UNIVERSIDAD TECNOLÓGICA NACIONAL

FACULTAD REGIONAL PARANÁ

Carrera: Ingeniería electrónica

Cátedra: Técnicas Digitales II

Trabajo Práctico N° 4: Producción de Luces y Sonidos

PROFESORES:
Caballero, Raúl Manuel
Maggiolo, Gustavo Daniel
Britos, Rubén Adrián

INTEGRANTES:
Battaglia, Carlo Ignacio
Escobar, Gabriel Hernán

Fecha de entrega: 09/04
Año Lectivo: 2022

Actividad I: Secuencia de luces

1. El circuito deberá contar con 8 LEDs, conectados a IOL en el Arduino y dispuestos de la forma que desee, para la resolución de la actividad. Un pulsador en la entrada 8 de IOH.

El esquema del circuito se incluirá al en el punto **1** de la actividad II, integrando esta actividad y la siguiente.

2. El funcionamiento general del circuito es:

a. Al inicio no debe estar encendido ninguno de los 8 LEDs (Puede usar el LED L de la placa Arduino como status y/o monitor).

Para lograr esto, el programa entra en el siguiente bucle siempre que el estado del sistema sea el de standby:

```
{
  for(int i = 0; i < 8; i++) digitalWrite(pinLED[i], LOW);
  digitalWrite(LED_BUILTIN, !digitalRead(LED_BUILTIN));
}
```

b. Al presionar el pulsador, repetidamente, deberá reproducir una secuencia a su elección determinada, sobre un total de DIEZ (10).

En nuestro caso, el bucle principal del programa ejecutará (siempre que se encuentre en este modo) el siguiente bloque de código:

```
if(!digitalRead(PIN_BUTTON)) t1 = millis();
else elapsed = millis() - t1;
if(elapsed > 10)
{
  if(elapsed > 1000) btnCount = 10;
  else{
    if(btnCount >= CANT_SEC-1) btnCount = 0;
    else btnCount++;
  }
  elapsed = 0;
}
```

Esto nos permite detectar la pulsación del botón y obtener la diferencia de tiempo entre el momento en que fue presionado y el momento en que dejó de serlo. En base a este tiempo, podemos determinar si es requerido incrementar la variable `btnCount` (encargada de llevar cuenta de las veces que fue presionado el botón y elegir el estado correspondiente) o, en el caso de haberlo presionado durante más de un segundo, poner al sistema en standby (como explicamos en el punto anterior) fijando el valor de `btnCount = 10`.

c. La secuencia se repite indefinidamente, hasta presionar nuevamente el pulsador.

Luego de ejecutarse el código anterior, el bucle principal recorrerá la secuencia correspondiente al estado actual de la variable `btnCount`:

```
if(btnCount < 7)
{
  for(int i = 0; i < 8; i++)
    digitalWrite(pinLED[i],secuencia[btnCount][state][i]);
  if(state >= cantEstados[btnCount]-1) state = 0;
  else state++;
}else if(btnCount < 10)
{
  int secMemPos = (btnCount-7)*(MAX_ESTADOS+1);
  int cantEstadosMem = EEPROM.read(secMemPos);
  for(i = 0; i < 8; i++)
    digitalWrite(pinLED[i], bitRead(EEPROM.read(secMemPos+state+1), i));
  if(state >= cantEstadosMem-1) state = 0;
  else state++;
}else
{
  for(int i = 0; i < 8; i++) digitalWrite(pinLED[i], LOW);
  digitalWrite(LED_BUILTIN, !digitalRead(LED_BUILTIN));
}
delay(100);
```

Como podemos ver, los primeros 7 casos corresponden a secuencias que se encuentran almacenadas en una matriz desperdiciando mucha memoria de la siguiente manera:

```
int secuencia[7][MAX_ESTADOS][8] = {
  {
    {0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,1},
    {0,0,0,0,0,0,1,1},
    {0,0,0,0,0,1,1,1},
    {0,0,0,0,1,1,1,1},
    {0,0,0,1,1,1,1,1},
    {0,0,1,1,1,1,1,1},
    {0,1,1,1,1,1,1,1},
    {1,1,1,1,1,1,1,1}
  },
  {
    {1,1,1,1,1,1,1,1},
    {0,1,1,1,1,1,1,1},
    {0,0,1,1,1,1,1,1},
    {0,0,0,1,1,1,1,1},
    {0,0,0,0,1,1,1,1},
    {0,0,0,0,0,1,1,1},
    {0,0,0,0,0,0,1,1},
    {0,0,0,0,0,0,0,1},
    {0,0,0,0,0,0,0,0}
  },

      ...
};
```

d. Tres (3) de estas secuencias deberán estar almacenadas en la memoria EEPROM del microcontrolador. Esto debe estar reflejado en el informe, es decir, como graba las secuencias en la EEPROM y como las lee luego para mostrarlas.

