ITERATORS

```
template<class ForwardIterator,class T>
ForwardIterator search(ForwardIterator start, ForwardIterator end, T key) {
    while(start!=end){
        if(*start==key){
            return start;
        }
        start++;
    }
    return end;
}
int main() {
    list<int> l;
    l.push_back(1);
    l.push_back(2);
    l.push_back(3);
    l.push_back(4);

    auto it = search(l.begin(),l.end(),5);
    if(it==l.end()){
        cout<<"ele not present"<<endl;
    }else{
        cout<<*it<<endl;
    }
}
```

COMPARATORS

```
template<class ForwardIterator,class T,class Compare>
ForwardIterator search(ForwardIterator start, ForwardIterator end, T key,Compare cmp) {
    while(start!=end){
        if(cmp(*start,key)){
            return start;
        }
        start++;
    }
    return end;
}
class Book{
    public:
    string name;
    int price;

    Book(){
```

```
    }

    Book(string name,int price){
        this->name = name;
        this->price = price;
    }
};

    class BookCompare{
        //overload one operator
        public:
        bool operator()(Book A, Book B){
            if(A.name==B.name){
                return true;
            }
            return false;
        }
    };

    /*
    how to call this function
    lets make a object of BookCompare cmp
    cmp(); this is going to make a call to this function
    this looks like a function call but cmp is an object but this is called a functor/functional
object
    */


int main() {
    Book b1("C++",100); //old edition of the book
    Book b2("python",120);
    Book b3("java",130);
    Book b4(b1);

    list<Book> l;
    l.push_back(b1);
    l.push_back(b2);
    l.push_back(b3);

    Book key("C++",110); //new edition of the book
    /*
    if(b1==key){
        cout<<"True"<<endl;
    }
    this comparison is known to the system. for this we are going to make a compare class
    */

     BookCompare cmp;\
```

```
    /*
    if(cmp(b1,key)){
       cout<<"same books";
    }
    */

    auto it = search(l.begin(),l.end(),key,cmp);

    if(it!=l.end()){
       cout<<"book found";
    }

    return 0;
}
```

FIND FUNCTION

```
   int arr[]={1,2,3,4,5};
   int n = sizeof(arr)/sizeof(int);
   int key;
   cin>>key;
   auto it = find(arr,arr+n,key);
   cout<<it<<endl;
   int index;
   index = it-arr;
   cout<<index;
```

BINARY SEARCH

```
bool present = binary_search(arr,arr+n,key);
```

LOWER BOUND & UPPER BOUND

```
   auto it = lower_bound(arr,arr+n,key);
   //instead of auto we can use int* as well because it returns address of a bucket
   cout<<(it-arr)<<endl;

   auto it1 = upper_bound(arr,arr+n,key);
   //instead of auto we can use int* as well because it returns address of a bucket
   cout<<(it1-arr)<<endl;
```

VECTORS

```
   vector<int> d{1,2,3,4,5};
   d.push_back(6);
   for(int x:d){
```

```
    cout<<x<<",";
}
d.pop_back();

//inserting an element in the middle
d.insert(d.begin()+3,10);
d.insert(d.begin()+3,4,20);

//erase elements from the middle
d.erase(d.begin()+3);
d.erase(d.begin()+3,d.begin()+5);

cout<<d.front()<<endl;
cout<<d.back()<<endl;

//use v.reserve() for reserving the capacity..as doubling of array is an expensive operation
```

STRING TOKENIZER

```
char s[100] = "Today is a rainy day";

char *ptr = strtok(s," ");
cout<<ptr<<endl;

ptr = strtok(NULL," ");
cout<<ptr<<endl;
//maintains a static variable for array..so on each subsequent calls it is going to extract all the tokens one by one
while(ptr!=NULL){
    ptr = strtok(NULL," ");
    cout<<ptr<<endl;
}
```

PRIORITY QUEUE

```
//priority_queue<int> pq; //max priority queue
priority_queue<int, vector<int>, greater<int> > pq;
//changing the priority by giving comparator function ...min priority queue
int n;
cin>>n;
for(int i=0;i<n;i++){
    int no;
    cin>>no;
    pq.push(no); //O(Log N)
}
//remove the elements from the heap
while(!pq.empty()){
```

```
        cout<<pq.top()<<" ";
        pq.pop();
    }
```

UNORDERED MAP COUNT FREQUENCY

```
void countFreq(int arr[], int n)
{
    unordered_map<int, int> mp;

    // Traverse through array elements and
    // count frequencies
    for (int i = 0; i < n; i++)
        mp[arr[i]]++;

    // Traverse through map and print frequencies
    for (auto x : mp) {
        cout << x.first << " " << x.second << endl;
    }
}
```

DEQUE

deque<T> is like vector<T>, but also supports:

```
#include <deque>          // Include deque (std namespace)
a.push_front(x);          // Puts x at a[0], shifts elements toward back
a.pop_front();            // Removes a[0], shifts toward front
```

UTILITY(PAIR)

```
#include <utility>        // Include utility (std namespace)
pair<string, int> a("hello", 3);  // A 2-element struct
a.first;                  // "hello"
a.second;                 // 3
```

SET

(store unique elements - usually implemented as binary search trees - avg. time complexity: O(log n))

```
#include <set>            // Include set (std namespace)
set<int> s;               // Set of integers
s.insert(123);            // Add element to set
if (s.find(123) != s.end()) // Search for an element
    s.erase(123);
cout << s.size();
```