|  |
| --- |
| Stacks And QueuesIn C++ STL |

|  |  |  |
| --- | --- | --- |
|  | Fatimaa Khan |  |
|  |  |  |
|  |  |  |

Stack:

Stacks are a type of container adaptors with LIFO(Last In First Out) type of working, where a new element is added at one end (top) and an element is removed from that end only.

# Stack Syntax:

For creating a stack, we must include the <stack> header file in our code. We then use this syntax to define the std::stack:

template <class Type, class Container = deque<Type> > class stack;

Type – is the Type of element contained in the std::stack. It can be any valid C++ type or even a user-defined type.

Container – is the Type of underlying container object.

# Functions Of Stack:

The functions associated with stack are:

empty() – Returns whether the stack is empty – Time Complexity : O(1)

size() – Returns the size of the stack – Time Complexity : O(1)

top() – Returns a reference to the top most element of the stack – Time Complexity : O(1)

push(g) – Adds the element ‘g’ at the top of the stack – Time Complexity : O(1)

pop() – Deletes the most recent entered element of the stack – Time Complexity : O(1)

# Code:>

#include <stack>

using *namespace* std;

*int* main() {

*stack*<*int*> stack;

    stack.push(21);// The values pushed in the stack should be of the same data which is written during declaration of stack

    stack.push(22);

    stack.push(24);

    stack.push(25);

*int* num=0;

    stack.push(num);

    stack.pop();

    stack.pop();

    stack.pop();

    while (!stack.empty()) {

        cout << stack.top() <<" ";

        stack.pop();

    }

}

# Swap Function In Stack

*stackname1*.swap(*stackname2*)

# Push Vs Emplace

In C++ STL, both push and emplace are used to insert elements into a stack, but they differ in how they add the element:

1. push()

push() adds an already constructed object to the top of the stack.

It takes the object as a parameter, so you have to create the object first, then pass it to push().

Example:

std::stack<std::pair<int, int>> myStack;

std::pair<int, int> p(1, 2);

myStack.push(p); // pushes the already constructed pair onto the stack

2. emplace()

emplace() constructs the element in place at the top of the stack, without creating a temporary object.

It forwards the arguments to the constructor of the object directly, which avoids extra copies.

Example:

std::stack<std::pair<int, int>> myStack;

myStack.emplace(1, 2); // constructs the pair directly in the stack

Key Differences:

Efficiency: emplace() is generally more efficient because it avoids creating a temporary object and copying or moving it. It constructs the object directly in the stack.

Usage: Use push() when you already have an object to insert. Use emplace() when you want to construct the object directly inside the stack.

In performance-sensitive situations, emplace() can be faster due to reduced copying overhead.

# Queue

Queues are a type of container adaptors that operate in a first in first out (FIFO) type of arrangement. Elements are inserted at the back (end) and are deleted from the front.

// CPP code to illustrate Queue in

// Standard Template Library (STL)

#include <iostream>

#include <queue>

using *namespace* std;

// Print the queue

*void* showq(*queue*<*int*> *gq*)

{

*queue*<*int*> g = *gq*;

    while (!g.empty()) {

        cout << '\t' << g.front();

        g.pop();

    }

    cout << '\n';

}

// Driver Code

*int* main()

{

*queue*<*int*> gquiz;

    gquiz.push(10);

    gquiz.push(20);

    gquiz.push(30);

    cout << "The queue gquiz is : ";

    showq(gquiz);

    cout << "\ngquiz.size() : " << gquiz.size();

    cout << "\ngquiz.front() : " << gquiz.front();

    cout << "\ngquiz.back() : " << gquiz.back();

    cout << "\ngquiz.pop() : ";

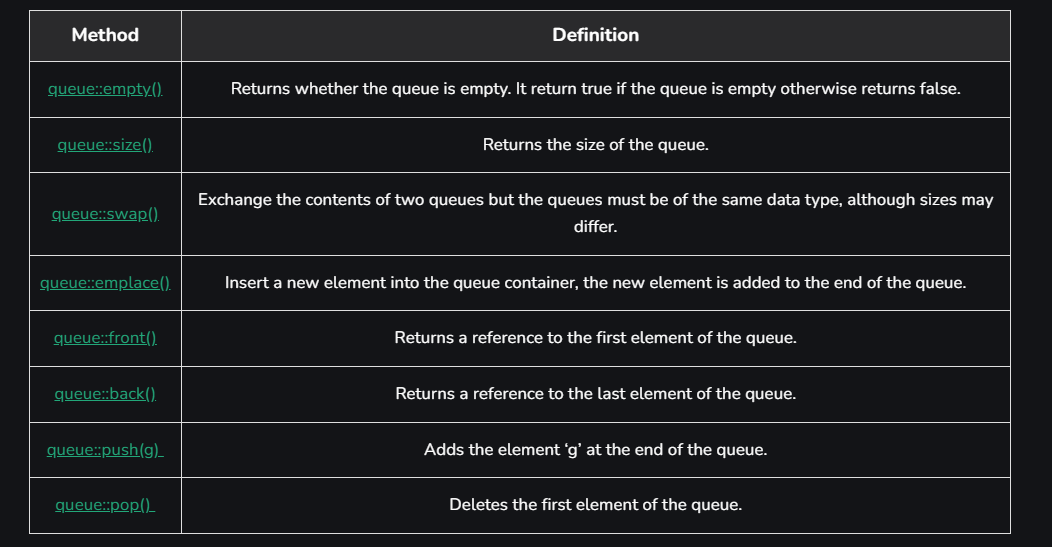
    gquiz.pop();

    showq(gquiz);

    return 0;

}

# Queue Functions



#include <iostream>

#include <queue>

using *namespace* std;

// Print the queue

*void* print\_queue(*queue*<*int*> *q*)

{

*queue*<*int*> temp = *q*;

    while (!temp.empty()) {

        cout << temp.front()<<" ";

        temp.pop();

    }

    cout << '\n';

}

// Driver Code

*int* main()

{

*queue*<*int*> q1;

    q1.push(1);

    q1.push(2);

    q1.push(3);

    cout << "The first queue is : ";

    print\_queue(q1);

*queue*<*int*> q2;

    q2.push(4);

    q2.push(5);

    q2.push(6);

    cout << "The second queue is : ";

    print\_queue(q2);

      q1.swap(q2);

      cout << "After swapping, the first queue is : ";

    print\_queue(q1);

      cout << "After swapping the second queue is : ";

    print\_queue(q2);

      cout<<q1.empty();  //returns false since q1 is not empty

    return 0;

}