Las secuencias correspondientes a los valores de `btnCount` entre 7 y 9 están almacenados en la memoria EEPROM del Arduino, y son recorridas de la siguiente manera (tal como se vió en le punto anterior):

```
else if(btnCount < 10)
{
  int secMemPos = (btnCount-7)*(MAX_ESTADOS+1);
  int cantEstadosMem = EEPROM.read(secMemPos);
  for(i = 0; i < 8; i++)
    digitalWrite(pinLED[i], bitRead(EEPROM.read(secMemPos+state+1), i));
  if(state >= cantEstadosMem-1) state = 0;
  else state++;
}
```

Al comienzo de cada secuencia se guardó la cantidad de estados que tiene dicha secuencia, por lo que primero leemos este valor y determinamos la cantidad de iteraciones que corresponden. El bucle avanzará en los estados de la secuencia mientras que este valor total de estados no sea excedido.
Cada secuencia es identificada por su posición que surge como múltiplo de la variable `btnCount-7`, es decir, cuando el programa identifica el estado 7, le restamos 7 para obtener 0, lo que significa que estamos en la primera posición de memoria correspondiente a la primer secuencia de luces. Para `btnCount = 8` nos encontraremos en la secuencia 1 de la memoria y así sucesivamente.
Cada estado es leído directamente de la memoria y luego mediante la función `bitRead()` leemos los bits individuales que componen el dato para escribir la información en el led correspondiente.

La memoria fue escrita previamente en un programa aparte que guarda cada secuencia en orden definiendo un tamaño máximo para cada una. En este caso el máximo es de 11 estados:

```
#include <EEPROM.h>
#define MAX_ESTADOS 11

void setup()
{
    int sec1 = 0, sec2 = MAX_ESTADOS+1, sec3 = (MAX_ESTADOS+1)*2;
    EEPROM.update(sec1,2);       // comienzo secuencia 1
    EEPROM.update(sec1+1,0);
    EEPROM.update(sec1+2,255);
    EEPROM.update(sec2,2);       // comienzo secuencia 2
    EEPROM.update(sec2+1,170);
    EEPROM.update(sec2+2,85);
    EEPROM.update(sec3,9);       // comienzo secuencia 3
    EEPROM.update(sec3+1,0);
    EEPROM.update(sec3+2,1);
    EEPROM.update(sec3+3,2);
    EEPROM.update(sec3+4,4);
    EEPROM.update(sec3+5,8);
    EEPROM.update(sec3+6,16);
    EEPROM.update(sec3+7,32);
    EEPROM.update(sec3+8,64);
    EEPROM.update(sec3+9,128);

    pinMode(LED_BUILTIN, OUTPUT);
}
```

```
void loop()
{
    digitalWrite(LED_BUILTIN, HIGH);
    delay(200);
    digitalWrite(LED_BUILTIN, LOW);
    delay(200);
}
```

Utilizamos la función `EEPROM.update()` porque permite actualizar la memoria evitando escribirla si no es necesario, ahorrando ciclos de escritura de la EEPROM.
Al terminar de grabado el led integrado en el Arduino parpadea comunicando su finalización exitosa.

