Ayudantía n°10

Maps y Stack(pilas)

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          }
```

Ejercicio

En una maraton de películas, se necesita organizar una colección de títulos clasificados por género con un metodo de Stack. A continuación, se detallan los requisitos del sistema.
El programa debe incluir las siguientes funcionalidades:

1. Clase Película:
    - Contendrá los atributos:
        - nombre (string)
        - duracion (int)
    - Métodos get’s correspondientes.
2. Clase Pila:
    - Atributos:
        - Un mapa (map<string, stack<Pelicula*>>) que almacena pilas de películas organizadas por género.
    - Métodos:
        - anadirGenero(string genero): Permite añadir un nuevo género a la lista.
        - anadirPeliculaxGenero(string genero): Permite agregar varias películas a un género específico.
        - ImprimirXgenero: Muestra todas las películas de cada género, indicando su nombre y duración.
        - CantidadXGenero: Muestra la cantidad de películas almacenadas en cada género.
        - Eliminarpelicula (string genero, string nombre): Elimina una película específica de un género dado.

Se debe hacer un menu.

Ejercicio herencia con vector

La clase base será Libro, que tendrá los atributos int codigo, string titulo, string autor y int anho. Además, debe contar con un método virtual void imprimir(), que muestra los detalles del libro.

La clase Novela heredará de Libro y añadirá el atributo string genero, tambien necesita de una función imprimir() para mostrar los detalles de la novela, incluyendo el género.

La clase CienciaFiccion, que hereda de Novela, añadirá un atributo string universo. tambien necesita de una función imprimir() para incluir todos los detalles de la novela, incluyendo el género y el universo.
La clase Historia hereda directamente de Libro y tendrá el atributo string periodo. tambien necesita de una función imprimir() para mostrar los detalles, incluyendo el periodo histórico.

Por último, la clase Biblioteca gestionará los libros. Tendrá como atributos un vector<Libro*> listaLibros y un int maxLibros, que define el número máximo de libros que la biblioteca puede almacenar. La clase contará con el método agregarLibro(Libro* libro), que añade un libro a la biblioteca si no se ha alcanzado el límite, y el método imprimir(), que muestra los detalles de todos los libros almacenados utilizando el método imprimir() de cada uno.

Contacto {

+56 9 6070 8865

diego.mena2@mail.udp.cl

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

}