Ayudantía 8

Stack y Queue (Pilas y Colas)

Benjamín Aceituno

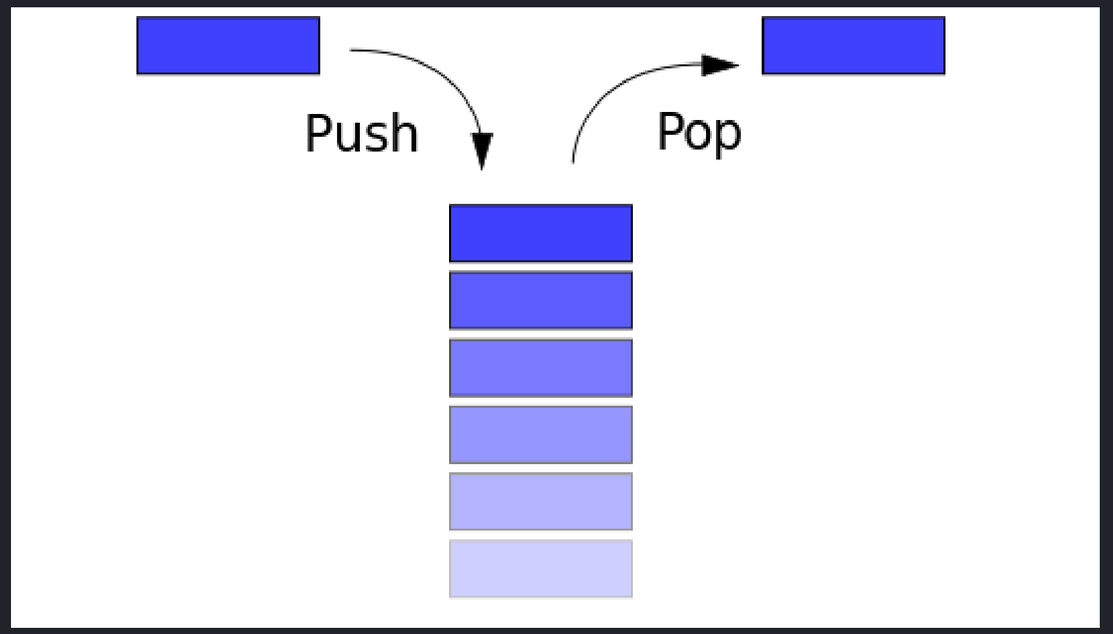
Programación Avanzada S09

Stack

El stack es un contenedor de objetos que ocupa la metodología LIFO, al igual que el vector, los ultimos elementos en entrar serán los últimos en salir. Se puede entender como por su traducción literal “pila”. Donde los elementos se van apilando dentro del contenedor

Para recordar...

LIFO quiere decir Last In First Out, es decir, último entrar, primero en salir. Intenten verlo como si fuera almacenar una pila de cajas, nosotros ponemos cajas una encima de otra, y si sacamos una caja será la última que hemos puesto.



Funciones

El stack en C++ se debe incluir de esta manera en el código

```
#include <stack>
```

Se declara de esta forma `stack<tipoDeDato> nombre;`

```
stack<int> numeros;
stack<string> palabras;
stack<Objeto*> objetos;
```

push(item): Inserta un ítem

pop(): Elimina un ítem.