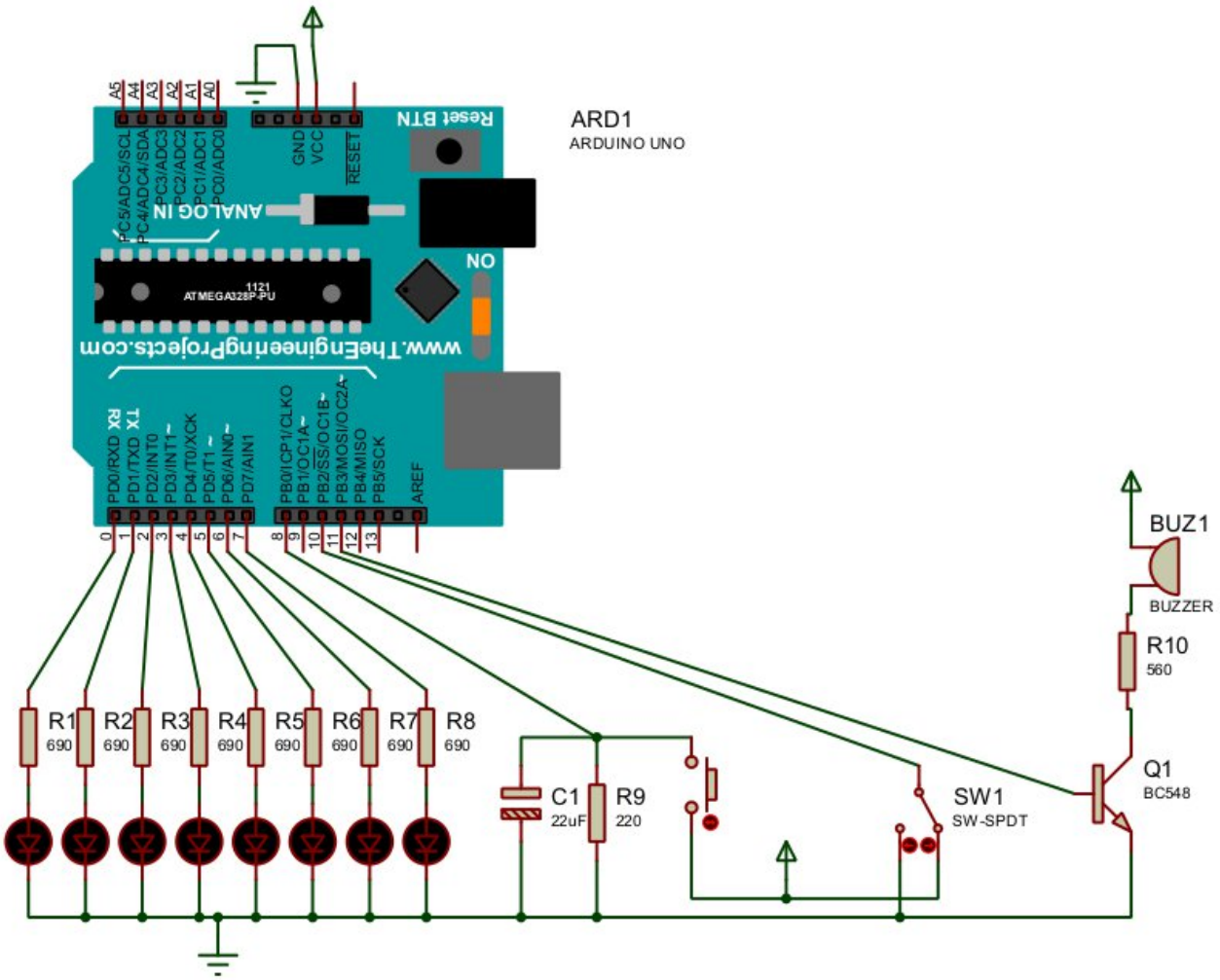
e. Si el pulsador se presiona por un segundo o más deberá volver a la condición inicial (punto a).

Esto fue cubierto previamente en el punto **2.b**.

Actividad II: Generación de melodías

1. El circuito deberá contar con un Buzzer y un pulsador, conectados al esquema Arduino, en pines adecuados para tal fin.

El circuito (integrando la actividad anterior) resulta de la siguiente manera:



2. El circuito del Buzzer deberá tener su etapa de driver, preferentemente utilizando la alimentación de entrada Vin.

En el esquema anterior se puede observar la etapa de driver implementada con un transistor NPN **BC548** y una resistencia de $560\,[\Omega]$ necesaria para atenuar el sonido y evitar una distorsión molesta.

3. El circuito del pulsador deberá tener una etapa de anti-rebote, por hardware y por software.

El pulsador conectado a el pin 8 del Arduino cuenta con un circuito anti-rebote por hardware compuesto por un capacitor de $220\,[uF]$ y una resistencia en paralelo de $220\,[\Omega]$.

