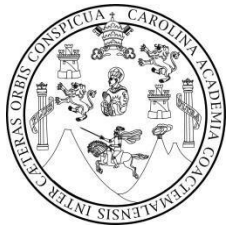
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
ESCUELA DE CIENCIAS Y SISTEMAS
LENGUAJES FORMALES Y DE PROGRAMACIÓN



Segundo Semestre 2018

Sección	Catedrática	Tutor académico
A-	Inga. Damaris Campos de López	Aylin Aroche
B-	Inga. Zulma Aguirre	César Solares

Enunciado de Proyecto No. 2

Objetivos:

Este proyecto tiene como objetivo que el estudiante practique el análisis léxico y el análisis sintáctico de un lenguaje formal y que amplíe sus conocimientos del lenguaje de programación C#, así como aplicar los conocimientos adquiridos en la clase y en el laboratorio.

Descripción:

La empresa DIS (Digital InGame Solutions) ha decidió lanzar al mercado un nuevo videojuego destinado al aprendizaje de la programación de una forma intuitiva y entretenida, para esto han contratado sus servicios.

La temática del juego consiste en resolver puzzles de laberintos, la creación de los niveles la hará el jugador ingresando comandos de programación de un lenguaje específicamente creado para este juego, dichos comandos servirán para todo el funcionamiento del juego que va desde el diseño del nivel hasta el flujo del juego mismo. El juego también contara con enemigos que tendrán un comportamiento sencillo y se podrán crear enemigos con un máximo de 3 por cada nivel y si el jugador hace contacto con cualquiera de los enemigos será GAME OVER.

Los archivos de texto plano que almacenan el código fuente que recibe la aplicación están escritos en un lenguaje formal que será llamado DIS, el nombre hace referencia al nombre de la empresa.

La aplicación debe proveer la opción de cargar archivos de texto con la extensión .DIS a un editor, y también debe existir la opción de analizar el contenido del editor para cargar el laberinto y la ruta que el personaje del juego debe recorrer para llegar al final del laberinto, además cuenta con la opción de iniciar recorrido, para que el personaje recorra la ruta que fue definida en el lenguaje cargado. En caso de que el texto contenga errores, se mostrará una página HTML con el detalle de estos.

Definición del lenguaje DIS:

Dentro del lenguaje propio de la aplicación es posible indicar que se desea definir un laberinto y una ruta de recorrido con el siguiente lenguaje:

- **Bloque principal**

```
[Principal]:{
[Intervalo]:(<duración>);
        <Bloque_Nivel>
      <Bloque_Enemigo>
      <Bloque_Personaje>
}
```

Cada archivo DIS, tiene un solo bloque principal que contiene varios bloques:

- El bloque **laberinto** contiene toda la información necesaria para cargar el laberinto del juego.
- El bloque **personaje** contiene la información sobre la ruta que debe seguir el personaje para resolver el laberinto.
- El bloque **enemigo** contiene la información sobre el comportamiento del enemigo, de este último pueden venir un máximo de 3 bloques o no venir.
- ***Intervalo***: Indica el intervalo de tiempo entre cada paso que el personaje y los enemigos utilizan para avanzar.
  - ***<duración>***: Se mide en milisegundos e indica la duración del intervalo.

- **Bloque Nivel**

Define el escenario del nivel.

```
[Nivel]: {
      [Dimensiones]:(<tam_x>, <tam_y>);
      [Inicio_personaje]:(<Pos_x>, <Pos_y>);
      [Ubicación_Salida]:(<PosV_x>, <PosV_y>);
      [Pared]: {
            [Casilla]:(<x>, <y>);
           [Varias_Casillas]: (<x1>..<x2>, <y>);
           [Varias_Casillas]: (<x>, <y1>..<y2>);
     [Varias_Casillas]: (<x1>..<x2>, <y1>..<y2>);
            ...
      }
}
```

A continuación, se detallan los elementos del bloque ***laberinto***:

- ***Dimensiones***: indica las dimensiones del laberinto, es decir el número de casillas que tendrá (el tamaño del escenario), este aparece solamente una vez en el bloque laberinto.
  - ***<tam_x>***: Indica el tamaño del escenario en el eje horizontal, es un número entero mayor o igual que 3 y menor o igual que 20.
  - ***<tam_y>***: Indica el tamaño del escenario en el eje vertical, es un número entero mayor o igual que 3 y menor o igual que 15.

- ***Inicio_personaje***: Indica la posición inicial del personaje dentro del laberinto.
  - ***<Pos_x>***: Indica la posición en x del personaje dentro del laberinto.
  - ***<Pos_y>***: Indica la posición en y del personaje dentro del laberinto.
- ***Ubicación Victoria***: Indica la posición a la que deberá llegar el personaje para ganar el nivel.
  - ***<PosV_x>***: Indica la posición en x para ganar el nivel.
  - ***<PosV_y>***: Indica la posición en y para ganar el nivel.

