ITER

El ejercicio propuesto consiste en implementar una plantilla de función llamada iter que reciba tres parámetros y no retorne ningún valor. A continuación, se detallan los componentes del ejercicio:

Descripción del ejercicio:

- **Nombre de la función:** iter
- **Parámetros:**
  - La dirección de un array.
  - La longitud del array.
  - Una función que será llamada en cada elemento del array.
- **Requisitos adicionales:**
  - La función iter debe ser una plantilla que funcione con cualquier tipo de array.
  - El tercer parámetro puede ser una función plantilla instanciada.
- **Archivos a entregar:**
  - Makefile
  - main.cpp
  - iter.h o iter.hpp
- **Funciones prohibidas:** Ninguna

Propósito de aprendizaje:

Este ejercicio tiene como objetivo principal familiarizar al estudiante con las plantillas de funciones en C++98, una característica que permite escribir funciones genéricas que operan con diferentes tipos de datos. Al completar este ejercicio, el estudiante:

- Comprenderá cómo declarar y definir plantillas de funciones.
- Aprenderá a aplicar funciones a cada elemento de un array utilizando punteros y plantillas.
- Se familiarizará con la creación y uso de archivos de cabecera (.h o .hpp) para declarar plantillas.
- Desarrollará habilidades para organizar proyectos en C++ utilizando un Makefile para la compilación.

Cómo abordar el ejercicio en C++98:

Para implementar la función iter según el estándar C++98, es necesario comprender varios conceptos clave:

1. **Plantillas de funciones (Function Templates):**
   a. **Definición:** Permiten crear funciones genéricas que operan con diferentes tipos de datos.
   b. **Sintaxis básica:** template <typename T>
   void funcion(T parametro) {
   // Implementación

```
}
```

   c. **Aplicación en iter:** La función iter debe ser una plantilla que pueda aceptar arrays de cualquier tipo.
2. **Punteros a arrays:**
   a. **Definición:** Un puntero que apunta al primer elemento de un array.
   b. **Uso en iter:** El primer parámetro de iter es un puntero al tipo genérico T, representando la dirección del array.
3. **Funciones como parámetros:**
   a. **Definición:** En C++98, es posible pasar funciones como argumentos a otras funciones.
   b. **Sintaxis para funciones que retornan void y aceptan un parámetro de tipo T:** void nombreFuncion(T);

   c. **Uso en iter:** El tercer parámetro es una función que se aplicará a cada elemento del array.
4. **Iteración sobre arrays:**
   a. **Concepto:** Recorrer cada elemento del array para aplicar la función proporcionada.
   b. **Implementación:** Utilizar un bucle for que itere desde 0 hasta n - 1, donde n es la longitud del array.
5. **Archivos de cabecera (.h o .hpp):**
   a. **Propósito:** Declarar la interfaz de funciones y plantillas para que puedan ser utilizadas en múltiples archivos fuente.
   b. **Contenido típico:** Declaraciones de funciones, plantillas y definiciones de tipos.
   c. **Uso en iter:** Declarar la plantilla de la función iter en un archivo de cabecera para su inclusión en main.cpp.
6. **Makefile:**
   a. **Definición:** Archivo que define cómo se compilan y enlazan los programas en C++.
   b. **Contenido típico:** Reglas que especifican cómo generar los archivos objeto y el ejecutable final.
   c. **Uso en el ejercicio:** Crear un Makefile que compile main.cpp y cualquier otro archivo necesario, generando el ejecutable correspondiente.

Recursos recomendados:

Para profundizar en los conceptos mencionados y facilitar la realización del ejercicio, se recomiendan los siguientes videos en español:

1. **"¿Qué es una plantilla de función en C++?"**

   Este video ofrece una explicación detallada sobre las plantillas de funciones en C++, incluyendo su definición y ejemplos prácticos de implementación.

   Ver video

2. **"Funciones de CallBack en C++ || Curso C++ Cap. #11"**

   En este capítulo del curso de C++, se aborda cómo pasar funciones como parámetros a otras funciones, un concepto clave para el ejercicio propuesto.

Ver video

3. **"Tutorial de C++ en Español -29- Pasando Arreglos a Funciones"**

   Este tutorial explica cómo pasar arrays como parámetros a funciones en C++, lo cual es esencial para la implementación de la función iter.

   Ver video

4. **"116.- Curso C++ No Tan Básico. Pasar funciones como argumentos a otras funciones."**

   Este video profundiza en cómo pasar funciones como argumentos a otras funciones en C++, proporcionando ejemplos claros y concisos.

   Ver video

5. **"Plantillas en c++ /** Explicación e Implementación*/"**
   Este video ofrece una explicación e implementación de plantillas en C++, incluyendo funciones plantilla, lo cual es relevante para el ejercicio.

   Ver video

Estos recursos deberían proporcionarte una comprensión sólida de los conceptos necesarios para abordar el ejercicio de implementar la función plantilla iter en C++98.

El código

El código presentado consta de dos archivos principales:

1. **iter.hpp**: Define una función de plantilla (iter) que recorre un array y aplica una función a cada elemento.
2. **main.cpp**: Prueba la funcionalidad de iter con diferentes tipos de datos y funciones.

El propósito de este código es demostrar el uso de **templates (plantillas)** en C++ para iterar sobre arrays de distintos tipos y aplicar funciones sobre sus elementos.

