Proyecto AnimationLed

Instituto Tecnológico de Costa Rica
Área de Ingeniería en Computadores
CE3104 – Lenguaje, Compiladores e Intérpretes
I Semestre 2021

Objetivo general

- Implementar una serie de "animaciones" usando leds en un arreglo mínimo de 8x8, las cuales deben ser implementadas por medio de un lenguaje de programación creado por requerimientos proporcionados en el presente documento contando además con su respectivo "compilador", el cual permitirá transformar las instrucciones del lenguaje a imágenes dentro del arreglo.

Objetivos específicos

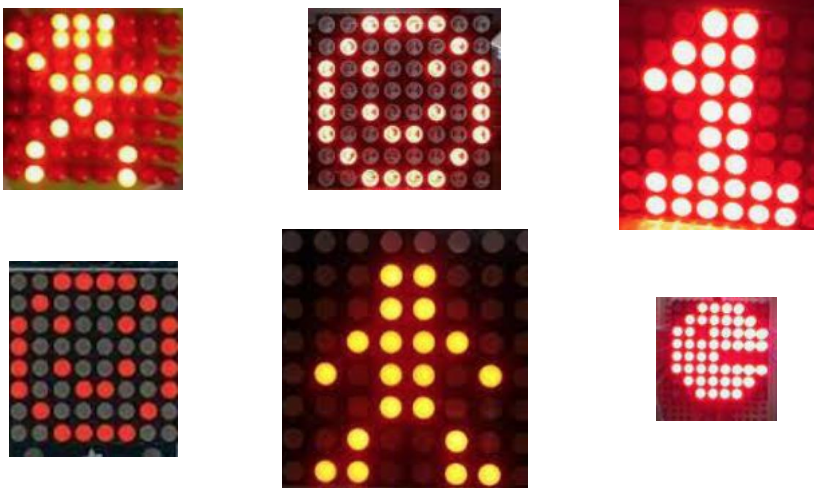
- ➔ Implementación del hardware necesario para la generación de las animaciones.
- ➔ Implementación de un lenguaje de programación completo.
- ➔ Implementación de un compilador del lenguaje anterior.

Datos Generales

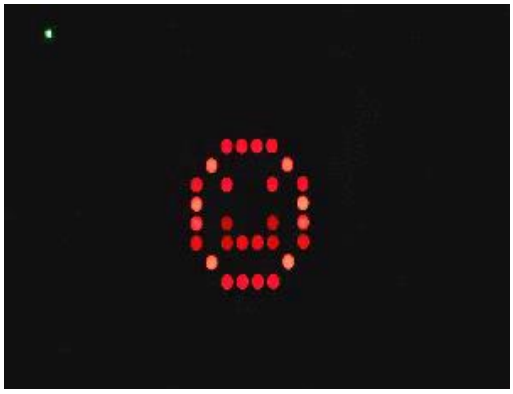
- ➔ El valor del Proyecto: 25%
- ➔ Nombre código: AnimationLed
- ➔ El proyecto debe ser implementado por grupos de máximo de 4 personas.
- ➔ La fecha de entrega del proyecto es de 11/junio/2021
- ➔ Cualquier indicio de copia de proyectos será calificado con una nota de 0 y será procesado de acuerdo con el reglamento.

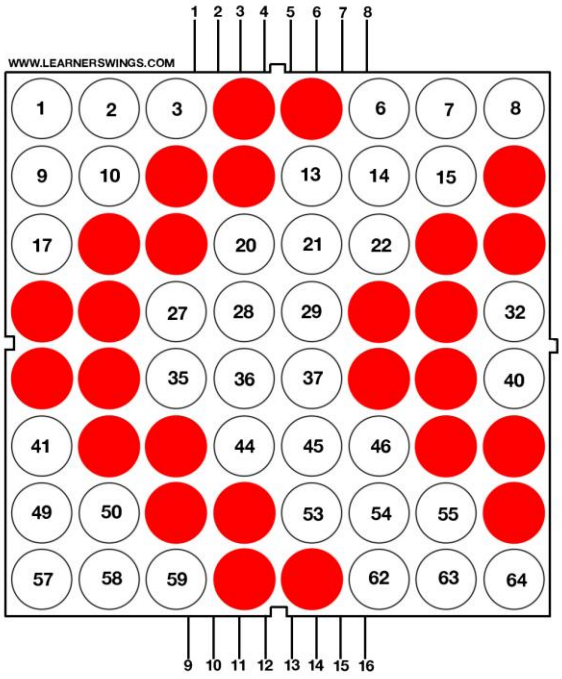
Requerimiento de Resultados

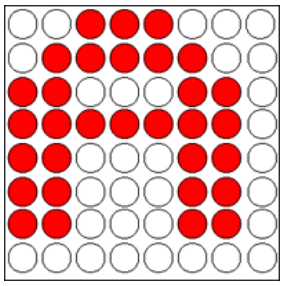
- Generación de una imagen en un dispositivo de leds.

