# Índice de Contenido

# 1. Programa (C++)

El programa está realizado en C++. En él se simula un bloque de 8 LEDs situados consecutivamente. Para manejar este bloque de LEDs se han implementado ciertas operaciones básicas con dichos LEDs, tales como apagar/encender los LEDs.

Cada LED puede estar en dos estados, encendido (1) o apagado (0). Cada LED se representa con 1 bit, por tanto, los 8 LEDS, se representan con 1 byte. Haciendo uso de variables del tipo 'unsigned char' se puede representar el bloque de LEDs. Cada variable dispone de exactamente 1 byte.

## 1.1. Funciones

### 1.1.1. Función on()

Se le pasa un objeto bloqueLed por referencia y un entero con la posición. La función no devuelve nada (void).

La función enciende el LED en la posición designada.

*Funcionamiento:*

- Creamos una máscara con un 1 en la posición que queremos encender.
- Utilizamos el operador OR (|) para encender el LED en la posición y lo guardamos.

```
void on(bloqueLed &b, int pos){

    unsigned char mask = 1 << pos;
    b = mask | b;
}
```

### 1.1.2. Función off()

Se le pasa un objeto bloqueLed por referencia y un entero con la posición. La función no devuelve nada (void).

La función apaga el LED en la posición designada.

*Funcionamiento:*

- Creamos una máscara con un 1 en la posición que queremos apagar.
- Usamos el operador NOT (~) para cambiar 0 por 1 y viceversa.
- Finalmente usamos el operador AND (&) para apagar el LED.

```
void off(bloqueLed &b, int pos){

        unsigned char mask = 1 << pos;
        b = ~mask & b;
}
```

### 1.1.3. Función get()

Se le pasa un objeto bloqueLed y un entero con la posición. La función devuelve un objeto bool con el estado del LED (encendido = true, apagado = false) en la posición designada.

*Funcionamiento:*

- Se crea una máscara con un 1 (Encendido) en la posición dada.
- Si el LED esta encendido devuelve true, si no, false.
- Se usa la operación AND (&).

```
bool get(bloqueLed b, int pos){
        unsigned char mask = 1 << pos;
        return (b & mask);
}
```

### 1.1.4. Función print()

Se le pasa un objeto bloqueLed. La función no devuelve nada (void).

Muestra por la salida estándar una secuencia de 0s y 1s correspondiente  al estado de cada LED.

*Funcionamiento:*

- Se hace un bucle for que compruebe las posiciones de 0-7 (8 en total).
- En el printf imprimimos un valor entero en base 10 con signo (int).
- (1<<i) Como 1 == 0000 0001, lo que hacemos es desplazar el 1 a la izquierda i posiciones.
- b & (1<<i) Comparamos esa posición en b con el 1 anterior, lo que nos devuelve el valor de b en esa posición.
- (b & (1<<i))>>i) El valor de b lo mandamos a la posición 0, usando i, para poder imprimirlo en pantalla.

```
void print(bloqueLed b){

        for(int i = 7; i >= 0; i--){
                printf("%d",((b & (1<<i))>>i));
        }
        cout << endl;

}
```

### 1.1.5. Función encender()

Se le pasa un objeto bloqueLed por referencia. La función no devuelve nada (void).

Enciende todos los LEDs (los pone a 1).

*Funcionamiento:*

- Como debemos poner todas las posiciones a 1, igualamos b a 255.

```
void encender(bloqueLed &b){
        b = 255;
}
```

### 1.1.6. Función apagar()

Se le pasa un objeto bloqueLed por referencia. La función no devuelve nada (void).

Apaga todos los LEDs (los pone a 0).

*Funcionamiento:*

- Apagar supone poner todas las posiciones a 0, igualamos b a 0.

```
void apagar(bloqueLed &b){
        b = 0;
}
```

### 1.1.7. Función asignar()

Se le pasa un objeto bloqueLed por referencia y un array de bool constante. La función no devuelve nada (void).

Enciende o apaga el LED de la posición del array según si es true o false.

