is not
- .insert(index, element) insert element any index
- .insert(index, copy, element) insert jinna hm dalna chahe eleme.
- .emplace(.begin(), element) insert at begin
- .assign(indexes, value) value assign
- .emplace_back(element) insert at back (.push_back())
- .erase(.begin()) remove element begin
- .resize(); resize the vetor

pair:

- make_pair(1value, 2value) or p = {1value, 2value}; inserting value in pair
- .first return or insert first value in pair

- .second return or insert second value in pair

tuple:

- male_tuple(t1, t2, t3) or p = {1value, 2value, 3value}
- get < position_value > (obj) call value

list:

- .sort() sort the element {Quick, heap, insertion}
- .reverse() reverse elements
- .remove(element) delete particular element
- .push_back(element) enter element in last
- .push_front(element) enter element in front
- .pop_back(element) pop element from last
- .pop_front(element) pop element from start
- .clear() clear all element
- .front() return first element
- .back() return last element
- .erase(element)

Map:

- .at(index) print element
- loop_variable->second next element
- .insert(pair<t1, t2>(ele1, ele2)); insert element
- .insert({{index, value},{index, value},…..}); insert element
- (or) varia_name[index] = value; insert value
- .emply() return Boolean
- .max_size() return max size

Deque:

- .push_back(element)
- .push_front(element)
- .pop_back()
- .pop_front()
- .erase(limit) e.g. d.erase(d.begin(), d.begin()+1)
  one element delete
- .at(index)
- .front()
- .back()
- .empty()

stack:

- .push(element) insert element
- .pop() remove element
- .top() return top element

Queue:

- .push(element) insert element
- .pop() remove element
- .front() return first element

Priority_queue:

- .push(element) insert element
- .pop(); remove element
- .top(); return element

Set:

- .insert(element) insert value
- .count(element) element exist or not
- .find(element)