Easy find

El ejercicio propuesto, titulado "Easy find", tiene como objetivo que practiques el uso de plantillas de funciones (function templates) y la Biblioteca Estándar de Plantillas (STL) en C++98, específicamente enfocándote en los contenedores estándar y los algoritmos asociados.

Descripción del ejercicio:

Debes escribir una plantilla de función llamada easyfind que acepte un tipo T y tome dos parámetros:

1. El primero es de tipo T, que se asume es un contenedor de enteros.
2. El segundo es un entero que representa el valor a buscar dentro del contenedor.

La función debe encontrar la primera ocurrencia del segundo parámetro en el primer parámetro. Si no se encuentra ninguna ocurrencia, puedes optar por lanzar una excepción o devolver un valor de error de tu elección.

Propósito de aprendizaje:

Este ejercicio tiene como objetivo que te familiarices con:

- **Plantillas de funciones**: Te permiten escribir funciones genéricas que operan con diferentes tipos de datos.
- **Contenedores estándar de la STL**: Como vector, list, map, entre otros, que almacenan colecciones de datos.
- **Algoritmos de la STL**: Funciones genéricas como find que operan sobre rangos de elementos en contenedores.

Cómo abordarlo en C++98:

1. **Plantillas de funciones**: En C++98, las plantillas permiten crear funciones genéricas. Para este ejercicio, definirás una plantilla de función que acepte un contenedor de enteros y un entero a buscar.
2. **Contenedores de la STL**: Aunque el ejercicio no especifica un tipo de contenedor particular, se espera que utilices los contenedores estándar de la STL, como std::vector o std::list. Estos contenedores almacenan elementos y proporcionan métodos para acceder y manipular dichos elementos.
3. **Algoritmo find de la STL**: La STL ofrece el algoritmo find en el encabezado <algorithm>, que busca un valor en un rango de elementos. Este algoritmo es ideal para este ejercicio, ya que puedes usarlo para buscar la primera ocurrencia del entero en el contenedor.
4. **Manejo de excepciones**: Si el valor no se encuentra, puedes optar por lanzar una excepción. En C++98, las excepciones se manejan utilizando bloques try, catch y throw.

Recursos recomendados:

A continuación, te proporciono una selección de recursos en español que están actualmente disponibles en internet y que te ayudarán a comprender mejor los conceptos necesarios para abordar el ejercicio "Easy find":

1. **"Curso de C++: Plantillas de Funciones"**

   Este video ofrece una explicación detallada sobre las plantillas de funciones en C++, incluyendo su sintaxis y aplicaciones prácticas.

   Ver video

2. **"Curso de C++ Avanzado: Contenedores de la STL"**

   En este video se exploran los diferentes contenedores disponibles en la Standard Template Library (STL) de C++, como vector, list y map, y cómo utilizarlos eficazmente.

   Ver video

3. **"Curso de C++ Avanzado: Algoritmos de la STL"**

   Este recurso se centra en los algoritmos que ofrece la STL, incluyendo cómo aplicarlos a los contenedores para realizar operaciones comunes de manera eficiente.

   Ver video

4. **"Plantillas y Sobrecarga de Funciones en C++"**

   Este video aborda la creación y uso de plantillas en C++, así como la sobrecarga de funciones, proporcionando ejemplos prácticos para su implementación.

   Ver video

5. **"Primeros Pasos con la Standard Template Library de C++"**

   Una introducción a la STL de C++, destacando sus características principales y cómo empezar a utilizarla en tus proyectos.

   Ver video

Estos recursos te proporcionarán una comprensión sólida de las plantillas de funciones, los contenedores y los algoritmos de la STL en C++98, fundamentales para resolver el ejercicio propuesto.

El código

Explicación detallada del código

Este código en C++ implementa una función genérica easyfind, que busca un número entero dentro de un contenedor (por ejemplo, un std::vector<int>). Si encuentra el número, devuelve true; si no lo encuentra, lanza una excepción personalizada.

A continuación, se explica **cada parte del código en detalle**, siguiendo el estándar **C++98**.

1 Archivo easyfind.hpp (Encabezado)

Este archivo declara la función easyfind y define una excepción personalizada para manejar el caso en el que no se encuentre el valor en el contenedor.

Explicación línea por línea:

```
#pragma once
```

- **Evita la inclusión múltiple** del mismo archivo en un programa.

```
#include <algorithm>
#include <exception>
#include <iostream>
#include <string>
#include <vector>
```

- **Librerías utilizadas**:
  - <algorithm>: Permite el uso del algoritmo std::find.
  - <exception>: Necesaria para la clase std::exception usada en la excepción personalizada.
  - <iostream>: Para entrada/salida estándar con std::cin y std::cout.
  - <string>: No se usa directamente en este código, pero podría ser útil para manejar excepciones con cadenas.
  - <vector>: Para usar std::vector<int> en la búsqueda.

```
#define BLUE "\033[1;34m"
#define GREEN "\033[1;32m"
```

```
#define RED "\033[1;31m"
#define RESET "\033[0m"
```