*Funcionamiento:*

- Hacemos un bucle for que recorre las 8 posiciones.
- Si v[i] es true, en esa posición llamamos a on().
- Si v[i] es false, en esa posición llamamos a off().

```
void asignar(bloqueLed &b, const bool v[]){

        for(int i = 0; i < 8; i++){
                if(v[i]){
                        on(b, i);
                }else{
                        off(b, i);
                }
        }
}
```

### 1.1.8. Función volcar()

Se le pasa un objeto bloqueLed  y un array de bool (al ser array es por referencia). La función no devuelve nada (void).

Asigna a cada posición del array de bool el estado del LED correspondiente.

*Funcionamiento:*

- Hacemos un bucle for que recorre las 8 posiciones.
- Usamos get() que devuelve true si el LED esta encendido y false si está apagado en la posición dada.
- Guardamos el resultado en v[]. El índice del vector corresponderá con la posición del LED.

```
void volcar(bloqueLed b, bool v[]){

        for(int i = 0; i < 8; i++){
                v[i] = get(b, i);
        }
}
```

### 1.1.9. Función encendidos()

Se le pasa un objeto bloqueLed, un array de int (al ser array es por referencia) y un entero por referencia. La función no devuelve nada (void).

Guarda en el vector las posiciones del LED correspondientes donde están encendidos y en el entero pasado por referencia guarda cuantos están encendidos.

*Funcionamiento:*

- Ponemos el entero contador a 0 para asegurarnos.
- Hacemos un bucle for que recorre las 8 posiciones.
- Usamos get() para saber si en esa posición el LED esta encendido.
- Si lo está, guardamos su posición en posic[].
- Incrementamos el entero contador en 1.
- Si el LED está apagado, no hacemos nada.

```
void encendidos(bloqueLed b, int posic[], int &cuantos){

        cuantos = 0;
        for(int i = 0; i < 8; i++){
                if(get(b, i)){
                        posic[cuantos] = i;
                        cuantos++;
                }
        }
}
```

### 1.1.10. Función bloqueLedToString()

Se le pasa un objeto bloqueLed. La función devuelve un string.

Concatena el estado de todos los LED, usando los 0s y 1s.

*Funcionamiento:*

- Se hace un bucle desde la posición 7 hasta la 0, ambas incluidas.
- Usando la función get() obtenemos si el estado del LED es encendido (true) o apagado (false).
- Se concatena en forma de 1s (true) y 0s (false) y se devuelve el string.

```
string bloqueLedToString (bloqueLed b){
        string cadena = "";
        for(int i = 7; i >= 0; i--){
                if(get(b, i)){
                        cadena.push_back('1');
                }else{
                        cadena.push_back('0');
                }
        }
        return cadena;
}
```

## 1.2. Animaciones

Se nos pide que mostremos los LEDs haciéndolos aparecer por pantalla siguiendo un determinado patrón, llamado animación. Para ello nos valemos de los bucles for y varias funciones de control de LEDs para encender y apagar en cada iteración del bucle según el patrón dado.

```
cout << "\nAhora la animacion\nEjemplo 1 \n";
// aqui debes escribir las instrucciones para que se muestre
// el primer ejemplo de animacion.

encender(b);
cout << bloqueLedToString(b) << endl;
for(int i = 7; i >= 0; i--){
            off(b, i);
            cout << bloqueLedToString(b) << endl;
            encender(b);
     }

cout << "\n\nEjemplo 2 \n";
// aqui debes escribir las instrucciones para que se muestre
// el segundo ejemplo de animacion.

encender(b);
cout << bloqueLedToString(b) << endl;
for(int i = 7; i >= 4; i--){
            off(b, i);
            off(b, 7-i);
            cout << bloqueLedToString(b) << endl;
     }
     for(int i = 4; i <= 7; i++){
            on(b, i);
            on(b, 7-i);
            cout << bloqueLedToString(b) << endl;
     }
```

### 1.3. Problemas Surgidos

El único problema surgido durante el desarrollo de la práctica está relacionado con la función bloqueLedToString(). Al utilizar objetos de tipo string, había que incluir dos líneas extra en el código:

- using namespace std;
- #include <cstring>

Si no se incluyen estas líneas en todos los sitios donde usemos el objeto string obtenemos un error de compilación.