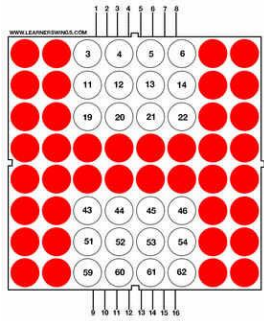


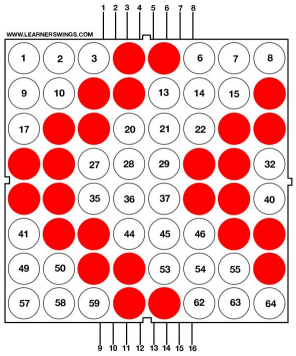
- Generación de animación

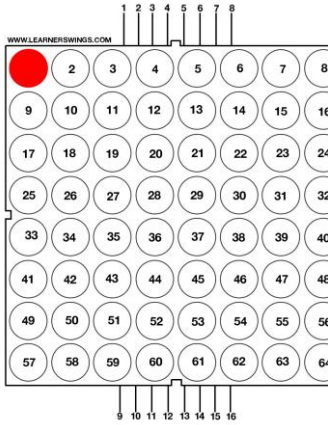


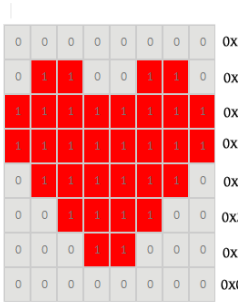

WWW.LEARNERSWINGS.COM

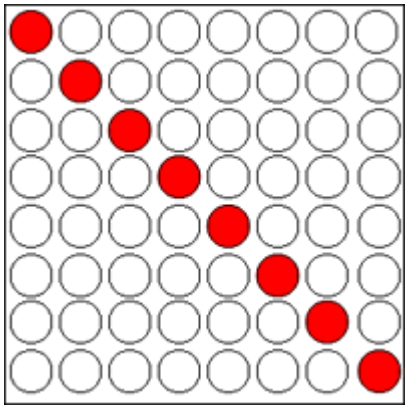


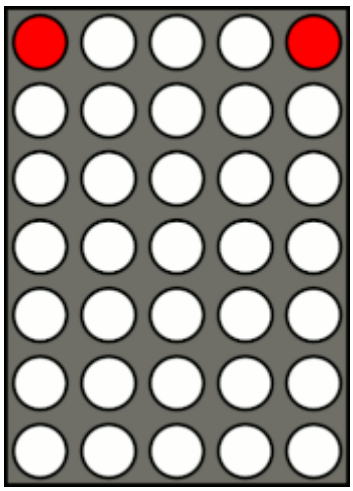


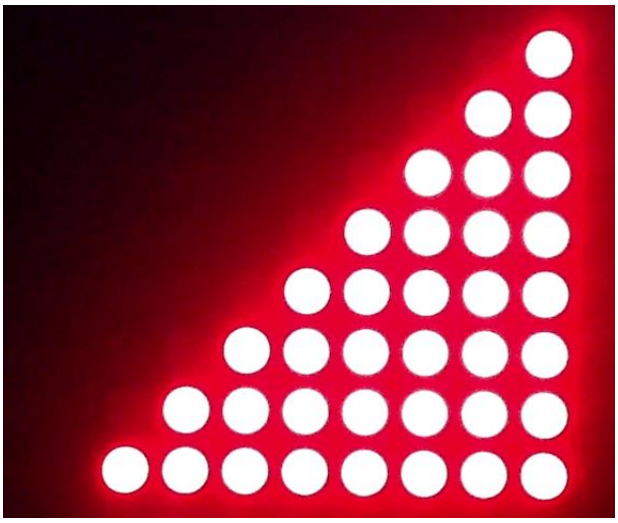












Es importante aclarar que por tratarse de un proyecto de Compiladores se requiere que el "hardware" sea el producto final que venga a presentar los resultados de una correcta compilación de una serie de sentencias escritas bajo reglas de sintaxis previamente establecidas y acordadas.

La programación de las imágenes es mucho más complicada pues se debe ajustar a los requerimientos de hardware (numero de leds) y a la complejidad de la imagen que se desea demostrar. Con la idea de provocar facilidades para ser capaces de hacer más "obras de arte" iluminadas con los leds, es que se requiere la implementación de un lenguaje de programación basada en ciertos requerimientos, y el desarrollo de un compilador que pueda ser capaz de llevar las instrucciones de alto nivel a un nivel del dispositivo de leds y de esa manera ser testigos de nuevas obras de creatividad iluminada.

Sintaxis básica:

Se debe respetar la sintaxis y reglas del lenguaje colocado en el presente enunciado, todo lo que explícitamente no se indique se deja a criterio del grupo, en otras palabras, el grupo tiene la libertad de incluir nuevas funcionalidades y reglas de sintaxis, siempre y cuando se respete lo indicado (no pueden modificar lo indicado en el documento).

Comentarios

Para los comentarios se utiliza el símbolo ## y se comentará desde los guiones hasta el final de línea, por ejemplo

X = 5 + 6 ## Esta es una suma

```
## Esta línea se encuentra comentada.
```

Tipos de Datos

Se manejan dos tipos de datos (booleanos y números).

Las variables se crean usando el operador de asignación = de la siguiente forma:

< Nombre de la variable > = < Valor de la variable >

Ejemplo:

```
Variable1 = 5;

x, y = 20, 15;  donde x = 15 y  Y=20

mi_variable = True;

mi_variable2 = False;
```

Se puede consultar el tipo de datos que posee una variable de la siguiente manera:

```
Variable1 = 5
type(mi_variable)
int

mi_variable = true
type(mi_variable)
bool
```

Variables y asignación de variables

- Las variables usadas en el programa fuente deben tener un máximo de 10 posiciones. Debe iniciar siempre con una letra minúscula, las demás letras pueden ser minúsculas o mayúsculas, y solamente deberá permitir letras, números, y los símbolos especiales "@", "_", y "?".

- Existen variables Globales y variables locales. El "scope" de las variables globales es sobre todo el programa incluyendo dentro de los procedimientos. El "scope" de las variables locales es solamente a nivel de los procedimientos, fuera de estos, dichas variables son descartadas. En caso en que dentro de un

procedimiento existan una variable local con nombre igual a una variable Global, la precedencia será sobre la local, en otras palabras, la variable local tendrá prioridad.

- Las variables pueden ser definidas en cualquier lugar del código fuente.