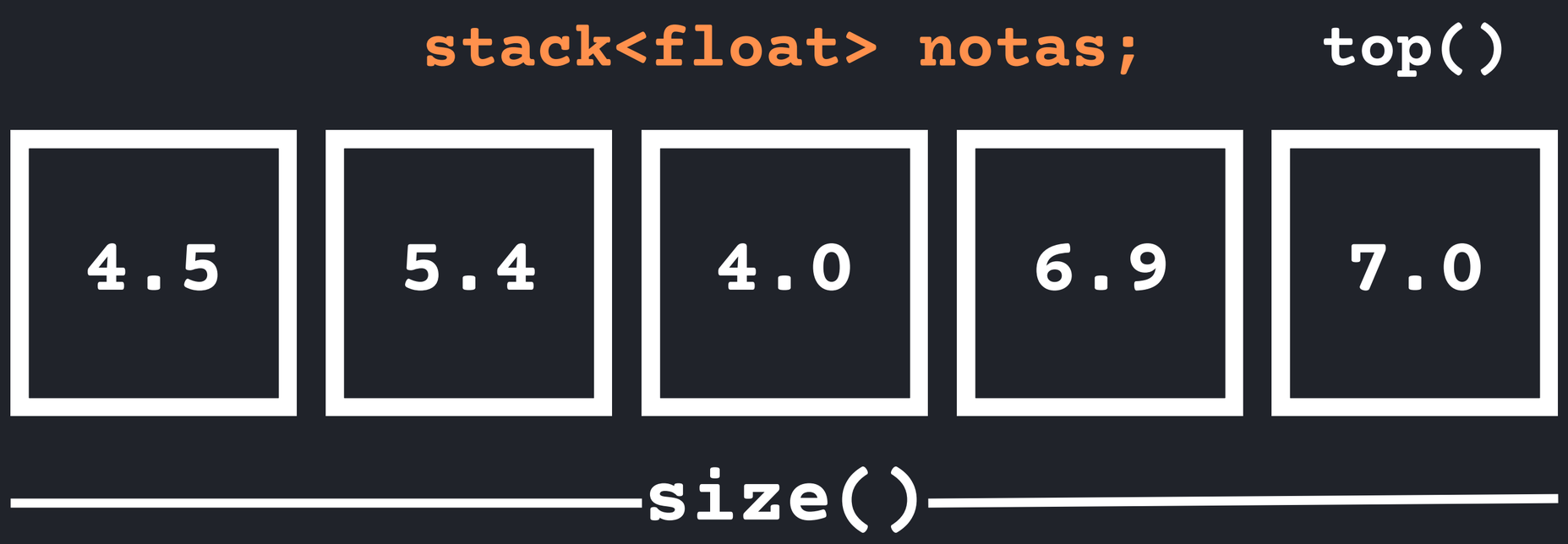
top(): Devuelve el último elemento.

size(): Retorna el tamaño de la pila.

swap(stack): Intercambia valores de una pila a otra pila dada.

empty() : Retorna true si la pila está vacía.

Funciones de stack



notas.size() -> 5

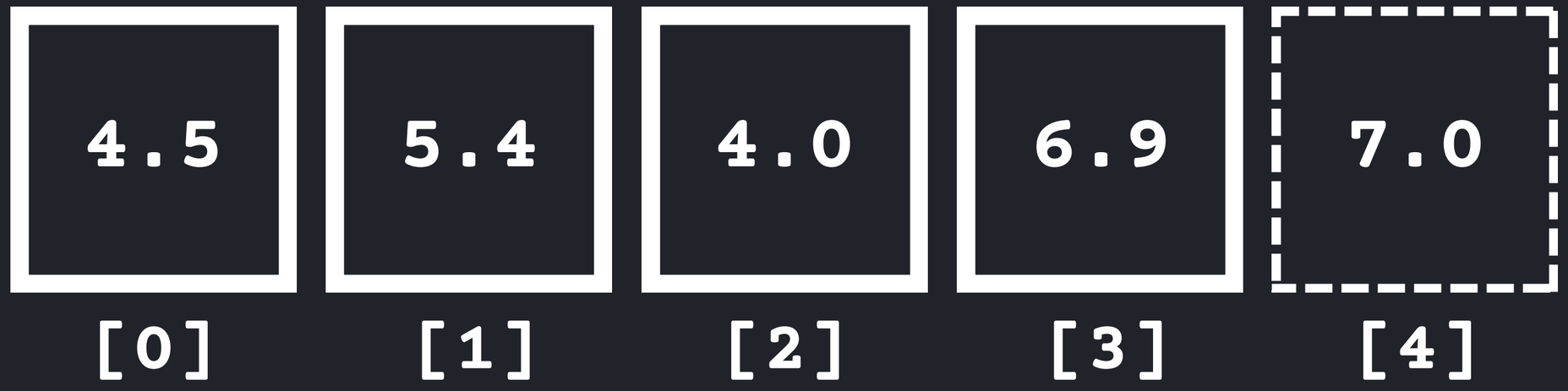
Funciones de stack

```
stack<float> notas;
```


```
notas.push(1.0);
```

