Harl filter

El ejercicio consiste en implementar un programa llamado harlFilter que filtra los mensajes de Harl según el nivel de registro (log level) que el usuario quiere escuchar. Harl tiene cuatro niveles de mensajes: DEBUG, INFO, WARNING y ERROR, y el programa debe mostrar los mensajes correspondientes al nivel solicitado y todos los niveles superiores.

Funcionamiento esperado:

- El programa toma como parámetro un nivel de registro (como WARNING o ERROR).
- Muestra todos los mensajes correspondientes al nivel indicado y a los niveles más altos (por ejemplo, si se especifica WARNING, se mostrarán los mensajes de WARNING y ERROR).
- Si el nivel proporcionado no coincide con ninguno de los cuatro niveles definidos, el programa mostrará un mensaje genérico como:

  [Probably complaining about insignificant problems]

Ejemplo de ejecución:

- Comando: ./harlFilter "WARNING" Salida:

  [WARNING]
  I think I deserve to have some extra bacon for free.
  I've been coming for years whereas you started working here since last month.
  [ERROR]
  This is unacceptable, I want to speak to the manager now.

Comando: ./harlFilter "I am not sure how tired I am today..." Salida:

[Probably complaining about insignificant problems]

Cómo afrontarlo:

Para resolver este ejercicio, sigue estos pasos:

1. Organiza los archivos:
   a. Crea los archivos indicados en la carpeta ex06/: Makefile, main.cpp, Harl.{h, hpp}, Harl.cpp.
2. Define la clase Harl:
   a. Crea una clase Harl que contenga métodos para manejar los diferentes niveles de mensajes.
   b. Define un método para cada nivel (debug, info, warning, error) y un método principal para procesar el nivel recibido como entrada.

3. Usa el switch statement:

    a. El ejercicio requiere usar la estructura switch para determinar qué mensajes se deben mostrar según el nivel especificado. Esto significa que debes usar un mapeo o alguna técnica para que los niveles (DEBUG, INFO, etc.) puedan ser procesados dentro del switch.

4. Gestión de entrada:

    a. En main.cpp, recibe el nivel como argumento desde la línea de comandos y pásalo a un método de la clase Harl.
    b. Si el nivel no coincide con los cuatro niveles válidos, muestra el mensaje genérico mencionado.

5. Crea un Makefile:
    a. Asegúrate de que el programa pueda compilarse fácilmente con el comando make.

Resumen del enfoque:

- Identifica los niveles: Define los cuatro niveles (DEBUG, INFO, WARNING, ERROR) como cadenas o constantes.
- Implementa un filtro: Usa un switch para determinar qué mensajes deben mostrarse.
- Manejo de errores: Si el nivel no es válido, incluye un mensaje genérico.
- Prueba: Asegúrate de que el programa responde correctamente para todos los niveles y con entradas inválidas.

Harls.hpp

El código presentado es la definición de la clase Harl en el archivo de encabezado (Harl.hpp). A continuación, se explica cada componente de manera clara y detallada.

1. Estructura General

- Protección de encabezado (#ifndef, #define, #endif):
    - Evita que el archivo sea incluido más de una vez en el programa, lo que podría causar errores de redefinición.
    - Define un identificador único HARL_HPP.

- Inclusión de librerías estándar:

    - #include <cstdlib>: Usado para funciones generales como conversión de cadenas o gestión de memoria.
    - #include <iostream>: Necesario para la salida estándar (std::cout y std::cerr).
    - #include <string>: Proporciona soporte para la clase std::string, usada en los métodos de la clase Harl.

2. Clase Harl

- La clase Harl modela el comportamiento del sistema que procesa y filtra los mensajes según niveles de registro.

  a) Métodos Públicos

  1. Constructor y Destructor:

     a. Harl(void) y ~Harl(void):
        i. Constructor y destructor por defecto, respectivamente.
        ii. Aunque no tienen implementación visible aquí, se definen para garantizar que cualquier inicialización o limpieza necesaria pueda ser realizada.

  2. filterComplaint(std::string level):

     a. Método público que recibe un nivel de registro como argumento (level).
     b. Este método será el punto de entrada para determinar qué mensajes mostrar según el nivel especificado.

  b) Métodos Privados

  - Estos métodos implementan la lógica para cada nivel de registro:

    - debug(void):

      - Gestiona el nivel más bajo (DEBUG) e imprime mensajes relacionados con información de depuración o trivialidades.

    - info(void):

      - Maneja mensajes informativos que son relevantes pero no críticos.

    - warning(void):

      - Genera mensajes que advierten de problemas que requieren atención pero no son errores fatales.

    - error(void):

      - Representa el nivel más alto de gravedad; los mensajes son errores críticos.