- La variable obtendrá el tipo de datos en su primera inicialización. Esto es:

    - X = 5; -- implícitamente se le esta indicando que X es numérico

    - Y = true; -- implícitamente se le esta indicando que Y es un booleano

- Una variable no podrá cambiar su tipo de datos posterior a su inicialización. En caso en que se intente, se deberá generar un error semántico. Esto es, si la variable fue inicializada con un numero, y posteriormente se le asigna un valor booleano (o viceversa) esto deberá retroalimentar al usuario con un error.

- Se pueden definir más de una variable de forma conjunta.

Ej.

x, y = 20, 15;

En este caso, se crean dos variables numéricas (x, y) y se inicializan x = 20, y =15.

Operaciones Aritméticas

Las operaciones aritméticas son bastante sencillas y se basan en los siguientes operadores:

Operador	Descripción	Ejemplo
+	**Suma**	**x = 5 + 6 (x es 11)**
-	**Resta**	**x = 5 - 6 (x es -1)**
-	**Negación**	**x = 5 y x = -x (x es -5)**
*	**Multiplicación**	**x = 5 * 5 (x es 25)**

**	Exponente	x = 2 ** 3 (x es 8)
/	División	x = 77 / 8 (x es 9.625)
//	División Entera	x = 77 // 8 (x es 9)
%	Módulo	x = 5 % 2 (x es 1)

Se sigue el mismo orden que el de la aritmética convencional:

- Exponentes
- Módulos
- Multiplicaciones y Divisiones
- Sumas y Restas

```
x + y * 10 / 4
```
23.25

```
x + ((y * 10) / 4)
```
23.25

Pero se puede alterar este orden usando paréntesis.

```
((x + y) * 10) / 4
```
33.75

Listas

Las listas son estructuras **indexadas** que nos permiten almacenar datos booleanos y que representan cada uno de los leds en el dispositivo.

Para crear una lista se usa []

```
mi_variable = []
```

```
x = [True, True, False, False, False, False, True, True]
```

Indice:0	Indice:1	Indice:2	Indice:3	Indice:4	Indice:5	Indice:6	Indice:7
Valor: True	Valor: True	Valor: False	Valor: False	Valor: False	Valor: False	Valor: True	Valor: True

```
x[0]
```
True

```
x[1:4]
```
[True, False, False]

Si se accede a un índice que no existe, se debe generar un error.

Podemos utilizar la función range para crear listas:

```
range(< Tamaño de la lista >, < valor booleano >)
```

```
x = list(range(5, True))
```

```
[True, True, True, True, True]
```

Modificando los valores de la lista

Si buscamos modificar un solo valor por índice es tan sencillo como:

```
milista[0] = True
```

Si buscamos modificar varias entradas a la vez se tiene que hacer por una sublista.

```
milista[1:3] = [True, False]
```

Se debe tener cuidado de no intentar acceder a un índice que no existe en la lista. En dicho caso, se debería generar un Error Semántico.

Agregando valores a una lista

Si queremos agregar un valor en una posición específica se tiene la función insert.

```
MiLista = [True, False, True, False]
```

```
MiLista.insert(2,False)

MiLista = [True, False, False, True, False]
```

Borrando valores de una lista

En nuestro lenguaje del es una palabra reservada que nos permite eliminar un objeto.

```
MiLista = [True, False, True, False]
```

```
MiLista.del(2)
MiLista = [True, False, False]
```

Longitud de una lista

Podemos usar la función len para obtener el tamaño de la lista.

```
MiLista = [True, False, False]
len(MiLista)
```

Retorna 3

Operaciones booleanas

Para efectos del proyecto, se debe interpretar el valor True como el LED ENCENDIDO, y el valor False como el LED APAGADO en la posición o índice ya sea de la lista o matriz representado en el dispositivo físico.

Neg **Cambia el valor booleano**	**x**[1]=true ## Es True **x**[1].Neg ## Ahora **x**[1] es False **x**[1].Neg ## Ahora **x**[1] es True Esta operación solamente se puede aplicar sobre valores booleanos. Caso contrario debe dar un Error. Se puede utilizar en listas, matrices, o rangos de las mismas.
Función **T**	milista[1].T ## Significa que actualiza el índice 1 de milista con el valor True milista[1:3].T ## Significa que actualiza a True desde el índice 1 hasta el 3 (sin incluir). Se puede utilizar en listas, matrices, o rangos de las mismas.
Función **F**	milista[1].F ## Significa que actualiza el índice 1 de milista con el valo r False milista[1:3].F ## Significa que actualiza a False desde el índice 1 hasta el 3 (sin incluir). Se puede utilizar en listas, matrices, o rangos de las mismas.

Blink

Cambia el valor booleano durante un lapso de tiempo

Blink (elemento, tiempo, rango, estatus)

Blink(Dato, Cantidad, RangoTiempo, Estado)

Donde Dato puede ser un indice, un arreglo, etc.

El Estado es True cuando se desea que el led o leds empiecen a "parpadear", y es False cuando se desea deshabilitar ese estado.

El RangoTiempo puede tener alguno de los siguientes valores:

- "**Seg**" en caso de ser Segundos la medida de tiempo,
- "**Mil**" en caso de ser Milisegundos la medida de tiempo,
- "**Min**" en caso de ser Minutos la medida de tiempo

Cualquier otro valor debe generar en error.

Cantidad será el tiempo en que permanecerá el o los leds encendidos para luego apagarse y continuar ese mismo proceso de "parpadeo".

Luego de escrita esta sentencia en el código, se podrá continuar escribiendo código en incluso alterando el dispositivo de leds, pero estos leds continuarán "parpadeando" hasta que se deshabilite a todos o algunos de ellos.

Ej.

Blink(x[1],5, "Seg", True)

En la sentencia anterior, el led [1] de x iniciará en su proceso de parpadeo. Se apagará (False) durante 5 segundos, pasado ese tiempo, se encenderá (True) durante 5 segundos, y así sucesivamente, pues el estatus es True, o sea, se mantiene activo.