Funciones de stack

```
stack<float> notas;
```




```
notas.pop();
```

Recorrer un stack o queue

Con stack y queue no tenemos iteradores como tal como en los vectores, solo tenemos las funciones push y pop, que realizan solo la acción de añadir y eliminar.

Estas funciones realizan una “copia” del elemento que se añada, por lo cual no podemos recorrer el stack y el queue de la misma manera que un vector, por ejemplo.

Recorrer un stack

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

int main() {
    stack<int> x;
    x.push(10);
    x.push(20);
    x.push(30);
    x.push(40);
    // Creamos una copia del stack, como un auxiliar
    stack<int> aux = x;
    cout << "Recorriendo la pila... " << endl;

    while (!aux.empty()) {  // Mientras auxiliar tenga elementos...
        cout << aux.top() << endl;  // Imprimimos el elemento de la cima
        aux.pop();  // Y eliminamos el elemento auxiliar...
    }

    return 0;
}
```

Tenemos que crear una copia del stack ya que si empezamos a eliminar elementos de la original pueden verse afectados

```
Recorriendo la pila...
40
30
20
10
```

Recorrer una queue

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

int main() {
    queue<int> x;
    x.push(10);
    x.push(20);
    x.push(30);
    x.push(40);
    // Creamos una copia del queue, como un auxiliar
    queue<int> aux = x;
    cout << "Recorriendo la cola... " << endl;

    while (!aux.empty()) {  // Mientras auxiliar tenga elementos...
        cout << aux.front() << endl;  // Imprimimos el elemento de la cima
        aux.pop();  // Y eliminamos el elemento auxiliar...
    }

    return 0;
}
```

Tenemos que crear una copia de la cola ya que si empezamos a eliminar elementos de la original pueden verse afectados

```
Recorriendo la cola...
10
20
30
40
```

Ejercicio

Después de incontables años, la empresa Blockbuster volvió a existir, por lo cual se le pide que implemente un programa que ordene distintas peliculas dependiendo de su género. Las pelicula son ingresadas a una caja donde están todas desordenadas, ud debe ordenar las películas por género. Existe una clase Película que tiene titulo, género, director, año y precio. La clase Blockbuster tendrá solo stacks para la cajaDesordenada y otros 3 stacks para distintos géneros a elección. Las películas se agregarán directamente a la cajaDesordenada para después usar un método que ordena dependiendo del género.

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

class Pelicula {
private:
    string titulo, genero, director;
    int año, precio;
public:
    Pelicula(string titulo, string genero, string director, int año, int precio) {
        this->titulo = titulo;
        this->genero = genero;
        this->año = año;
        this->director = director;
        this->precio = precio;
    }
    string getGenero() {
        return genero;
    }
    void info() {
        cout << "Titulo: " << titulo << " - Genero: " << genero << " - Director: " << director << " - Año: " << año << " - Precio: " << precio << endl;
    }
};

class Blockbuster {
private:
    stack<Pelicula*> cajaDesordenada;
    stack<Pelicula*> cajaAnimacion;
    stack<Pelicula*> cajaDrama;
    stack<Pelicula*> cajaComedia;
public:
    Blockbuster() {
    }