## 2. Makefile

Los makefiles son los ficheros de texto que utiliza make para llevar la gestión de la compilación de programas. Se nos pide crear un fichero makefile con la intención de automatizar el proceso de compilación de nuestro programa.

El uso del fichero makefile se realiza usando el comando make. La reglas están preparadas para funcionar tanto con un make all como con un make de una regla en concreto.

En primer lugar hemos creado las variables que contienen los nombres de los directorios necesarios.

```
# Directorio include (ficheros .h)
INC = include

# Directorio bin (ejecutables)
BIN = bin

# Directorio obj (ficheros .o)
OBJ = obj

# Directorio src (ficheros .cpp)
SRC = src
```

Una vez hecho esto, hemos creado las reglas necesarias para la compilación del programa.

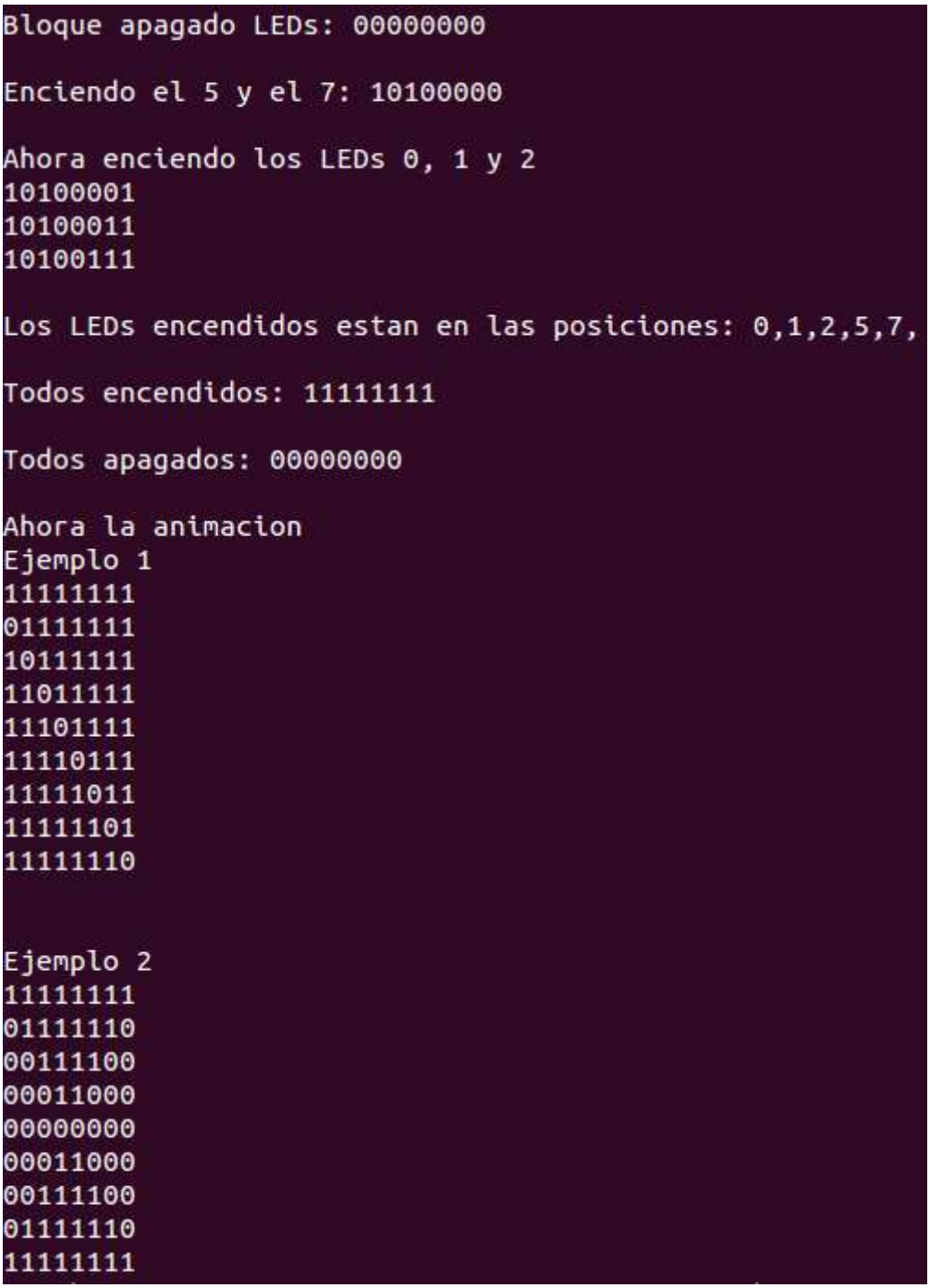
```
all: $(BIN)/main

bin/main: reconstruir $(OBJ)/main.o $(OBJ)/bloqueLed.o
    g++ -o $(BIN)/main $(OBJ)/main.o $(OBJ)/bloqueLed.o

obj/bloqueLed.o: reconstruir $(SRC)/bloqueLed.cpp $(INC)/bloqueLed.h
    g++ -c $(SRC)/bloqueLed.cpp -o $(OBJ)/bloqueLed.o -I$(INC)/

obj/main.o: reconstruir $(SRC)/main.cpp $(INC)/bloqueLed.h
    g++ -c $(SRC)/main.cpp -o $(OBJ)/main.o -I$(INC)/

reconstruir:
    test -d $(OBJ) || mkdir -p $(OBJ)
    test -d $(BIN) || mkdir -p $(BIN)

clean:
    echo "Limpiando..."
    rm $(OBJ)/*.o

mrproper: clean
    rm $(BIN)/main
```