Blink(x[1],5, "Seg", False)

En la sentencia anterior, se "desactiva" el blink en el led.

Temporalizador

Las operaciones de temporalizador con que cuenta el lenguaje de programación son las siguientes:

Delay( ) **Mantiene un retraso de tiempo.**	**Delay(**cantidad, RangoTiempo) Ejemplo: **Delay(** 5, "Mil" **)** -- Ejecuta un temporalizador de 5 milisegundos. El RangoTiempo puede tener alguno de los siguientes valores: • "**Seg**" en caso de ser Segundos la medida de tiempo, • "**Mil**" en caso de ser Milisegundos la medida de tiempo, • "**Min**" en caso de ser Minutos la medida de tiempo • Cualquier otro valor debe generar en error. Luego de enviar a encender o apagar leds en el dispositivo y se encuentre la función Delay(), se puede generar ese "espacio de tiempo" hacia la siguiente instrucción en el código. Ejemplo: -- Se enciende el Led en el índice 0 de miLista. milista[0] = True Delay(15,"Seg") ## Se genera un delay de 15 segundos. milista[0] = False ## Transcurridos los 15 segundos se apaga el led La idea con esto, es que cuando se enciende un o varios leds, estos permanezcan encendidos o apagados y se continue ejecutando el programa, pero debido a que la ejecución puede ser demasiado veloz, es importante, manejar un "delay" para realmente ver los efectos en el dispositivo.

Sentencia PrintLed

Para efectos del proyecto, se debe interpretar el valor True como el LED ENCENDIDO, y el valor False como el LED APAGADO en la posición o índice ya sea de la lista o matriz representado en el dispositivo físico.

Para efectos de enviar al arreglo de Leds los valores necesarios para encender o apagar cada led de la matriz 8x8 se tienen las siguientes opciones:

PrintLed(Col, Row, Valor)

Esta sentencia se coloca dentro del arreglo de leds en la posición de los valores correspondientes a la Columna (Col) y Fila (Row) indicados en la sentencia, y de acuerdo con el "Valor" se encenderá o se apagará. Si es True se encenderá, si es False se apagará.

Ejemplo:

```
    Printled(1,1, True)
```

Esto significa que inmediatamente se encenderá el Led en la posición 1,1 (columna=1, Fila=1).

PrintLedX (TipoObjeto, Indice, Arreglo)

El parámetro TipoObjeto puede tener uno de los siguientes tres tipos de valores:

- "C": Columna
- "F": Fila
- "M": Matriz

Cualquier otro valor en el parámetro "TipoObjeto" deberá generar un error de sintaxis.

De acuerdo al valor del parámetro "TipoObjeto" así será accionado los leds del dispositivo físico.

Ejemplo:

```
ListaX = [True, True, False, False, False, False, True, False]

PrintLedX("F", 2, ListaX)
```

Esto significa que en la segunda fila del dispositivo de Leds, inmediatamente los Leds de esa fila tomarán los valores del arreglo encendiéndose o apagándose según el valor dado. Debe ser una ejecución de todos los leds de una sola vez, no de uno a uno.

Ejemplo:

```
 ListaX = [True, True, False, False, False, False, True, False]

PrintLedX("C", 5, ListaX)
```

Esto significa que en la quinta columna del dispositivo de Leds, inmediatamente los Leds de esa columna tomarán los valores del arreglo encendiéndose o apagándose según el valor dado. Debe ser una ejecución de todos los leds de una sola vez, no de uno a uno.

Ejemplo:

```
MiMatriz = [[True ,  True , True, True, False, True, False, True ],
            [True ,  True , True, True, True, True, True, True ],
            [False , False, False, True, True, True, True, True]])


PrintLedX("M", 0, MiMatriz)
```

Esto significa que dentro del dispositivo de Matiz8x8 cada led se encenderá o se apagará de acuerdo al valor de la matriz enviada.

Si los arreglos o la matriz no contiene todos los elementos del dispositivo los otros elementos faltantes en el dispositivo se mantendrán apagados. Por ejemplo un arreglo de solo 5 elementos, los otros tres elementos se mantendrán apagados automáticamente.

Luego de ejecutada esta sentencia, el o los Leds se mantendrán encendidos o apagados según lo enviado, y se continuará la ejecución del código del programa. En este punto, es importante hacer uso adecuado de los Delay.

Ciclos for

for < nombre de la variable que cambia > **in** < iterable > **Step** < Salto >

```
{
        ## ... Cualquier código que queramos repetir
}
```

< nombre de la variable que cambia > : Se refiere a una variable dentro del FOR que permitirá usar para acceder a los índices de lista de valores.

< iterable > : La estructura que será usada para recorrer, generalmente será lista de valores. También se puede utilizar un numero, en este caso se ejecutará el cuerpo del FOR la cantidad del numero colocado en este valor.

< Salto >: Es el incremento que se le aplicará a la variable local al final de cada iteración del FOR. El valor por default es 1. Si no aparece esta opción se usa el valor default.

Si se desea cambiar el valor de una lista en los valores pares, se podría hacer de la siguiente manera:

```
x = [True, True, False, False, False, False, True, True, True, True, False]
x[1].Neg
x[3].Neg
x[5].Neg
x[7].Neg
x[9].Neg
```

o se podría hacer de la siguiente manera que realiza exactamente el mismo proceso:

```
for var1 in x Step 2
{
    x[var1].Neg
}
```

Esto significa que el cuerpo del FOR se ejecutará (iterará) hasta llegar al final de la lista, recorriéndola con saltos de 2 índices por iteración. En resumen se ejecutará el cuerpo por la mitad de la cantidad de elementos de x.