El anti-rebote por software se logró implementando la lectura de manera que un cambio de estado en el pulsador signifique una pulsación que no se repetirá hasta ser soltado y presionado nuevamente:

```
bool antiRebote(int in)
{
  bool ret = false;
  if(digitalRead(in) == 0)
    pulsado = 1;
  if(pulsado && digitalRead(in))
  {
    ret = true;
    pulsado = 0;
  }
  return ret;
}
```

Esta función cambia el valor de una variable global encargada de bloquear la lectura del pin luego de ser leído una vez hasta que la lectura cambie.

4. El funcionamiento general del circuito es:

a. Al inicio no deberá generarse ninguna melodía (Puede usar el LED L de la placa Arduino como status y/o monitor).

De la misma manera que en el modo de la Actividad I, el circuito entra en standby al iniciar el programa:

```
default:
  digitalWrite(LED_BUILTIN, !digitalRead(LED_BUILTIN));
  delay(100);
  break;
```

Que en este oportunidad es el caso default de un switch en función de la variable `btnCount`, tal como veremos en el punto **4.b** en la explicación de la selección de los casos.

b. Al presionar el pulsador repetidamente deberá ir reproduciendo (indefinidamente) cada una de las melodías que se detallan a continuación:

Al comienzo del bucle principal se encuentra el siguiente bloque de código encargado de incrementar la variable `btnCount` cuando corresponda, y detecte en una ventana de tiempo de $500\,[ms]$ una doble pulsación del botón, llevando el sistema al estado de standby y reiniciando la reproducción de las melodías:

```
if(antiRebote(PIN_BUTTON))
{
  t1 = millis();
  while((millis() - t1) < 500)
  {
    if(antiRebote(PIN_BUTTON))
      reset = true;
  }
  if(!reset)
  {
    if(btnCount < 2) btnCount++;
    else btnCount = 0;
  }else{
    reset = false;
    btnCount = 3;
  }
  nota = 0;
}
```

Luego, en función del valor de `btnCount` se elegirá la canción a reproducir:

```
switch (btnCount)
{
case 0:
  cantNotas = sizeof(felizcumple) / sizeof(int) / 2;
  finMelodia = reproducir(felizcumple,nota);
  break;
case 1:
  cantNotas = sizeof(lacucaracha) / sizeof(int) / 2;
  finMelodia = reproducir(lacucaracha,nota);
  break;
case 2:
  cantNotas = sizeof(takeonme) / sizeof(int) / 2;
  finMelodia = reproducir(takeonme,nota);
  break;
default:
  digitalWrite(LED_BUILTIN, !digitalRead(LED_BUILTIN));
  delay(100);
  break;
}
```

A continuación se detalla la estructura de cada canción. Las canciones son vectores que guardan en las notas en el orden correspondiente de reproducción, acompañadas de su duración, que en este caso representa el divisor de un tono musical.

En un archivo header externo denominado ***notas.h*** incluimos la definición de cada una de estas notas y otras magnitudes necesarias, como el tempo de la música y la duración en de una nota entera:

```
#define TEMPO       140
#define WHOLE_NOTE  ((60000 * 4) / TEMPO) //duración de una nota completa
```

De esta manera el número que acompaña la nota determina que fracción de una nota entera durará dicha nota. Para notas que se prolongan un tiempo mayor al de una nota completa, usamos números negativos.

i. El “Feliz cumpleaños”.

```
int felizcumple[] = {

  NOTA_C4,4, NOTA_C4,8,
  NOTA_D4,-4, NOTA_C4,-4, NOTA_F4,-4,
  NOTA_E4,-2, NOTA_C4,4, NOTA_C4,8,
  NOTA_D4,-4, NOTA_C4,-4, NOTA_G4,-4,
  NOTA_F4,-2, NOTA_C4,4, NOTA_C4,8,

  NOTA_C5,-4, NOTA_A4,-4, NOTA_F4,-4,
  NOTA_E4,-4, NOTA_D4,-4, NOTA_AS4,4, NOTA_AS4,8,
  NOTA_A4,-4, NOTA_F4,-4, NOTA_G4,-4,
  NOTA_F4,-2,

};
```

