MANEJO DE UNA ESTRUCTURA PILA

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

int main() {
	stack<char> pila;

	/* ¿La pila está vacía? */
	cout << pila.empty() << endl;		// 1 = verdad

	pila.push('a');				// ['a']
	pila.push('b');				// ['a', 'b']
	pila.push('c');				// ['a', 'b', 'c']

	cout << pila.top() << endl;		// c
	pila.pop();				// ['a', 'b']
	cout << pila.top() << endl;		// b

	/* ¿La pila está vacía? */
	cout << pila.empty();			// 0 = falso

	return 0;
}
```

MANEJO DE UNA ESTRUCTURA COLA

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

int main() {

	queue<char> cola;

	/* ¿La cola está vacía? */
	printf("%d\n", cola.empty());		// 1 = verdad

	cola.push('a');				// ['a']
	cola.push('b');				// ['a', 'b']
	cola.push('c');				// ['a', 'b', 'c']

	printf("%c\n", cola.front());		// a
	cola.pop();				// ['b', 'c']
	printf("%c\n", cola.front());		// b

	/* ¿La cola está vacía? */
	printf("%d\n", cola.empty());		// 0 = falso

	getchar();

	return 0;
}
```

MANEJO DE UNA ESTRUCTURA BICOLA

```
#include <cstdio>
#include <deque>
using namespace std;

int main() {

	deque<char> bicola;

	/* ¿La bicola está vacía? */
	printf("%d\n", bicola.empty());		// 1 = verdad

	bicola.push_front('a');			// ['a']
	bicola.push_front('b');			// ['b', 'a']
	bicola.push_back('x');			// ['b', 'a', 'x']
	bicola.push_back('y');			// ['b', 'a', 'x', 'y']

	printf("%c - %c\n", bicola.front(),
		bicola.back());			// b - y

	bicola.pop_front();			// ['a', 'x', 'y']
	bicola.pop_back();			// ['a', 'x']

	printf("%c - %c\n", bicola.front(),
		bicola.back());			// a - x

	/* ¿La bicola está vacía? */
	printf("%d\n", bicola.empty());		// 0 = falso

	getchar();

	return 0;
}
```

MANEJO DE UNA ESTRUCTURA COLA DE PRIORIDAD

```
#include <cstdio>
#include <string>
#include <queue>
using namespace std;

int main() {

	priority_queue<int> cp;

	/* ¿La cola de prioridad está vacía? */
	printf("%d\n", cp.empty());		// 1 = verdad

	cp.push(100);				// [100]
	cp.push(10);				// [100, 10]
	cp.push(50);				// [100, 50, 10]

	printf("%d\n", cp.top());		// 100
	cp.pop();				// [50, 10]
	printf("%d\n", cp.top());		// 50

	/* ¿La cola de prioridad está vacía? */
	printf("%d\n", cp.empty());		// 0 = falso

	priority_queue< pair<int, string> > pares;
	pair<int, string> resultado;

	/* ¿La cola de prioridad está vacía? */
	printf("%d\n", pares.empty());		// 1 = verdad

	pares.push(make_pair(100, "a"));	// [<100,a>]
	pares.push(make_pair(10, "b"));		// [<100,a>, <10, b>]
	pares.push(make_pair(50, "c"));		// [<100,a>, <50, c>, <10, b>]

	resultado = pares.top();
	printf("id=%d, nombre=%s \n", resultado.first,
		((string)resultado.second).c_str());	// Id=100, nombre=a
	pares.pop();				// [<50, c>, <10, b>]
	resultado = pares.top();
	printf("id=%d, nombre=%s \n", resultado.first,
		((string)resultado.second).c_str());	// Id=50, nombre=c

	/* ¿La cola de prioridad está vacía? */
	printf("%d\n", pares.empty());		// 0 = falso

	getchar();

	return 0;
}
```