3. Definición de Tipos y Constantes

1. typedef void (Harl::* HarlLogFunction)(void);:

   a. Define un alias para un puntero a miembro función de la clase Harl que no toma argumentos ni devuelve valores.
   b. Será útil para llamar dinámicamente a los métodos debug, info, warning y error.

2. Mensajes de Error y Uso:

    a. #define ARGC_ERR y #define USAGE:

        i. Constantes para mensajes que se muestran al usuario cuando la entrada es inválida o cuando se requiere información sobre el uso del programa.

3. Mensajes por Nivel:

    a. DEBUG_MESSAGE, INFO_MESSAGE, WARNING_MESSAGE y ERROR_MESSAGE:
        i. Definen los textos que se deben mostrar para cada nivel de registro.

4. ¿Cómo funciona en conjunto?

El archivo de encabezado proporciona una estructura modular que permite a Harl:

1. Filtrar y mostrar mensajes según el nivel de registro proporcionado.
2. Usar punteros a funciones para organizar la lógica y hacer más eficiente la llamada a los métodos internos.

El comportamiento real se implementará en el archivo Harl.cpp. Este archivo probablemente:

1. Usará el switch en el método filterComplaint para determinar qué método privado (debug, info, etc.) se debe invocar.
2. Hará uso de las constantes definidas aquí para imprimir los mensajes apropiados.

Resumen:

El archivo Harl.hpp define la interfaz para una clase Harl que filtra mensajes de registro según niveles. Su diseño es modular, con métodos privados para cada nivel y constantes predefinidas para los mensajes. Es eficiente y utiliza conceptos avanzados como punteros a funciones, aunque la implementación detallada de la lógica está delegada al archivo fuente correspondiente.

Harld.cpp

El código proporcionado es la implementación de la clase Harl, previamente definida en el archivo Harl.hpp. A continuación, se explica cada componente de este archivo de implementación.

1. Inclusión de Dependencias

```
#include "Harl.hpp"
```

- Se incluye el archivo de encabezado Harl.hpp para utilizar la definición de la clase Harl y las constantes relacionadas.

2. Constructor y Destructor

```
Harl::Harl(void) {};
Harl::~Harl(void) {};
```

- Constructor por defecto (Harl::Harl): No realiza ninguna inicialización específica.
- Destructor (Harl::~Harl): No realiza ninguna limpieza específica.
- Ambos métodos están presentes para mantener la estructura modular de la clase y permitir posibles modificaciones futuras.

3. Métodos Privados

Los métodos privados definen el comportamiento asociado a cada nivel de registro. Cada uno imprime un encabezado (por ejemplo, [DEBUG]) seguido del mensaje correspondiente definido en Harl.hpp.

a) debug

```
void Harl::debug(void) {
    cout << "[ DEBUG ] " << endl;
    cout << DEBUG_MESSAGE << endl;
}
```

- Imprime el encabezado [DEBUG] y el mensaje definido en DEBUG_MESSAGE.

b) info

```
void Harl::info(void) {
    cout << "[ INFO ] " << endl;
    cout << INFO_MESSAGE << endl;
}
```

- Imprime el encabezado [INFO] y el mensaje definido en INFO_MESSAGE.

c) warning

```
void Harl::warning(void) {
    cout << "[ WARNING ] " << endl;
    cout << WARNING_MESSAGE << endl;
}
```

- Imprime el encabezado [WARNING] y el mensaje definido en WARNING_MESSAGE.

d) error

```
void Harl::error(void) {
    cout << "[ ERROR ] " << endl;
    cout << ERROR_MESSAGE << endl;
}
```

- Imprime el encabezado [ERROR] y el mensaje definido en ERROR_MESSAGE.

4. Método Público: filterComplaint

Este es el método principal que implementa la lógica del filtro de mensajes basado en el nivel proporcionado por el usuario.

a) Definición de Arrays

```
HarlLogFunction logFunctions[] = {&Harl::debug, &Harl::info, &Harl::warning, &Harl::error};
std::string logLevels[] = {"DEBUG", "INFO", "WARNING", "ERROR"};
```

- logFunctions: Array de punteros a los métodos de la clase Harl. Permite llamar dinámicamente a los métodos privados (debug, info, etc.).
- logLevels: Array que asocia cada nivel de registro (DEBUG, INFO, etc.) con su posición correspondiente en el array logFunctions.

b) Bucle de Búsqueda

```
for (int i = 0; i < 4; i++) {
  if (level == logLevels[i]) {
    ...
  }
}
```

- El bucle recorre el array logLevels buscando una coincidencia con el nivel proporcionado como argumento (level).
- Si encuentra una coincidencia, ejecuta el bloque switch.

