MEGAPHONE

El ejercicio consiste en crear un programa llamado megaphone que tome una cadena de texto como entrada y la imprima en mayúsculas en la salida estándar. Si no se proporciona ninguna entrada, el programa debe imprimir "* LOUD AND UNBEARABLE FEEDBACK NOISE *".

Cómo hacerlo:

1. Incluir la cabecera necesaria:

```
#include <iostream>
#include <cctype>
```

2. Crear la función main:

```
int main(int argc, char **argv) {
        // ...
}
```

   - argc: Contiene el número de argumentos pasados al programa (incluyendo el nombre del programa).
   - argv: Es un arreglo de cadenas de caracteres que contiene los argumentos pasados al programa. argv[0] es el nombre del programa.

3. Manejar los argumentos:

   - Caso 1: No se proporcionan argumentos (argc == 1):

```
if (argc == 1) {
  std::cout << "* LOUD AND UNBEARABLE FEEDBACK NOISE *" << std::endl;
  return 0;
}
```

   - Caso 2: Se proporcionan argumentos (argc > 1):

```
for (int i = 1; i < argc; ++i) {
        // Iterar sobre cada argumento (excepto el nombre del programa)
        for (size_t j = 0; argv[i][j]; ++j) {
          // Convertir cada carácter a mayúscula
          std::cout << static_cast<char>(toupper(argv[i][j]));
        }
        // Imprimir un espacio entre argumentos
        std::cout << " ";
      }
      std::cout << std::endl;
```

Código completo:

```
#include <iostream>
#include <cctype> // Para la función toupper

int main(int argc, char **argv) {
  if (argc == 1) {
    std::cout << "* LOUD AND UNBEARABLE FEEDBACK NOISE *" << std::endl;
    return 0;
  }

  for (int i = 1; i < argc; ++i) {
    for (size_t j = 0; argv[i][j]; ++j) {
      std::cout << static_cast<char>(toupper(argv[i][j]));
    }
    std::cout << " ";
  }
  std::cout << std::endl;

  return 0;
}
```

Explicación:

- El código primero verifica si se proporcionaron argumentos. Si no se proporcionaron, imprime el mensaje de "... feedback noise".
- Si se proporcionaron argumentos, itera sobre cada argumento y luego sobre cada carácter dentro de cada argumento.

¿Por qué el uso de int en el primer bucle y luego en el anidado se utiliza size_t ?

La elección entre int y size_t para los índices de los bucles puede parecer arbitraria a primera vista, pero en realidad hay razones sólidas detrás de esta decisión.

¿Por qué int en el primer bucle?

argc es un int: El argumento argc de la función main es de tipo int y representa el número de argumentos pasados al programa. Al usar int en el primer bucle, estamos iterando directamente sobre este valor, lo cual es natural y eficiente.

¿Por qué size_t en el bucle anidado?

- size_t es un tipo entero sin signo: Está diseñado para representar tamaños y conteos, y es el tipo de retorno de muchas funciones que devuelven tamaños, como strlen.

- Evita desbordamientos: Al usar size_t, evitamos posibles desbordamientos negativos que podrían ocurrir si usáramos un tipo con signo como int para índices que siempre son positivos.
- Mayor precisión: En sistemas de 64 bits, size_t suele ser de 64 bits, lo que permite representar tamaños más grandes que un int de 32 bits.

¿Por qué no usar int en ambos casos?

- Riesgo de desbordamiento: Si el número de argumentos o la longitud de una cadena es muy grande, un int podría desbordarse, lo que llevaría a un comportamiento indefinido.
- Pérdida de información: En sistemas de 64 bits, un int de 32 bits podría no ser capaz de representar todos los posibles tamaños de objetos, lo que podría llevar a una pérdida de información.

¿Por qué usar un ciclo for?

Los ciclos for en C++ son una estructura de control fundamental, que nos permite ejecutar un bloque de código repetidamente, un número determinado de veces o mientras se cumpla una determinada condición.

Estructura básica de un ciclo for:

```
for (inicialización; condición; incremento) {
  // Código a ejecutar en cada iteración
}
```

Cada parte de esta estructura tiene un propósito específico:

- Inicialización: Se ejecuta una sola vez al comienzo del ciclo. Aquí se suele declarar e inicializar una variable de control que se utilizará para controlar el número de iteraciones.
- Condición: Se evalúa al inicio de cada iteración. Si la condición es verdadera, se ejecuta el cuerpo del ciclo. Si es falsa, el ciclo termina.
- Incremento: Se ejecuta al final de cada iteración. Aquí se actualiza el valor de la variable de control, generalmente incrementándola o decrementándola.

Ejemplo:

```
for (int i = 0; i < 5; i++) {
  std::cout << "El valor de i es: " << i << std::endl;
}
```

En este ejemplo:

- Inicialización: int i = 0: Se declara una variable i y se inicializa en 0.
- Condición: i < 5: El ciclo se ejecutará mientras i sea menor que 5.
- Incremento: i++: Después de cada iteración, el valor de i se incrementa en 1.

- Cuerpo del ciclo: Se imprime en pantalla el valor actual de i.

- Volviendo al ejercicio, dentro del bucle anidado, se utiliza la función toupper() de la biblioteca <cctype> para convertir cada carácter a mayúscula.
- static_cast: Esta es una conversión explícita de tipo. Lo que hace aquí es tomar el resultado de la expresión entre paréntesis y convertirlo al tipo de dato char. Esto es necesario porque la función toupper devuelve un valor de tipo int.