Dentro de la sección de "Iterable" se puede hacer sobre un subconjunto de la lista, por ejemplo:

```
for var1 in milista[1:3]
 {

    x[var1].Neg

}
```

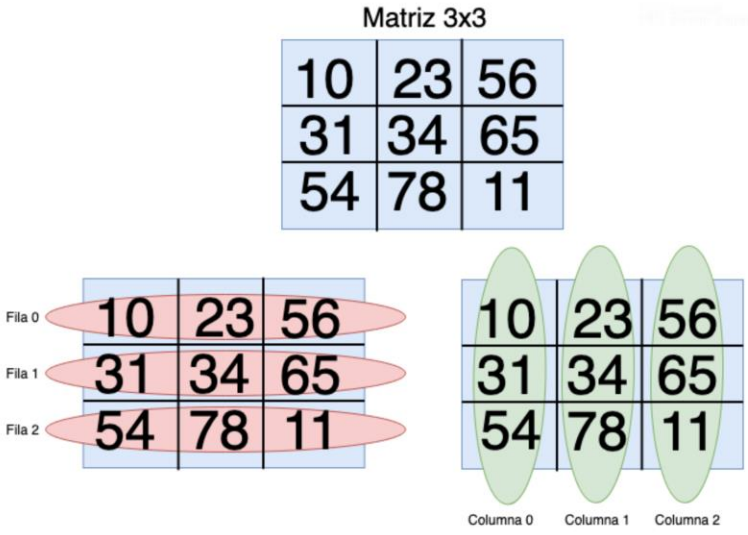
Observe que en el ejemplo anterior, el for utilizará tantas iteraciones como valores o índices se obtiene de la lista iterable colocada, además el FOR no posee Step lo cual hace que el salto sea de 1 en 1.

```
for var1 in 10
 {

    ## Se ejecutará el cuerpo del FOR 10 veces, inicializando en la primera iteración a la variable local "var1" con el valor de 0.

}
```

Matrices

En programación una matriz es una estructura de datos muy útil, y principalmente para nuestro proyecto en cuestión.

Nuestras matrices son listas de dos dimensiones, es decir una lista que dentro posee listas.



Nota: Una matriz de 3x3, es una matriz de 3 filas y de 3 columnas. Cuando mencionamos una matriz nxm estamos hablando de una matriz de n filas y de m columnas.

Para crear una matriz se tiene que hacer como una lista de lista, como sigue:

```
mivariable=[[True,False,True],[True, True, True],[False,False,True]]
```

Ya que una matriz es una lista de listas, podemos acceder a los elementos de esta matriz mediante la **indexación.**

```
mivariable[0] # La primera fila
```

Pero si quisiéramos el elemento de la fila 1 y columna 1, se tiene que hace lo siguiente:

```
mivariable[1][1]
```

Esto pasa porque el primer [] corresponde a las filas y el segundo [] corresponde a las columnas.

Otra forma de poder acceder a un elemento dentro de la lista es usar [,] y se usa de la siguiente manera:

- matriz**[indice_fila]** = nos retorna la fila en el índice **indice_fila**
- matriz**[indice_fila, indice_columna]** = nos retorna el elemento en la fila **indice_fila** y la columna **indice_columna**
- matriz**[:, indice_columna]** = nos retorna la columna en el índice **indice_columna**

```
M[1] # Fila en el índice 1

M[1,1] # Elemento en la fila 1 y columna 1

M[:,1] # Columna en el índice 1
```

Podemos preguntar por las dimensiones de una matriz de la siguiente manera:

```
Matriz1.shapeF    ## Devuelve el numero de Filas que posee la matriz

Matriz1.shapeC    ## Devuelve el numero de Columnas que posee la matriz
```

Estos comandos solamente pueden ser usados dentro de matrices y listas.

El operador **Neg** puede ser utilizado en todo elemento dentro de la matriz, esto es en un elemento en particular, en una columna completa, o en una fila completa, o en un rango completo.

Agregar filas y columnas

Para agregar filas y columnas a una matriz podemos usar la función insert y se usa de la siguiente manera:

Matriz.**insert**(< Nuevo Valor >, < tipo de inserción >,< índice >)

< tipo de inserción >: Si este valor es 0 se agregan los nuevos valores como una fila y si es un 1 se agregan como columna. Cualquier otro valor debe generar un error.

< índice >: Un valor opcional (puede aparecer o no) y significa la posición dentro de la matriz en donde se desea hacer la inserción. Si no aparece, la inserción se ejecutará al final.

Ejemplo:

Se tiene una matriz con los siguientes elementos:

```
[[True ,  True , True ],
 [False , False ,False]])
```

Sin usar el índice:

```
MiNuevaMatriz.insert([[True, False, True]],  0)
```

```
[[True ,  True , True ],
 [False , False ,False]
 [True , False , True ]])  --  Insertada
```

```
MiNuevaMatriz.insert [[True], [True] , [True]], 1)
```

```
[[True ,  True , True ,  True],
 [False , False ,False,  True]
 [True , False , True ,  True]])
```

Usando el índice:

```
MiNuevaMatriz.insert([[False, False, False, False]],  0, 0)
## Al puro inicio
```

```
[[False ,  False , False ,  False],
 [True ,  True , True ,  True],
 [False , False ,False,  True]
 [True , False , True ,  True]])
```

```
MiNuevaMatriz.insert [[True], [True] , [True] , [True]], 1,0)
```

```
[[True,  False ,  False , False ,  False],
 [True,  True ,  True , True ,  True],
 [True, False , False ,False,  True]
 [True, True , False , True ,  True]])
```

Se debe validar que el tamaño de lo insertado corresponda con la dimensión de la matriz, sino debe arrojar un Error. Por ejemplo, intentar insertar una lista de 3 elementos en un espacio donde la dimensión es de 2, esto debe generar un error.