    void agregarPelicula(Pelicula* pelicula) {
        cajaDesordenada.push(pelicula);
    }
```

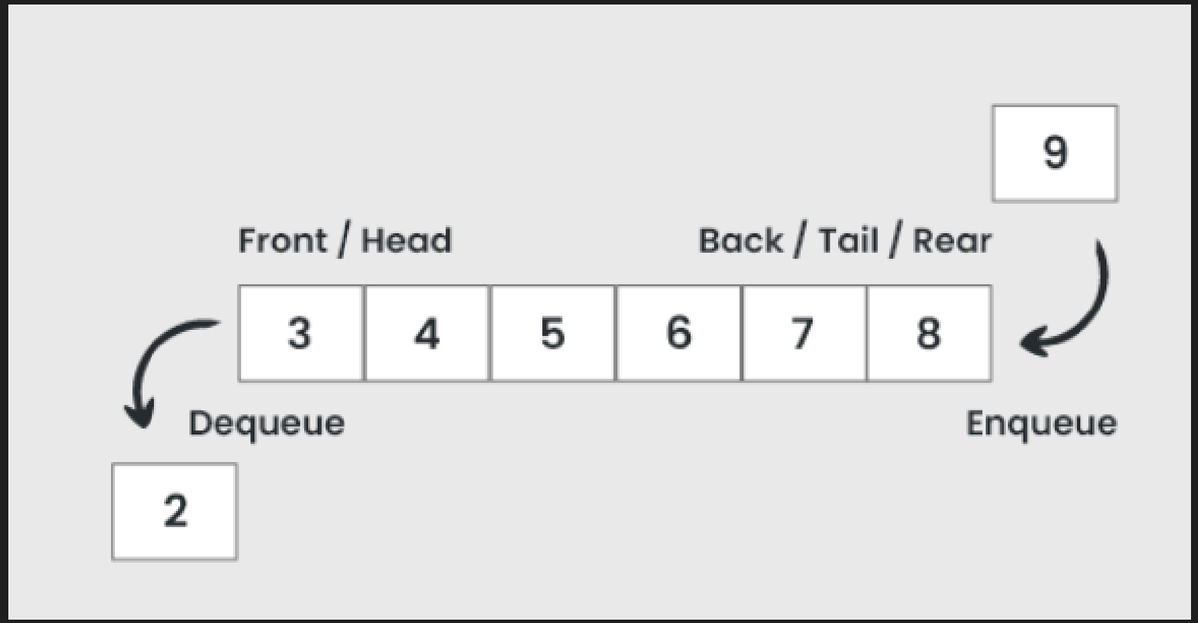
```
    void ordenar() {
        while (!cajaDesordenada.empty()) { // Mientras la caja desordenada no esté vacía
            string genero = cajaDesordenada.top()->getGenero(); // el género será el ultimo elemento añadido a la caja desordenada

            if (genero == "Animación") { // Si el genero es animacion
                cajaAnimacion.push(cajaDesordenada.top()); // Se añade el elemento en la cima de la pila de la caja desordenada a la caja de animación
                cout << "Pelicula Agregada a Animación." << endl;
                cajaAnimacion.top()->info(); // Se muestra la info del elemento añadido a cajaAnimacion
            }
            else if (genero == "Drama") {
                cajaDrama.push(cajaDesordenada.top());
                cout << "Pelicula Agregada a Drama." << endl;
                cajaDrama.top()->info(); // Se muestra la info del elemento añadido a cajaDrama
            }
            else if (genero == "Romance") {
                cajaComedia.push(cajaDesordenada.top());
                cout << "Pelicula Agregada a Romance." << endl;
                cajaComedia.top()->info(); // Se muestra la info del elemento añadido a cajaComedia
            }
            else {
                cout << "Error al ordenar." << endl;
            }
            cajaDesordenada.pop(); // Se elimina el elemento de la caja desordenada, ya que este se copió a la caja que corresponde.
            cout << "-----------------------------------------------------------------------------------------------------" << endl;
        }
        cout << "Se han ordenado todas las peliculas" << endl;
    }
};
```

```cpp
int main() {
    Blockbuster* tienda = new Blockbuster();