c) switch para Ejecutar los Métodos Correspondientes

```
switch (i) {
  case 0:
    (this->*logFunctions[0])();
    (this->*logFunctions[1])();
    (this->*logFunctions[2])();
    (this->*logFunctions[3])();
    break;
  case 1:
    (this->*logFunctions[1])();
    (this->*logFunctions[2])();
    (this->*logFunctions[3])();
    break;
  case 2:
    (this->*logFunctions[2])();
    (this->*logFunctions[3])();
    break;
  case 3:
    (this->*logFunctions[3])();
    break;
```

```
        default:
            break;
    }
```

- Utiliza un switch basado en el índice (i) del nivel encontrado.
- Según el nivel, llama a los métodos correspondientes desde el índice actual hasta el final del array logFunctions. Esto asegura que se muestren los mensajes del nivel actual y todos los superiores.
- Las llamadas a los métodos se realizan usando punteros a funciones: (this->*logFunctions[x])(); Aquí, x es el índice correspondiente al método (debug, info, etc.).

d) Mensaje por Niveles No Válidos

```
cout << " [ Probably complaining about insignificant problems ] " << endl;
```

- Si el nivel proporcionado no coincide con ninguno de los valores en logLevels, imprime un mensaje genérico indicando que el nivel no es válido.

Funcionamiento Resumido

1. El usuario proporciona un nivel (level) como entrada.
2. filterComplaint busca si el nivel proporcionado coincide con uno de los valores en logLevels.
3. Si encuentra coincidencia:
    a. Usa un switch para ejecutar el método correspondiente y todos los métodos de niveles superiores.
4. Si no encuentra coincidencia:
    a. Muestra un mensaje genérico.

En resumen:

Este archivo implementa la lógica principal para la clase Harl. Filtra mensajes según niveles de registro usando punteros a funciones y un switch eficiente. Su diseño es limpio, modular y fácil de entender, cumpliendo con los requisitos del ejercicio.

main.cpp

El código presentado corresponde al archivo principal del programa, que contiene la función main para ejecutar la lógica de la clase Harl. Este archivo se encarga de procesar los argumentos de la línea de comandos y llamar al método filterComplaint para manejar el nivel de registro.

1. Inclusión de Dependencias

```
#include "Harl.hpp"
```

- Se incluye el archivo Harl.hpp para acceder a la definición de la clase Harl y sus métodos, así como a las constantes predefinidas (ARGC_ERR y USAGE).

2. Espacios de Nombres Usados

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

- **cerr**: Utilizado para mostrar mensajes de error en la salida de error estándar.
- **cout**: Utilizado para imprimir mensajes generales en la salida estándar.
- **endl**: Inserta un salto de línea y asegura el vaciado del búfer de salida.

3. Función main

a) Declaración de un Objeto Harl

```
Harl harl;
```

- Crea una instancia de la clase Harl para utilizar sus métodos.

b) Validación de los Argumentos

```
if (argc != 2 || !argv[1]) {
   cerr << ARGC_ERR << '\n';
   cerr << USAGE << endl;
   return 1;
}
```

- Verifica que el programa reciba exactamente **un argumento adicional** (además del nombre del programa). Esto asegura que el usuario haya especificado un nivel de registro.
- Si la condición no se cumple:
    - Muestra un mensaje de error utilizando ARGC_ERR.
    - Imprime las instrucciones de uso (USAGE) para guiar al usuario.
    - Retorna 1 para indicar un error en la ejecución.

c) Llamada al Método filterComplaint

```
harl.filterComplaint(argv[1]);
```

- Llama al método filterComplaint de la instancia harl, pasando el argumento proporcionado por el usuario (argv[1]) como el nivel de registro.
- Este método procesa el nivel y muestra los mensajes correspondientes.

d) Retorno de Éxito

```
return 0;
```

- Si todo se ejecuta correctamente, retorna 0, indicando una ejecución exitosa.

Funcionamiento Resumido

1. **Validación de Entrada**:

    a. Verifica que el programa reciba exactamente un argumento adicional.
    b. Si no se proporciona el nivel o el número de argumentos es incorrecto, muestra mensajes de error (ARGC_ERR y USAGE) y termina la ejecución.

2. **Llamada a la Lógica Principal**:

    a. Si los argumentos son válidos, utiliza el método filterComplaint para procesar el nivel de registro proporcionado.

3. **Finalización**:

    a. Retorna 0 para indicar éxito o 1 si hubo un error con los argumentos.

Resumen

Este archivo implementa una función main simple y efectiva para manejar el flujo del programa. Se asegura de validar los argumentos antes de invocar el método principal de la clase Harl. El diseño es limpio y modular, siguiendo buenas prácticas como el uso de constantes y salida estándar para errores.