  ¿Por qué static_cast<char> en C++ y no (char) como en C?

  La principal razón es que C++ ofrece un conjunto más amplio y seguro de mecanismos de conversión de tipo que C. static_cast es una de esas herramientas y proporciona varias ventajas:
  - Mayor seguridad: El compilador puede realizar más comprobaciones en tiempo de compilación para detectar posibles errores de conversión que podrían pasar desapercibidos con un simple cast C-style ((char)).
  - Claridad: La sintaxis static_cast<tipo> deja explícita la intención del programador de realizar una conversión de tipo, mejorando la legibilidad del código.
  - Especificidad: Existen otros tipos de conversiones en C++ (como const_cast, dynamic_cast, reinterpret_cast) que se utilizan para diferentes propósitos. static_cast es específico para conversiones entre tipos relacionados.

  Ejemplo de lo anterior:

  ```
  int x = 65;
  char c = static_cast<char>(x);        // c ahora contiene 'A'
  ```

  En este ejemplo, static_cast<char> garantiza que la conversión de int a char se realice de forma segura y explícita.

- ¿Por qué es necesario convertir a char? Porque std::cout espera un carácter para imprimirlo en la pantalla. Al hacer esta conversión, estamos asegurándonos de que el valor que se imprime sea un carácter válido.
- Por tanto: se imprimen los caracteres en mayúsculas y un espacio entre cada argumento, finalizando con un salto de línea.

COMPILACIÓN Y EJECUCIÓN

Guarda el código en un archivo llamado megaphone.cpp

Compilación (Makefile):

```
CXX=c++
CXXFLAGS=-Wall -Wextra -Werror -std=c++98

all: megaphone
```

```
megaphone: megaphone.cpp
$(CXX) $(CXXFLAGS) -o megaphone megaphone.cpp
@echo -e "\033[32mCompilation is done!\033[0m"

clean:
rm -rf megaphone
```

Explicación:

- CXX: Define la variable CXX como c++, que es el comando utilizado para compilar el código C++.
- CXXFLAGS: Define la variable CXXFLAGS con las opciones de compilación:
  - -Wall: Activa todas las advertencias.
  - -Wextra: Activa advertencias adicionales.
  - -Werror: Trata las advertencias como errores, lo que ayuda a encontrar problemas potenciales más fácilmente.
  - -std=c++98: Especifica que el código debe ser compatible con el estándar C++98.
- all: Es la regla predeterminada. Ejecuta la regla megaphone para compilar el programa.
  - Dependencia: Depende del archivo megaphone.cpp.
  - Regla: Utiliza el comando $(CXX) (que es c++) con las opciones $(CXXFLAGS) para compilar el archivo megaphone.cpp y crear el ejecutable megaphone.
  - Agregar un toque de color a tus mensajes de compilación puede hacer que tus sesiones de desarrollo sean más agradables visualmente y te ayuden a identificar rápidamente el estado de tus builds.
    - → @echo -e:
      - @: Evita que el comando echo se imprima en la salida estándar.
      - -e: Habilita la interpretación de secuencias de escape.
    - → \033[32m: Inicia una secuencia de escape ANSI (32m: Establece el color de texto a verde).
    - → \033[0m: restablece los atributos de texto a los valores predeterminados.
- clean: Elimina el archivo ejecutable megaphone utilizando el comando rm -rf.

Cómo usar el Makefile:

- Guarda el código anterior en un archivo llamado Makefile en el mismo directorio que contiene megaphone.cpp.
- Ejecuta el comando make en la terminal. Esto compilará el código y creará el ejecutable megaphone.
  - Al ejecutar make, se llega a la regla all.
  - Se ejecuta el comando de compilación.
  - Después de la compilación, se ejecuta el comando echo con la secuencia de escape ANSI para imprimir el mensaje en verde.

- Limpiar: Para eliminar el ejecutable, ejecuta el comando make clean.

Ejecutar: Ejecuta el programa con diferentes argumentos:

```
./megaphone "shhhhh... I think the students are asleep..."
./megaphone Damnit " ! " "Sorry students, I thought this thing was off."
./megaphone
```

EXPLICACIÓN DE LAS CABECERAS

#include <iostream>

¿Qué hace?

Esta línea incluye la biblioteca estándar de entrada/salida de C++.

Por qué es necesaria:

- std::cout: Esta línea permite utilizar el objeto std::cout para imprimir información en la consola. Sin esta cabecera, no podrías usar std::cout para mostrar el texto en la pantalla.
- std::endl: Inserta un salto de línea en la salida. También limpia el búfer de salida, lo que garantiza que el texto se muestre inmediatamente en la pantalla.

#include <cctype>

¿Qué hace?

Esta línea incluye la biblioteca de caracteres de C.

Por qué es necesaria:

- toupper(): Esta función, definida en <cctype>, convierte un carácter en minúscula a su equivalente en mayúscula. En el código, la usas para convertir cada carácter de la entrada a mayúscula.

En resumen:

- Las cabeceras en C++ son archivos que contienen declaraciones de funciones, clases, macros y otras entidades predefinidas.
- Al incluir una cabecera en tu código, estás haciendo que el compilador tenga acceso a estas definiciones.
- En este caso específico, necesitas <iostream> para realizar operaciones de entrada/salida y <cctype> para utilizar la función toupper().