    // Agregar peliculas después de la creación del objeto
    tienda->agregarPelicula(new Pelicula("Fallen Angels", "Drama", "Wong Kar-wai", 1995, 15000));
    tienda->agregarPelicula(new Pelicula("Submarine", "Romance", "Richard Ayoade", 2010, 12000));
    tienda->agregarPelicula(new Pelicula("Isle of Dogs", "Animación", "Wes Anderson", 2018, 18000));
    tienda->agregarPelicula(new Pelicula("The End of Evangelion", "Animación", "Hideaki Anno", 1997, 17000));

    tienda->ordenar();

    return 0;
}
```

Queue

El queue es un contenedor de objetos en C++ que sigue la metodología FIFO (First In, First Out). A diferencia de un stack, donde los últimos elementos en entrar son los primeros en salir, en un queue, los primeros elementos en entrar son los primeros en salir. Se puede entender por su traducción literal como "cola", similar a una fila de personas donde el primero en entrar es el primero en ser atendido.

En este ejemplo se muestra como los elementos se van “encolando” y el elemento de al frente es el primero en ser “desencolado”



Funciones

El queue en C++ se debe incluir de esta manera en el código

```
#include <queue>
```

Se declara de esta forma `queue<tipoDeDato> nombre;`

```
queue<int> numeros;
queue<string> palabras;
queue<Objeto*> objetos;
```

push(item): Inserta un ítem al final de la cola

pop(): Elimina el ítem frontal de la cola

front(): Devuelve el primer elemento.

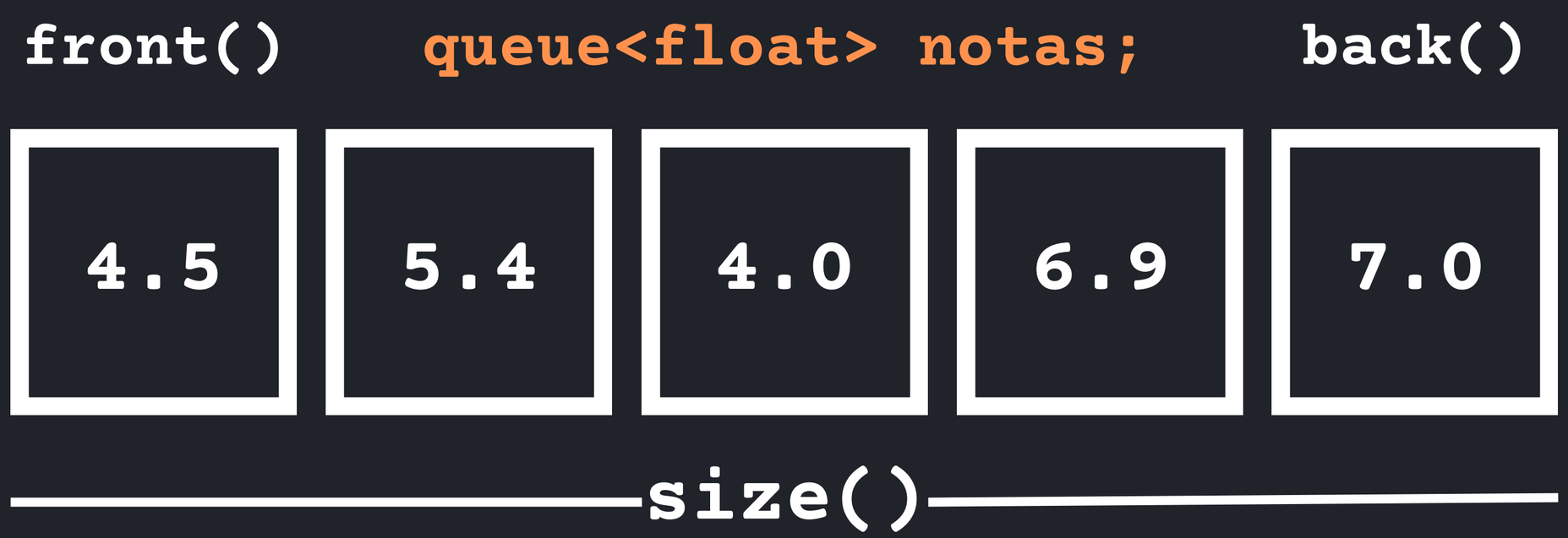
back(): Devuelve el último elemento.

size(): Retorna el tamaño de la cola.

swap(queue): Intercambia valores de una cola a otra cola dada.

empty() : Retorna true si la cola está vacía.

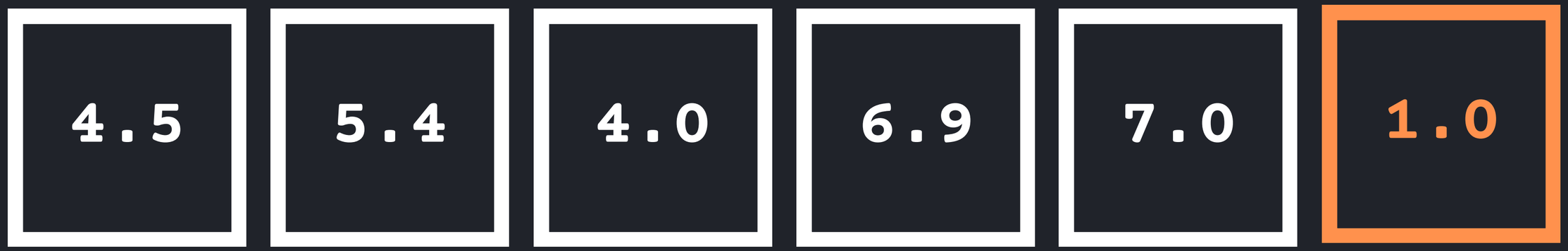
Funciones de queue



notas.size() -> 5

Funciones de queue

