Ayudantía n°11

+Repaso

Diego Mena

Mapas

Un mapa contiene dos tipos de datos, uno el cual funcionara como llave para poder acceder al otro tipo de dato, esto se puede observar de la siguiente manera:

LLave	**LLave**	**LLave**
Cont.	**Cont.**	**Cont.**

```
#include <map>
//Tipo de elemento=TD

map<TD1, TD2>Nombre;
map<string, int>Nombre;
map<string,
stack<Int>>Nombre;

Nombre[LLave]=Contenido;
Nombre[string]=Int;
```

Funciones

- Añadir algo al mapa: map[llave] = valor;
- begin(): Retorna el iterador de la primera posición.
- end(): Retorna el iterador antes de la última posición.
- empty(): Devuelve true si el mapa está vacío.
- size(): Devuelve el tamaño del mapa.
- count(llave): Retorna true si la llave existe en el mapa, false en caso contrario.
- find(llave): Retorna un iterador al elemento con la clave especificada.
- erase(llave): Borra el contenido de la llave.
- clear(): Borra todos los elementos del mapa.
- at(llave): Retorna el valor asociado a la llave; lanza una excepción si la llave no existe.
- swap(map): Intercambia el contenido de dos mapas

¿Como podemos recorrer un map?

```
50      void MostrarTodosConIterador()
51      {
52          //map<int, string> Compas;
53          if (Compa.empty())
54          {
55              cout << "No hay compas registrados. :c" << endl;
56          }
57          else
58          {
59              map<string, int>::iterator rec;
60
61              for (rec = Compas.begin(); rec != Compas.end(); rec++)
62              {
63                  cout << "Rut: " << rec->first << " Nombre: " << rec->second << endl;
64              }
65          }
66      }
```

Stack (LIFO)

Para hacer uso de este debemos poner:

```
#include <stack>
stack<tipo_de_dato>Nombre;
```

- push(item): Inserta un elemento al final del stack.
- pop(): Elimina el ultimo elemento.
- top(): Devuelve el ultimo elemento.
- empty(): Devuelve true si la cola está vacía y false si no la esta.
- size(): Devuelve el tamaño del queue.
- swap(stack): Intercambia los valores de una pila a otra.

Nombre.push(4);

Nombre.pop();

Nombre.front();

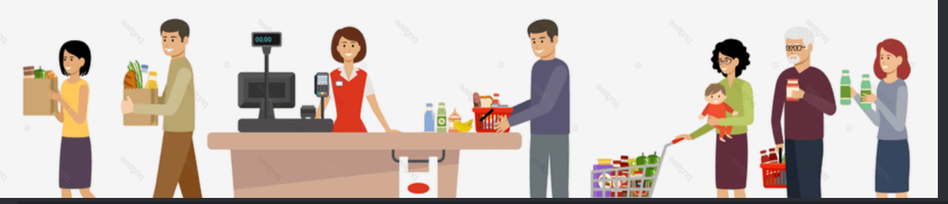
Salida: 4

Nombre.empty();

Salida: true

¿Como podemos recorrer un map con un stack?

```
31          map<string, stack<string> > ::iterator rec;
32          for(rec=Pepe.begin(); rec != Pepe.end(); rec++)
33          {
34              int i=0;
35              cout<<"Animal: "<<rec->first<<endl;
36              stack<string>aux = rec->second;
37              while (aux.empty()!=true)
38              {
39
40                  cout<<"Nombre "<<i+1<<": "<<aux.top()<<endl;
41                  aux.pop();
42                  i=i+1;
43              }
44          }
```



Queue (FIFO)

Para hacer uso de este debemos poner:

```
#include <queue>
queue<tipo_de_dato>Nombre;
```

- push(item): Inserta un elemento al final del queue.
- pop(): Elimina el elemento del enfrente.
- front(): Devuelve el primer elemento.
- back(): Devuelve el último elemento.
- empty(): Devuelve true si la cola está vacía y false si no la esta.
- size(): Devuelve el tamaño del queue.
- swap(queue): Intercambia los valores de una cola a otra.

Nombre.push(4);

1	2	3	4

Nombre.pop();

~~1~~	2	3	4

Nombre.front();

1	2	3	4

Salida: 1

Nombre.empy();

Salida: true

¿Como podemos recorrer un queue?

```cpp
#include <iostream>
#include <queue>
using namespace std;
int main()
{
    int cantidad;
    queue<int>Ejemplo;
    cout<<"Ingrese la cant. de numeros a anadir"<<endl;
    cin>>cantidad;
    for(int i=0;i<cantidad;i++)
    {
        int numero;
        cout<<"Ingrese un numero a anadir"<<endl;
        cin>>numero;
        Ejemplo.push(numero);
    }

    queue<int>Aux=Ejemplo;
    cout<<"Los numeros en orden de la lista es el siguiente: "<<endl;
    while(Aux.empty()!=true)
        {
        cout<<Aux.front()<<"//";
        Aux.pop();

        }
    }
```

Normal

```cpp
    for (int i = 0; i < cantidad; i++)
    {
        queue<int> Aux = Vector[i];
        int total = 0;
        cout<<"-----------------------------------------------------------------------------------------"<<endl;
        cout << "Los numeros en la suma " << i + 1 << " son: ";
        while (Aux.empty()!=true)
        {
            cout<<Aux.front()<<"//";
            total += Aux.front();
            Aux.pop();
        }
        cout << " => Total: " << total << endl;
    }
}
```

Vector con queue

Ejercicio

El sistema debe permitir agregar películas, buscar una película por su nombre y mostrar todas las películas almacenadas.

Pelicula: Es la clase principal. Sus atributos son: map<string, float> peliculas,y su funciones son

Métodos de la clase Pelicula:

- agregarPelicula(string nombre, float calificacion): Permite agregar una película con su calificación.
- imprimirPeliculas(): Muestra todas las películas almacenadas junto con sus calificaciones.
- buscarxNombre(string nombre): Permite buscar una película por su nombre y mostrar su calificación si es encontrada.

El sistema debe permitir que el usuario ingrese varias películas, imprima la lista de películas con sus calificaciones y busque una película por su nombre.

Ejercicio herencia con vector

El sistema debe permitir agregar solicitudes de empleo, atender solicitudes pendientes, mostrar las solicitudes actuales y visualizar un historial de candidatos atendidos.

GestionSolicitudes: Es la clase principal. Sus atributos son:

queue<string> solicitudes: Cola para gestionar las solicitudes pendientes.
vector<string> historialAtendidos: Vector para almacenar el historial de candidatos atendidos.

Métodos de la clase GestionSolicitudes:

- agregarSolicitud(string candidato): Permite añadir una nueva solicitud de empleo al sistema.
- atenderSolicitud(): Procesa la siguiente solicitud pendiente, eliminándola de la cola y añadiéndola al historial.
- mostrarSolicitudesPendientes(): Muestra las solicitudes pendientes en la cola.
- mostrarHistorialAtendidos(): Muestra el historial de todos los candidatos atendidos hasta el momento.

El sistema debe permitir que el usuario agregue solicitudes de empleo, atienda las solicitudes de manera secuencial, revise las solicitudes pendientes y consulte el historial de candidatos atendidos.

Contacto {

+56 9 6070 8865

diego.mena2@mail.udp.cl

https://github.com/Dmena1/Ayudantia-prograAvanzada

}