Como se puede observar, se ha incluido una regla extra llamada “reconstruir”. Esta regla comprueba la existencia de los directorios donde se guardaran los objetos (.o) y donde se guardara el programa compilado. Dado que estas carpetas están vacías hasta que el programa es compilado se corre el riesgo de que sean borradas. Incluyendo esta nueva regla evitamos errores de compilación derivados de la no existencia del directorio.

No se ha realizado este procedimiento con el resto de directorios puesto que si fueran borrados, el código que contienen también desaparecería y la compilación sería imposible.

## 3. Ejecución del Programa

En este apartado se pueden ver imágenes del programa siendo ejecutado en el terminal de Ubuntu. También se adjuntan capturas del uso del fichero makefile para atestiguar que todo funciona correctamente y de acuerdo a las especificaciones dadas.




Esta primera captura muestra el funcionamiento del programa. En la siguiente podemos ver el uso del fichero makefile. En la ejecución no se aprecia pero está realizando también la comprobación de directorios antes mencionada.

```
rafabailon@ei142178:~/Escritorio/Practica 02/bloqueLED$ make mrproper
echo "Limpiando..."
Limpiando...
rm obj/*.o
rm bin/main
rafabailon@ei142178:~/Escritorio/Practica 02/bloqueLED$ make all
g++ -c src/main.cpp -o obj/main.o -Iinclude/
g++ -c src/bloqueLed.cpp -o obj/bloqueLed.o -Iinclude/
g++ -o bin/main obj/main.o obj/bloqueLed.o
```

Para completar la imagen anterior, tenemos otra donde se aprecia claramente el trabajo que realiza el fichero makefile comprobando previamente los directorios que son susceptibles de haber sido borrados accidentalmente.

```
test -d obj || mkdir -p obj
test -d bin || mkdir -p bin
g++ -c src/main.cpp -o obj/main.o -Iinclude/
g++ -c src/bloqueLed.cpp -o obj/bloqueLed.o -Iinclude/
g++ -o bin/main obj/main.o obj/bloqueLed.o
```