Stacks:
Stack is a container which follows the **LIFO (Last In First Out)** order and the elements are inserted and deleted from one end of the container. The element which is inserted last will be extracted first.

Declaration:

```
stack <int> s;
```

Some of the member functions of Stack are:
push( ): Insert element at the top of stack. Its time complexity is O(1).
pop( ): removes element from top of stack. Its time complexity is O(1).
top( ): access the top element of stack. Its time complexity is O(1).
empty( ): checks if the stack is empty or not. Its time complexity is O(1).
size( ): returns the size of stack. Its time complexity is O(1).

Implementation:

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

using namespace std;
int main( )
{
    stack <int> s;  // declaration of stack

    //inserting 5 elements in stack from 0 to 4.
    for(int i = 0;i < 5; i++)
    {
        s.push( i ) ;
    }

    //Now the stack is {0, 1, 2, 3, 4}

    //size of stack s
    cout<<"Size of stack is: " <<s.size( )<<endl;

    //accessing top element from stack, it will be the last inserted
element.
```

```
    cout<<"Top element of stack is: " <<s.top( ) <<endl ;

    //Now deleting all elements from stack
    for(int i = 0;i < 5;i++)
    {
        s.pop( );
    }

    //Now stack is empty,so empty( ) function will return true.

    if(s.empty())
    {
        cout <<"Stack is empty."<<endl;
    }
    else
    {
        cout <<"Stack is Not empty."<<endl;
    }

    return 0;
```

```
}
```

Output:

```
Size of stack is: 5
Top element of stack is: 4
```

```
Stack is empty.
```

Queues:
Queue is a container which follows **FIFO order (First In First Out)** . Here elements are inserted at one end (rear) and extracted from another end(front) .

Declaration:

```
queue <int> q;
```

Some member function of Queues are:
push( ): inserts an element in queue at one end(rear). Its time complexity is O(1).
pop( ): deletes an element from another end if queue(front). Its time complexity is O(1).
front( ): access the element on the front end of queue. Its time complexity is O(1).
empty( ): checks if the queue is empty or not. Its time complexity is O(1).
size( ): returns the size of queue. Its time complexity is O(1).

Implementation:

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

using namespace std;

int main() {
    char qu[4] = {'a', 'b', 'c', 'd'};
    queue <char> q;
    int N = 3;                          // Number of steps
    char ch;
    for(int i = 0;i < 4;++i)
        q.push(qu[i]);
    for(int i = 0;i < N;++i) {
            ch = q.front();
    q.push(ch);
            q.pop();
    }
    while(!q.empty()) {
        printf("%c", q.front());
        q.pop();
    }
    printf("\n");
    return 0;
}
```

Output:

```
dabc
```

Priority Queue:
A priority queue is a container that provides constant time extraction of the largest element, at the expense of logarithmic insertion. It is similar to the heap in which we can add element at any time but only the maximum element can be retrieved. In a priority queue, an element with high priority is served before an element with low priority.

Declaration:

```
priority_queue<int> pq;
```

Some member functions of priority queues are:
empty(): Returns true if the priority queue is empty and false if the priority queue has at least one element. Its time complexity is O(1).
pop(): Removes the largest element from the priority queue. Its time complexity is O(logN) where N is the size of the priority queue.
push(): Inserts a new element in the priority queue. Its time complexity is O(logN) where N is the size of the priority queue.
size(): Returns the number of element in the priority queue. Its time complexity is O(1).
top(): Returns a reference to the largest element in the priority queue. Its time complexity is O(1).

Implementation:

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

using namespace std;

int main()
{
    priority_queue<int> pq;
    pq.push(10);
    pq.push(20);
    pq.push(5);
    while(!pq.empty())
    {
        cout << pq.top() << endl;
        pq.pop();
    }
```

```
    return 0;
}
```

Output:

```
20
10
5
```