- ***Pared***: Dentro del este bloque se indica la localización de las múltiples casillas que representan una pared por la que el personaje no podrá avanzar dentro del laberinto, dentro de este bloque está permitido el uso de variables, más adelante se indicará cómo.
  - ***Varias Casillas***: Es un rango de casillas que serán paredes para el personaje en el laberinto, existen tres formas de definir un rango de casillas:
    - `[Varias_Casillas]: (<x1>..<x2>, <y>);`
      Esta forma de definirla indica que habrá una línea de bloques que conformaran una pared horizontal, sobe la posición <y> que va de la posición <x1> a la posición <x2> en el eje horizontal.
    - `[Varias_Casillas]: (<x>, <y1>..<y2>);`
      Esta forma de definirla indica que habrá una línea de bloques que conformaran una pared vertical, sobe la posición <x> que va de la posición <y1> a la posición <y2> en el eje vertical.
    - `[Varias_Casillas]: (<x1>..<x2>, <y1>..<y2>);`
      Esta forma de definirla indica que habrá un cuadro de bloques que conformaran una pared cuya esquina superior izquierda está en la coordenada (x1, y1) y su esquina inferior derecha está en la coordenada (x2, y2).
  - ***Casilla:*** Esta instrucción indica que habrá una pared en la casilla con coordenadas (x, y).

- **Bloque Enemigo**

  En este bloque se define la ruta que el enemigo tendrá que seguir durante el juego, esta ruta será unidimensional.

```
[Enemigo]: {
       [Caminata]: (<x1>..<x2>, <y>);
       [Caminata]: (<x>, <y1>..<y2>);
}
```

  A continuación, se detallan los elementos del bloque ***Enemigo***:

  - ***Caminata***: Es un rango de casillas que el enemigo recorrerá a lo largo del laberinto, por las que puede pasar, es decir, en las que no hay una pared.

    Existen dos formas de definir una caminata y el enemigo solo podrá tener una de las dos formas:

    - `[Caminata]: (<x1>..<x2>, <y>);`

      Esta forma de definirla indica que el enemigo caminara en línea recta, horizontalmente sobre la posición <y> desde <x1> hasta <x2>. Luego regresara desde la posición <x2> hasta <x1>.
    - `[Caminata]: (<x>, <y1>..<y2>);`

      Esta forma de definirla indica que el enemigo caminara en línea recta, verticalmente sobre la posición <x> desde <y1> hasta <y2>. Luego regresara desde la posición <y2> hasta <y1>.

- **Bloque personaje**

  En este bloque se define la ruta que el personaje debe recorrer para llegar a la casilla de Salida y así resolver el laberinto, en este bloque también se permite el uso de variables. Si la ruta definida no lleva a la solución la aplicación debe mostrar un mensaje de que no se resolvió el laberinto luego de haber recorrido la ruta definida, de igual manera si se resolviera, debe mostrarse el mensaje de que se resolvió.

```
[Personaje]: {

       [Paso]:(<x>, <y>);
       [Caminata]: (<x1>..<x2>, <y>);
       [Caminata]: (<x>, <y1>..<y2>);
       ...
}
```

A continuación, se detallan los elementos del bloque ***personaje***:

- ***Caminata***: Es un rango de casillas que el personaje recorrerá a lo largo del laberinto, por las que puede pasar, es decir, en las que no hay pared.
  Existen dos formas de definir una caminata:
  - `[Caminata]: (<x1>..<x2>, <y>);`
    Esta forma de definirla indica que el personaje caminara en línea recta, horizontalmente sobre la posición <y> desde <x1> hasta <x2>.
  - `[Caminata]: (<x>, <y1>..<y2>);`
    Esta forma de definirla indica que el personaje caminara en línea recta, verticalmente sobre la posición <x> desde <y1> hasta <y2>.
- ***Paso:*** Esta instrucción indica que el personaje debe dar un paso desde su posición actual hasta la posición con coordenadas (x, y), respetando las reglas de movimiento del juego.

- **Uso de variables**

Las variables pueden utilizarse únicamente dentro de los bloques personaje y pared.

- ***Declaración de variables***
  Todas las variables declaradas son de tipo entero y en una sentencia de declaración se puede declarar más de una variable. El ámbito de las variables se restringe al bloque en el que son declaradas, es decir, las variables declaradas en el bloque Nivel no pueden ser utilizadas en el bloque personaje y viceversa. La sintaxis es la siguiente:

```
[Variable]:<id>;
[Variable]:<id>,<id>, ... ,<id>;
```

- ***Asignación de variables***
  A todas las variables se les puede asignar un valor entero, el valor de otra variable o una expresión aritmética:

```
<id>:=<id>;
<id>:=<número_entero>;
<id>:=<expresión_aritmética>;
```

  - ***<id>:*** Un identificador es una secuencia de una letra seguida de cero o muchas letras o dígitos, su expresión regular sería: L(L|D)*
  - ***<número_entero>:*** Cualquier número entero mayor o igual que cero, no se toman en cuenta los números negativos porque las variables serán utilizadas para definir coordenadas y dentro de las coordenadas no se admiten números negativos.
  - ***<expresión_aritmética>*** Es cualquier expresión aritmética de dos operandos y un operador, que admite únicamente variables o números enteros mayores o iguales a cero. Los operadores válidos son suma, resta, multiplicación y división con la precedencia típica de estos operadores.