```
queue<float> notas;
```




```
notas.push(1.0);
```

Funciones de queue

```
queue<float> notas;
```


```
notas.pop();
```

Ejercicio

Realice un código que maneje la cola de un banco, para lo cual tiene que hacer una cola de clientes. Tiene que realizar los métodos que añadan un cliente a la cola, que se atienda el cliente de la cola, que se muestre el proximo cliente de la cola, y que se pueda mostrar el estado de la cola

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

class Cliente {
private:
    string nombre;
public:
    Cliente(string nombre) {
        this->nombre = nombre;
    }

    string getNombre() {
        return nombre;
    }

    void mostrarInfo() {
        cout << "Cliente: " << nombre << endl;
    }
};

class Banco {
private:
    queue<Cliente*> colaClientes; // Cola de clientes
public:
    void agregarCliente(Cliente* cliente) {
        colaClientes.push(cliente);
        cout << cliente->getNombre() << " ha sido añadido a la cola." << endl;
    }

    void atenderCliente() {
        if (!colaClientes.empty()) {
            cout << "Atendiendo al cliente: ";
            colaClientes.front()->mostrarInfo();
            colaClientes.pop(); // Elimina al cliente al frente de la cola
        } else {
            cout << "No hay clientes en la cola para atender." << endl;
        }
    }
```

```
    void mostrarEstadoCola() {
        if (colaClientes.empty()) {
            cout << "No hay clientes en la cola." << endl;
            return;
        }

        cout << "Clientes restantes en la cola:" << endl;
        queue<Cliente*> tempQueue = colaClientes; // Copia temporal de la cola para mostrar los elementos
        while (!tempQueue.empty()) {
            tempQueue.front()->mostrarInfo();
            tempQueue.pop();
        }
    }
};

int main() {
    Banco banco;

    // Crear y agregar clientes
    banco.agregarCliente(new Cliente("Vicente"));
    banco.agregarCliente(new Cliente("Maicol"));
    banco.agregarCliente(new Cliente("Josefina"));
    banco.agregarCliente(new Cliente("Pepe"));

    // Mostrar el próximo cliente a ser atendido
    banco.mostrarProximoCliente();

    // Atender al primer cliente
    banco.atenderCliente();

    // Mostrar el próximo cliente después de atender uno
    banco.mostrarProximoCliente();

    // Mostrar el estado de la cola
    banco.mostrarEstadoCola();

    return 0;
}
```

Contacto

+569 86031881

benjamin.aceituno@mail.udp.cl

benja.mp4

https://github.com/benjamp4/prog.av-2024-2