Eliminar filas y columnas

Para eliminar filas y columnas a una matriz podemos usar la función `delete` y se usa de la siguiente manera:

Matriz.**del** (< Índice >, < tipo de eliminación >)

< tipo de eliminación >: Si este valor es 0 se elimina la fila en el índice indicado y si es un 1 se elimina la columna. Cualquier otro valor, debe generar un error.

Ejemplo:

Se tiene una matriz con los siguientes elementos:

```
[[True ,  True , True ],
 [True ,  True , True ],
 [False , False ,False]])
```

```
MiMatriz.delete(0,0)
```

```
[
 [True ,  True , True ],
 [False , False ,False]])
```

```
MiMatriz.delete(1,0)
```

```
[[   True , True ],
 [   False ,False]])
```

Dentro de < Índice > se puede colocar cualquier índice de la matriz, si no existe, se debe generar un error.

No se puede borrar algo que no existe, por tanto se debe controlar que el índice a borrar realmente exista.

Se recomienda para controlar todos los aspectos de los índices, utilizar una tabla de Símbolos de forma eficiente.

Recorriendo una matriz

Podemos usar el **for** para recorrer una matriz

Ocupamos dos **for**, uno para movernos en las filas y otro para movernos en las columnas

Ejemplo:

```
filas = Matriz1.shapeF    ## Numero de Filas que posee la matriz M

columnas = Matriz1.shapeC    ## Numero de Columnas que posee la matriz M
```

```
for indice_fila in filas
{
    for indice_columna in columnas
   {
         Matriz1[indice_fila, indice_columna].Neg
   }
}
```

Si se intenta acceder a un índice que no existe dentro de la matriz, se debe generar el correspondiente error.

Recuerde que dentro del FOR se puede colocar cualquier instrucción que permite el presente lenguaje de programación.

Modificar Matrices

Las matrices pueden recibir actualizaciones de muy diversas maneras, por ejemplo se puede actualizar un elemento especifico:

```
mi_variable[1][1] = TRUE;
mi_variable2[1][1] = mi_variable[1][1];
```

También se puede modificar usando una fila o columna completa de otra lista o matriz:

```
M[1] = M2[5]        ## Fila en el índice 1
```

```
M[:,1] = M2 [:,3]       ## Columna en el índice 1
```

Obviamente en todos estos casos en que la actualización es otra lista o matriz deben ser compatibles a nivel de dimensiones y cantidad de elementos.

Bifurcación

Es la sentencia utilizada para realizar un conjunto de operaciones u otra, o sea, es la sentencia de opción.

If < iterable > **Operador** < valor >
{

```
   ##  Conjunto de instrucciones que forman el cuerpo del IF.   Puede e
scribir cualquier instrucción disponible en el lenguaje de programació
n.
```

}

< iterable > : La estructura que será usada para realizar la validación, puede ser una variable, o bien una lista de valores. En caso de que sea una lista de valores, el cuerpo del IF se ejecutará tantas veces como elementos posea la lista.

Operador: Son los operados de comparación por ejemplo (==, <,<=,>,>=, <>, etc.)
< valor > : Puede ser un número, o un valor booleano, y se usará para realizar la comparación junto con el operador.

Ejemplo:

If MiVariable == **5**
{

```
       ##  Escribir el cuerpo del IF.    Esto se ejecutará una única
vez si se cumple la sentencia en el IF
```

}

```
ListaX = [True, True, False, False, False]
```

```
If ListaX[2] == True
 {
       ##  Escribir el cuerpo del IF.    Esto se ejecutará una única
vez si se cumple la sentencia en el IF
}
```

```
mi_Matriz =[[True, False, True],[True, True, True],[False, False, True] ]

Recuerde que M[:,1] # Columna en el índice 1
```

```
If M[:, 1] == True
 {
       ##  Escribir el cuerpo del IF.

       ##  Se validarán usando la comparación del IF, por cada
elemento de la columna del índice 1.   Por ejemplo, si la primer Fila
de la columna 1 es True, entonces se ejecuta el cuerpo del IF, cuando
termina, se procede con la segunda Fila de la columna 1 y se compara,
y así sucesivamente ejecutando el cuerpo del IF
}
```

Otros aspectos de sintaxis

- Las instrucciones se delimitarán con un punto y coma (;). Si no se encuentra debe generar un error.
- En una misma línea se pueden colocar más de una instrucción delimitadas cada una con un punto y coma.
- Se debe permitir la implementación de procedimientos con la siguiente sintaxis:

```
Procedure NombreRutina (parámetros)
{
   Expresión
};
```

  - Donde NombreRutina es el nombre de la rutina y usa la misma sintaxis de las variables.
  - Expresión puede ser cualquier elemento de programación usando la sintaxis que más adelante se detalla.

- Un procedimiento puede llevar parámetros. En caso de que no necesite parámetros siempre debe llevar los paréntesis. Los parámetros deben ser conceptualizados como variables locales.

- Un procedimiento difiere de otro procedimiento por el nombre del mismo y los parámetros que contenga. Por tanto, cabe mencionar, que la firma de un procedimiento es el nombre y sus parámetros, tomando en cuenta cantidad de parámetros. No puede existir más de un procedimiento con la misma firma.

- Se deben realizar todas las validaciones pertinentes como a nivel léxico, sintáctico, semántico, etc.

- El programa principal es un procedimiento denominado "**Main**()" el cual puede estar colocado en cualquier lugar del programa, pero únicamente puede existir un Main en todo el programa, de lo contrario deberá generar un error el compilador.

La estructura básica del programa es la siguiente:

Procedure **NombreRutina (parámetros)**
Begin
Expresión
end;
	Procedimientos
	Es opcional la colocación de variables en los procedimientos, respetando los lineamientos anteriormente indicados.