ii. La canción “La cucaracha”

```
int lacucaracha[] = {

  NOTA_G4,8, NOTA_G4,8, NOTA_G4,8, NOTA_C5,4, NOTA_E5,8, REST,4,
  NOTA_G4,8, NOTA_G4,8, NOTA_G4,8, NOTA_C5,4, NOTA_E5,8, REST,4,
  NOTA_C4,4, NOTA_C4,8, NOTA_B4,8, NOTA_B4,8, NOTA_A4,8, NOTA_A4,8, NOTA_G4,8, REST,8,
  NOTA_G4,8, NOTA_G4,8, NOTA_G4,8, NOTA_B4,4, NOTA_D5,8, REST,4,
  NOTA_G4,8, NOTA_G4,8, NOTA_G4,8, NOTA_B4,4, NOTA_D5,8, REST,4,
  NOTA_G4,4, NOTA_A4,8, NOTA_G4,8, NOTA_F4,8, NOTA_E4,8, NOTA_D4,8, NOTA_C4,8, REST,8,
};
```

iii. Un tema a elección del grupo.

```
int takeonme[] = {

  NOTA_FS5,8, NOTA_FS5,8,NOTA_D5,8, NOTA_B4,8, REST,8, NOTA_B4,8, REST,8, NOTA_E5,8,
  REST,8, NOTA_E5,8, REST,8, NOTA_E5,8, NOTA_GS5,8, NOTA_GS5,8, NOTA_A5,8, NOTA_B5,8,
  NOTA_A5,8, NOTA_A5,8, NOTA_A5,8, NOTA_E5,8, REST,8, NOTA_D5,8, REST,8, NOTA_FS5,8,
  REST,8, NOTA_FS5,8, REST,8, NOTA_FS5,8, NOTA_E5,8, NOTA_E5,8, NOTA_FS5,8, NOTA_E5,8,
  NOTA_FS5,8, NOTA_FS5,8,NOTA_D5,8, NOTA_B4,8, REST,8, NOTA_B4,8, REST,8, NOTA_E5,8,

  REST,8, NOTA_E5,8, REST,8, NOTA_E5,8, NOTA_GS5,8, NOTA_GS5,8, NOTA_A5,8, NOTA_B5,8,
  NOTA_A5,8, NOTA_A5,8, NOTA_A5,8, NOTA_E5,8, REST,8, NOTA_D5,8, REST,8, NOTA_FS5,8,
  REST,8, NOTA_FS5,8, REST,8, NOTA_FS5,8, NOTA_E5,8, NOTA_E5,8, NOTA_FS5,8, NOTA_E5,8,
  NOTA_FS5,8, NOTA_FS5,8,NOTA_D5,8, NOTA_B4,8, REST,8, NOTA_B4,8, REST,8, NOTA_E5,8,
  REST,8, NOTA_E5,8, REST,8, NOTA_E5,8, NOTA_GS5,8, NOTA_GS5,8, NOTA_A5,8, NOTA_B5,8,

  NOTA_A5,8, NOTA_A5,8, NOTA_A5,8, NOTA_E5,8, REST,8, NOTA_D5,8, REST,8, NOTA_FS5,8,
  REST,8, NOTA_FS5,8, REST,8, NOTA_FS5,8, NOTA_E5,8, NOTA_E5,8, NOTA_FS5,8, NOTA_E5,8,
};
```

Codigo reproductor de melodías:

Esta función reproduce una nota individual determinada de la melodía que se le pasa como argumento. Como cada melodía es un vector compuesto de notas y su duración, este vector es recorrido de forma externa a la función, que solo se encarga de la reproducción de la nota correspondiente.

```
bool reproducir(int melodia[], int nota){
  int divider = 0, noteDuration = 0;
  // el vector de la canción tiene un largo de notas*2, porque guarda la
duración correspondiente a cada nota
  if(nota < cantNotas * 2) {
    // calcula la duración de la nota
    divider = melodia[nota + 1];
    if (divider > 0) {
      noteDuration = (WHOLE_NOTE) / divider;
    } else if (divider < 0) {
    // las notas prolongadas más de un tiempo se representan con números
negativos
      noteDuration = (WHOLE_NOTE) / abs(divider);
      noteDuration *= 1.5; // incrementa la duración por la mitad
    }
    // suena la nota el 90% del tiempo estipulado y el resto es una pequeña
pausa entre notas
    tone(PIN_BUZZER, melodia[nota], noteDuration * 0.9);
    // espera el tiempo que dura la nota
    delay(noteDuration);
    // detiene la nota antes de la siguiente
    noTone(PIN_BUZZER);
    return false;
  }else return true;
}
```

Luego, en el bucle principal, cada iteración se moverá una posición en el vector de notas que compone la melodía elegida mediante el pulsador. Cuando la canción termine, esto se verá reflejado en el booleano retornado por la función *reproducir()*, y la nota a reproducir en la siguiente iteración será aquella en la posición 0 del vector, tal como se ve en el siguiente bloque de código:

```
if(!finMelodia) nota = nota+2;
else nota = 0;
```

c. Si el pulsador se presiona dos veces seguidas (similar a un doble clic del mouse) se debe volver al inicio (punto a).