1. Análisis de iter.hpp

Este archivo contiene:

1.1 Inclusión de Bibliotecas

```
#include <cstdlib>          // Para la definición de size_t
#include <iostream>         // Para std::cout y std::endl
```

```
#include <stdint.h>          // Para tipos de datos enteros como uint32_t
#include <string>            // Para usar el tipo std::string
```

Se incluyen estas librerías para proporcionar funcionalidades básicas necesarias en el código.

1.2 Definición de Macros

```
#define GREEN "\033[0;32m"       // Código ANSI para imprimir texto en verde
#define RESET "\033[0m"    // Código ANSI para resetear el color de la terminal
```

Estas macros se utilizan para dar formato de color a la salida en la terminal.

1.3 Definición de Funciones de Plantilla

```
template <typename T>
void addOne(T& x) { x++; }
```

Esta función de plantilla **incrementa el valor** de la variable x en 1.

```
template <typename T>
void print(const T& x) { std::cout << x << std::endl; }
```

Esta función de plantilla **imprime** un valor en la terminal.

1.4 Sobrecarga de la Función iter

```
template <typename T>
void iter(T* array, size_t size, void (*f)(T&)) {
   for (size_t i = 0; i < size; i++)
      f(array[i]);
}
```

Esta versión de iter permite modificar los elementos de un array, ya que f recibe T& (referencia modificable).

```
template <typename T>
void iter(const T* array, size_t size, void (*f)(const T&)) {
   for (size_t i = 0; i < size; i++)
      f(array[i]);
}
```

Esta segunda versión maneja **arrays constantes**, permitiendo solo funciones que no modifiquen los elementos (const T&).

2. Análisis de main.cpp (Desarrollo Propio)

```
#include "iter.hpp"

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

#define arraySize 3
```

Aquí se incluye el archivo de cabecera iter.hpp y se define el tamaño del array con #define.

2.1 Declaración del Array

```
int arrayInt[] = { 1, 10, -100 };
```

Se crea un array de enteros con tres elementos.

2.2 Impresión del Array Antes de la Iteración

```
cout << "\n";
cout << GREEN << "arrayInt before iter:" << RESET << endl;
iter(arrayInt, arraySize, &::print<int>);
```

Llama a iter pasando arrayInt, su tamaño y la función print<int>, lo que imprimirá el contenido inicial del array.

2.3 Modificación del Array

```
::iter(arrayInt, arraySize, &::addOne<int>);
```

Llama a iter con la función addOne<int>, lo que **incrementará cada elemento del array en 1**.

2.4 Impresión del Array Después de la Iteración

```
cout << "\n";
cout << GREEN << "arrayInt after iter:"<< RESET << endl;
::iter(arrayInt, arraySize, &::print<int>);
```

Imprime nuevamente los valores del array para mostrar los cambios realizados por addOne.

3. Análisis de main.cpp (Proporcionado)

3.1 Definición de la Clase Awesome

```
class Awesome {
  public:
    Awesome( void ) : _n( 42 ) { return; }
    int get( void ) const { return this->_n; }
  private:
    int _n;
};
```

- Esta clase tiene un **único atributo _n** que siempre es **42**.
- Incluye un método get() que devuelve el valor de _n.

3.2 Sobrecarga del Operador <<

```
std::ostream & operator<<( std::ostream & o, Awesome const & rhs ) {
  o << rhs.get();
```

```
  return o;
}
```

Permite imprimir objetos Awesome usando std::cout, mostrando el valor de _n.

3.3 Nueva Versión de print

```
template< typename T >
void print( T& x ) {
  std::cout << x << std::endl;
  return;
}
```

Esta versión de print imprime cualquier tipo de dato.

3.4 Prueba de iter en main

```
int main() {
  int tab[] = { 0, 1, 2, 3, 4 };
  Awesome tab2[5];

  iter( tab, 5, print<const int> );
  iter( tab2, 5, print<Awesome> );

  return 0;
}
```

Explicación Paso a Paso

1. Se declara un array tab[] de enteros {0, 1, 2, 3, 4}.
2. Se declara un array tab2[] de objetos Awesome (donde todos los elementos tienen _n = 42).
3. Se llama a iter(tab, 5, print<const int>) → Imprime {0, 1, 2, 3, 4}.
4. Se llama a iter(tab2, 5, print<Awesome>) → Imprime 42 cinco veces.

4. Resumen de los Puntos Clave

1. **Plantillas de Función (template <typename T>)**

   Se usan para definir print, addOne e iter de manera genérica.

2. **Gestión de Arrays Mediante Función iter**
   a. iter recorre el array y aplica una función a cada elemento.
   b. Se sobrecarga iter para manejar arrays constantes y no constantes.
3. **Manejo de Punteros a Función**

   iter recibe un **puntero a función** para aplicar sobre cada elemento del array.

4. **Uso de Clases Personalizadas**

   La clase Awesome se usa para mostrar cómo iter puede manejar tipos de datos complejos.

5. **Sobrecarga de Operadores (<<)**

   Permite imprimir instancias de Awesome con std::cout.

5. Conclusión

Este código es un excelente ejemplo de **plantillas en C++** y el uso de **punteros a función**. La función iter es flexible y permite trabajar con cualquier tipo de dato, asegurando que las operaciones sean seguras y eficientes.