Procedure Main ()
begin
Expresión
end;
	Rutina principal
	
	En la rutina principal es factible la definición de variables las cuales serán identificadas como variables globales dentro del programa.

Recuerde que dentro de los procedimientos, se puede tener la capacidad de utilizar, sus propios "parámetros", las variables declaradas dentro del mismo procedimiento (variables locales) y las variables globales definidas en el Main. Deben realizar las validaciones que sean

convenientes para poder interpretar cada una de estas. Esto será parte fundamental de la revisión del proyecto.

Ejemplo:

```
Procedure NombreRutina (parámetros)
 {
       ##  Variables locales y cualquier instrucción del lenguaje.
}

Procedure Main ()
 {
Variable1 = 5      ## Variables globales
x, y = 20, 15
mi_variable = False

call NombreRutina(2);

 ##  Cuerpo del procedimiento.  Variables locales y cualquier instrucción
del lenguaje.
}
…
Procedure NombreRutina (parámetros)
 {
       --  Cuerpo del procedimiento.  Variables locales y cualquier
instrucción del lenguaje.
}
```

Cuando se ejecuta el programa, se debe ejecutar primero las instrucciones del procedimiento principal (“**Main**”) y los demás procedimientos serán llamados usando la instrucción:

```
CALL miProc();   -- cuando no tiene parámetros
CALL miProc(1,2) -- cuando tiene parámetros.
```

IDE

Se debe crear una interfaz al mejor estilo de un “IDE” de programación, el cual posea las siguientes características:

- Debe permitir escribir código de una manera sencilla
- Debe permitir “cargar” programas previamente grabados
- Debe permitir visualizar los errores de compilación que se vaya encontrando.
- Debe existir un botón de “solo compilar” y otro de “Compilar y Ejecutar”

- Los errores deben presentar exactamente la línea de código en donde se encontró, y que sea muy sencillo identificar dicha línea directamente en el código.
- No se permitirá utilizar otras herramientas como el IDE en donde se programó el proyecto.

Para la solución del problema del proyecto presentado se espera del estudiante:

- Fuerte investigación sobre las posibles soluciones en las distintas etapas del problema
- Búsqueda de distintas fuentes que servirán de insumo técnico-práctico para las soluciones así como la ayuda de otras áreas para cumplir con el objetivo del proyecto.
- Selección de criterios acordes al nivel de nuestro ambiente Tec en donde se seleccionen las mejores soluciones correspondientes.
- Una solución solvente de acuerdo con lo esperado.
- Se espera una robusta administración de la tabla de símbolos.
- Si lo desean, pueden utilizar herramientas como Lex y Yacc (y sus distintos "sabores") para facilidades en la implementación del análisis léxico y Sintáctico. Cualquier otra herramienta que "facilite" el desarrollo del proyecto, debe ser consultado previamente con el profesor, para determinar la conveniencia o no de usarlo.
- Para efectos de la revisión del proyecto se debe contemplar lo siguiente:
    - Cada uno de los grupos deberá traer el código suficiente y el dispositivo necesario para presentar una de las animaciones presentadas al inicio del presente documento.
    - Cada uno de los grupos deberá traer el código suficiente y el dispositivo necesario para hacer una animación donde pueda demostrar a completitud todas las bondades del proyecto realizado.
    - El profesor llevará a la revisión un requerimiento tipo "letrero" que deberá generar animación (letras pasando lentamente por el dispositivo). El grupo deberá generar el código suficiente y necesario para realizar dicha animación.
    - Lo anterior será fundamental para efectos de corroborar la integración entre el compilador y el dispositivo de Leds.
    - El profesor podría cambiar la "lógica" dentro de los programas presentados por cada grupo para visualizar algún otro comportamiento.

Documentación y aspectos operativos

- ➔ Se deberá entregar un documento que contenga:
  - ✓ Diagrama de arquitectura de la solución que refleje un nivel de detalle suficiente para entender a términos generales el funcionamiento del proyecto.
  - ✓ Alternativas de solución consideradas y justificación de la seleccionada.
  - ✓ Problemas conocidos: En esta sección se detalla cualquier problema que no se ha podido solucionar en el trabajo.
  - ✓ Actividades realizadas por estudiante: Este es un resumen de las bitácoras de cada estudiante (estilo timesheet) en términos del tiempo invertido para una actividad específica que impactó directamente el desarrollo del trabajo, de manera breve (no más de una línea) se describe lo que se realizó, la cantidad de horas invertidas y la fecha en la que se realizó.
  - ✓ Problemas encontrados: descripción detallada, intentos de solución sin éxito, soluciones encontradas con su descripción detallada, recomendaciones, conclusiones y bibliografía consultada para este problema específico.
  - ✓ Conclusiones y Recomendaciones del proyecto. Conclusiones y recomendaciones con sentido y que confirmen una experiencia profunda en el proyecto.
  - ✓ Bibliografía consultada en todo el proyecto

Atributos

Se debe entregar un **documento aparte** que contenga una portada y el siguiente detalle.

Debe comentar de qué manera se aplicaron los atributos que posee el curso y su impacto, esto por medio de una tabla en donde se detalle para cada atributo lo siguiente:

- Aplicación del atributo en la solución del proyecto
- Impacto del proyecto en la sociedad
- Retroalimentación obtenida gracias al proyecto

Favor colocar la información antes solicitada para cada uno de los atributos siguientes:

- **Conocimiento base de ingeniería**
- **Impacto de la ingeniería en la sociedad y el medio ambiente**
- **Aprendizaje continuo**

Conocimiento de ingeniería
Capacidad para aplicar los conocimientos a nivel universitario de matemáticas, ciencias naturales, fundamentos de ingeniería y conocimientos especializados de ingeniería para la solución de problemas complejos de ingeniería.