Esta función fue debidamente explicada en el punto **4.b** donde se expresa la forma de manejar las pulsaciones del botón selector de melodías.

Anexo

Selector de modos

Tal como se sugiere en la nota al final del enunciado, disponemos de una entrada extra que usamos como selector entre ambas actividades. En esta entrada conectamos un switch que alterna entre V_{in} y GND, de manera que al leerla podemos usar su estado para seleccionar el modo reproductor de patrones luminosos y el modo reproductor de melodías:

```
if(digitalRead(PIN_SLIDER))
{
      *Modo luces*
}else
{
      *Modo melodías*
}
```

Header “*notas.h*”

```
#ifndef NOTAS_H
#define NOTAS_H

#define NOTA_B0  31
#define NOTA_C1  33
#define NOTA_CS1 35
#define NOTA_D1  37
#define NOTA_DS1 39
#define NOTA_E1  41
#define NOTA_F1  44
#define NOTA_FS1 46
#define NOTA_G1  49
#define NOTA_GS1 52
#define NOTA_A1  55
#define NOTA_AS1 58
#define NOTA_B1  62
#define NOTA_C2  65
#define NOTA_CS2 69
#define NOTA_D2  73
#define NOTA_DS2 78
#define NOTA_E2  82
#define NOTA_F2  87
#define NOTA_FS2 93
#define NOTA_G2  98
#define NOTA_GS2 104
#define NOTA_A2  110
#define NOTA_AS2 117
#define NOTA_B2  123
```

```
#define NOTA_C3  131
#define NOTA_CS3 139
#define NOTA_D3  147
#define NOTA_DS3 156
#define NOTA_E3  165
#define NOTA_F3  175
#define NOTA_FS3 185
#define NOTA_G3  196
#define NOTA_GS3 208
#define NOTA_A3  220
#define NOTA_AS3 233
#define NOTA_B3  247
#define NOTA_C4  262
#define NOTA_CS4 277
#define NOTA_D4  294
#define NOTA_DS4 311
#define NOTA_E4  330
#define NOTA_F4  349
#define NOTA_FS4 370
#define NOTA_G4  392
#define NOTA_GS4 415
#define NOTA_A4  440
#define NOTA_AS4 466
#define NOTA_B4  494
#define NOTA_C5  523
#define NOTA_CS5 554
#define NOTA_D5  587
#define NOTA_DS5 622
#define NOTA_E5  659
#define NOTA_F5  698
#define NOTA_FS5 740
#define NOTA_G5  784
#define NOTA_GS5 831
#define NOTA_A5  880
#define NOTA_AS5 932
#define NOTA_B5  988
#define NOTA_C6  1047
#define NOTA_CS6 1109
#define NOTA_D6  1175
#define NOTA_DS6 1245
#define NOTA_E6  1319
#define NOTA_F6  1397
#define NOTA_FS6 1480
#define NOTA_G6  1568
#define NOTA_GS6 1661
#define NOTA_A6  1760
#define NOTA_AS6 1865
#define NOTA_B6  1976
#define NOTA_C7  2093
#define NOTA_CS7 2217
#define NOTA_D7  2349
#define NOTA_DS7 2489
#define NOTA_E7  2637
#define NOTA_F7  2794
#define NOTA_FS7 2960
#define NOTA_G7  3136
#define NOTA_GS7 3322
#define NOTA_A7  3520
#define NOTA_AS7 3729
#define NOTA_B7  3951
#define NOTA_C8  4186
#define NOTA_CS8 4435
#define NOTA_D8  4699
#define NOTA_DS8 4978
#define REST      0

#define TEMPO       140
#define WHOLE_NOTE  ((60000 * 4) / TEMPO) //duración de una nota completa

#endif
```