- **Define colores ANSI para la terminal**:
  - BLUE: Azul (para imprimir el vector).
  - GREEN: Verde (para mostrar éxito).
  - RED: Rojo (para mostrar errores).
  - RESET: Restaura el color original de la terminal.

```
class EasyFindException : public std::exception {

        public:

        const char *what() const throw() {
                return "Error: Value not found in container 👹 ";
        }
};
```

- **Define una excepción personalizada (EasyFindException)** que hereda de std::exception.
- Sobrescribe el método what() para devolver un mensaje de error cuando el número no se encuentra.

```
#include "easyfind.tpp"
```

- **Incluye la implementación de easyfind**. En C++98, es común definir **plantillas en archivos .tpp** para mantener el código más organizado.

```
template <typename T>
bool easyfind(T &container, int toFind);
```

- **Declaración de la función easyfind**.
- Es una **función plantilla** (template), lo que permite que T sea cualquier tipo de contenedor (siempre que tenga .begin() y .end()).

2 Archivo easyfind.tpp (Implementación de easyfind)

Aquí se encuentra la implementación de easyfind.

#include "easyfind.hpp"

- **Incluye el encabezado** para asegurarse de que la función está correctamente declarada antes de implementarla.
  template <typename T>
  bool easyfind(T& container, int toFind) {

- **Define la función plantilla easyfind**, que acepta:
  - container: Un contenedor genérico (por ejemplo, std::vector<int>).
  - toFind: El número entero a buscar dentro del contenedor.

  if (std::find(container.begin(), container.end(), toFind) == container.end()) {

- **std::find(container.begin(), container.end(), toFind)** busca toFind dentro del contenedor.
- Si el iterador de retorno es igual a container.end(), significa que **el valor no fue encontrado**.
  throw EasyFindException();
- **Lanza la excepción EasyFindException** si el número no se encuentra en el contenedor.
  else
  return true;
- **Si el número fue encontrado**, la función devuelve true.

3 Archivo main.cpp (Función principal)

Este archivo contiene la función main(), que prueba la funcionalidad de easyfind.

#include "easyfind.hpp"

using std::cerr; using std::cout; using std::endl;

- **Incluye easyfind.hpp** para usar la función y la excepción personalizada.
- **Usa using** para evitar escribir std::cout, std::cerr y std::endl en cada línea.

int main() {

- **Función principal** donde se ejecutará el programa.

std::vector<int> vec;

- **Declara un std::vector<int>** vacío.

```
cout << BLUE << "\nVector: { ";
for (int i = 0; i < 10; i++) {
        vec.push_back(i);
        cout << i << (i < 9 ? ", " : " }");
}
cout << RESET << endl;
```

- **Llena el vector con los números del 0 al 9** usando push_back(i).
- **Imprime el contenido del vector en color azul (BLUE)**.

```
int j;
cout << "\nSelect number to find in vector 🕵️ : ";
std::cin >> j;
cout << endl;
```

- **Solicita al usuario un número entero (j)** para buscar en el vector.

```
try {
        if (easyfind(vec, j))
                cout << GREEN << "Perfect! " << RESET << "Value " << j << " found in vector 👌 " << endl;
} catch (const std::exception &e) {
        cerr << RED << e.what() << RESET << endl;
}
```

- **Usa un bloque try-catch** para manejar excepciones.
- **Llama a easyfind(vec, j)**:
  - Si el número está en el vector, imprime un mensaje en verde (GREEN).
  - Si el número no está, **se lanza EasyFindException** y se captura en catch, mostrando el mensaje en rojo (RED).

```
return 0;
```

- **Termina la ejecución del programa correctamente**.

Resumen del flujo de ejecución

1. Se crea un std::vector<int> y se llena con valores del 0 al 9.
2. Se muestra el contenido del vector.

3. Se solicita al usuario un número (j) para buscar en el vector.
4. Se llama a easyfind(vec, j), que:
    a. Usa std::find para buscar j en vec.
    b. Si lo encuentra, devuelve true.
    c. Si no lo encuentra, **lanza una excepción**.
5. En main(), se maneja el resultado:
    a. Si easyfind retorna true, muestra un mensaje en **verde** indicando éxito.
    b. Si se lanza una excepción, muestra un mensaje en **rojo** de error.

Conceptos clave usados en C++98

- **Plantillas de funciones (template)**

  Permiten escribir código genérico para trabajar con cualquier tipo de contenedor.

- **Contenedores de la STL (std::vector)**

  Un tipo de estructura de datos flexible que permite almacenar elementos dinámicamente.

- **Algoritmos de la STL (std::find)**

  Permite buscar un elemento en un rango sin necesidad de escribir un bucle manual.

- **Manejo de excepciones (try-catch, throw)**

  Permite gestionar errores sin detener la ejecución del programa.

Conclusión

Este código implementa una **búsqueda eficiente en un contenedor usando std::find**, junto con una **excepción personalizada** para manejar casos donde el valor no se encuentra.