Medio ambiente y Sostenibilidad
Capacidad para comprender y evaluar la sostenibilidad y el impacto del trabajo profesional de ingeniería, en la solución de problemas complejos de ingeniería en los contextos sociales y ambientales.

Aprendizaje Continuo
Capacidad para reconocer las necesidades propias de aprendizaje y la habilidad de vincularse en un proceso de aprendizaje independiente durante toda la vida, en un contexto de amplio cambio tecnológico.

Evaluación

1. La implementación del dispositivo de Leds corresponde a un 45% de la nota final del proyecto.
2. El Lenguaje de programación (con IDE incorporado) y todas las etapas del Compilador corresponde a un 45% de la nota final del proyecto. Esto implica también la comunicación entre el compilador y el programa enviado al dispositivo de Leds.
3. La documentación tendrá un valor de un 10% de la nota final del proyecto, cumplir con los puntos especificados en la documentación no significa que se tienen todos los puntos.
4. Cada grupo recibirá una nota en cada uno de los siguientes apartados:
    a. Funcionalidad
    b. Compilador
    c. Documentación
5. De las notas mencionadas en el punto anterior se calculará la Nota Final del Proyecto.
6. No debe imprimirse la documentación, y esta debe encontrarse en formato PDF. Y debe "cargarse" en el TecDigital
7. Documentación digital:
    a. Incluya dentro de su documentación todo el código de programación generado
8. Las citas de revisión oficiales serán determinadas por el profesor durante las lecciones o mediante algún medio electrónico.
9. El profesor llevara un listado de todos y cada uno de los requerimientos mencionados en el presente enunciado del proyecto, de tal manera que se pueda evaluar cada uno de ellos y dar el puntaje determinado.
10. El grupo debe contar con un grupo de procedimientos definidos previamente que demuestre los avances en la ejecución del proyecto. Estos procedimientos deben cumplir con las reglas indicadas previamente en este documento. En otras palabras, el grupo debe presentar un programa que demuestre lo realizado en el proyecto. Este programa no debe tener ningún problema de ejecución. Este programa deberá cumplir con lo estipulado anteriormente con respecto a las animaciones solicitadas. Prestar mucha atención en estos programas pues son fundamentales para la debida revisión.
11. El grupo deberá demostrar la funcionalidad del dispositivo llevando los elementos necesarios para que el dispositivo cumpla

con su función. Se solicita que puedan realizar el programa solicitado de manera que puedan demostrar todo lo implementado.

12. Dentro de la revisión del proyecto se deberá presentar la gramática creada y dar una explicación detallada de la manera en que se implementó el intérprete mostrando todos los elementos necesarios para entender su funcionalidad.
13. Todos los integrantes del grupo deberán estar presente en la defensa del proyecto, y todos deben estar preparados para contestar las preguntas que se les puede realizar en base a la implementación realizada, aunque se entiende que algunos se especializarán en secciones del proyecto, pero todos deben tener una noción general del mismo. Solamente en casos justificados y vistos previamente con por lo menos un día de anticipación se podrá ausentar algún miembro del equipo.
14. Aun cuando el código y la documentación tienen sus notas por separado, se aplican las siguientes restricciones
    a. Si no se entrega documentación, automáticamente se obtiene una nota de 0.
    b. Si la documentación no incluye el diagrama de arquitectura se obtiene una nota de 0
    c. Sí el código y la documentación no se entregan en la fecha indicada se obtiene una nota de 0.
    d. Sí el código no compila se obtendrá una nota de 0, por lo cual se recomienda realizar la defensa con un código funcional.
15. La revisión de la documentación será realizada por parte del profesor, no durante la defensa del proyecto. El único requerimiento que se consultará durante la defensa del proyecto es el diagrama de arquitectura, documentación interna y cualquier otra documentación que el profesor considere necesaria.
16. Cada grupo tendrá como máximo 25 minutos para exponer su trabajo al profesor y realizar la defensa de éste, es responsabilidad de los estudiantes mostrar todo el trabajo realizado, por lo cual se recomienda tener todo listo antes de ingresar a la defensa, y será responsabilidad del grupo administrar el tiempo dispuesto para su revisión de tal manera que el profesor pueda observar todo el producto terminado.
17. Todo error que salga durante la ejecución del proyecto y que se considere debió haber sido contemplada durante el desarrollo del proyecto, se castigará con 2 puntos de la nota final del proyecto.
18. Cada grupo es responsable de llevar los equipos requeridos para la revisión, si no cuentan con estos deberán avisar al menos 2 días antes de la revisión a el profesor para coordinar el préstamo de estos.

19. No se revisarán funcionalidades incompletas o no integradas.
20. Durante la revisión únicamente podrán participar los miembros del grupo, asistentes, otros profesores y el coordinador del área.
21. Para la generación del análisis léxico y sintáctico se da la alternativa de utilizar Lex y Yacc (y sus derivados).

El trabajo tiene como máximo grupos de 4 personas pensando en que podría (y como sugerencia) haber especialistas en ciertas tareas de tal manera que se trabaje en equipo. Dados los requerimientos anteriormente mostrados, se esperaría los siguientes roles dentro del equipo de trabajo

- Enfoque en el Lenguaje de programación y el Compilador. Son las tareas de implementación del IDE, de la gramática, de los elementos de programación y la generación de cada una de las etapas del compilador.
- Interfaz y lógica del dispositivo de leds (matrices). Son las tareas de la etapa clásica de los compiladores de "generación de código", de tal manera que pueda tomar el código generado y validado por el compilador y forme las instrucciones necesarios para ser ejecutado dentro del dispositivo físico. También esta las tareas de la lógica que deben tener las listas y/o matrices para representar las imágenes solicitadas.
- Enfoque en aspectos físicos del hardware. Son las tareas de creación física del dispositivo, tanto la colocación de los leds como la lógica con el microcontrolador para representar las imágenes solicitadas.