El que solo se puedan tener dos operandos limita un poco el lenguaje, esto es así porque los usuarios de la aplicación desean aprender programación básica.

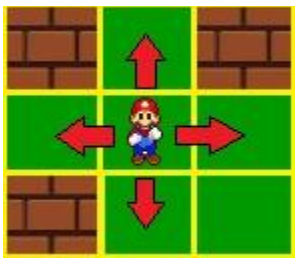
Por ejemplo, si se quisiera hacer la operación: 1*(2+3)/5 En el lenguaje DIS haría falta algo como esto:

```
[Variable]:T1,T2,T3;
T1:=2+3;
T2:=1*T1;
T3:=T2/5;
```

El resultado quedaría almacenado en la variable T3.

- **Reglas de movimiento**

  El personaje solo puede ejecutar un paso a la vez, hacia arriba, abajo, izquierda o derecha, siempre que no haya una pared que se lo impida. Como se muestra en la siguiente figura:



- **Opción recorrer ruta**

  Luego de cargar el archivo de entrada, la aplicación debe mostrar el laberinto cargado, en el que se vea al personaje en su posición inicial y al tesoro en el lugar que le corresponde.
  Luego de cargado el laberinto la aplicación debe tener la opción recorrer ruta, al seleccionarla el personaje debe recorrer paso por paso la ruta definida en el archivo de entrada, el periodo de tiempo entre cada paso será el que se definió en el bloque principal, en la instrucción intervalo. Si el personaje alcanza la salida y resuelve el laberinto debe mostrarse un mensaje de éxito, si no lo alcanza o se topa con alguno de los enemigos debe mostrarse un mensaje de fracaso.

- **Opción resolver laberinto**

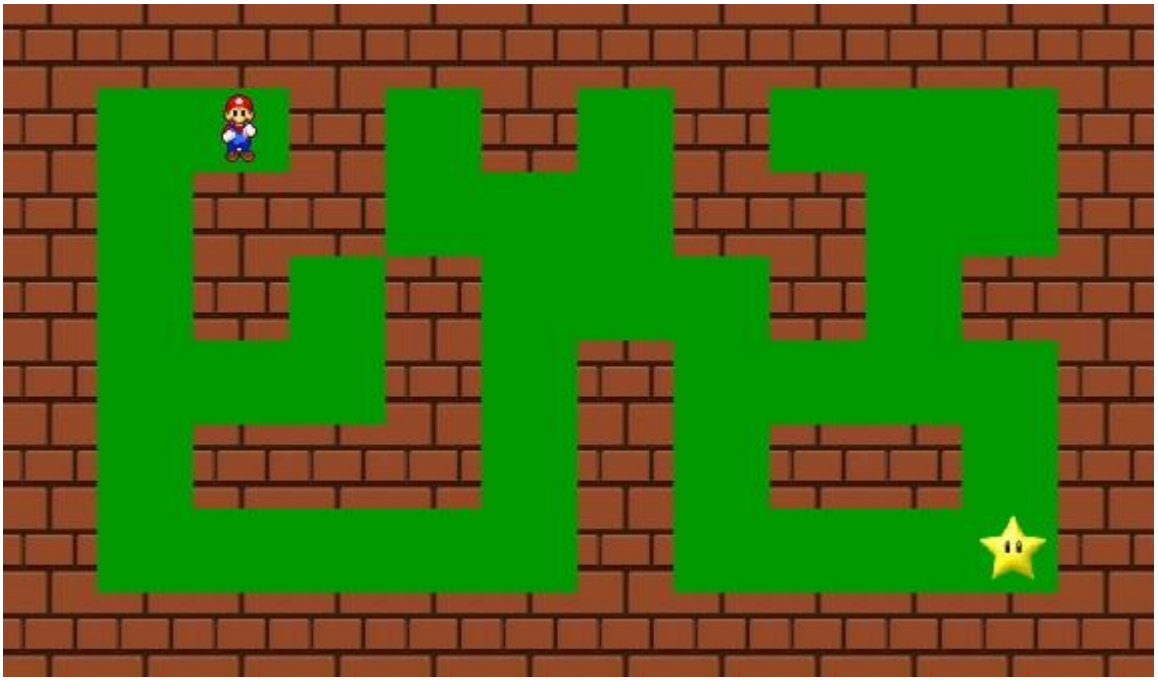
  Una vez cargado el laberinto, debe existir una opción “Resolver Laberinto”, que encuentra automáticamente una ruta para resolver el laberinto.
  Para esto se pide al usuario la ruta de un archivo de texto en el que se almacenará una secuencia de instrucciones ***paso*** que definen una ruta para el personaje, que lo lleva desde su posición inicial hasta la posición en la que se encuentra el tesoro, es decir, esta secuencia de instrucciones ***paso*** resuelven el laberinto.
  Esto para que el usuario pueda incrustar esta secuencia en un archivo .DIS que podría utilizarse en la aplicación.

Archivo de entrada ejemplo:

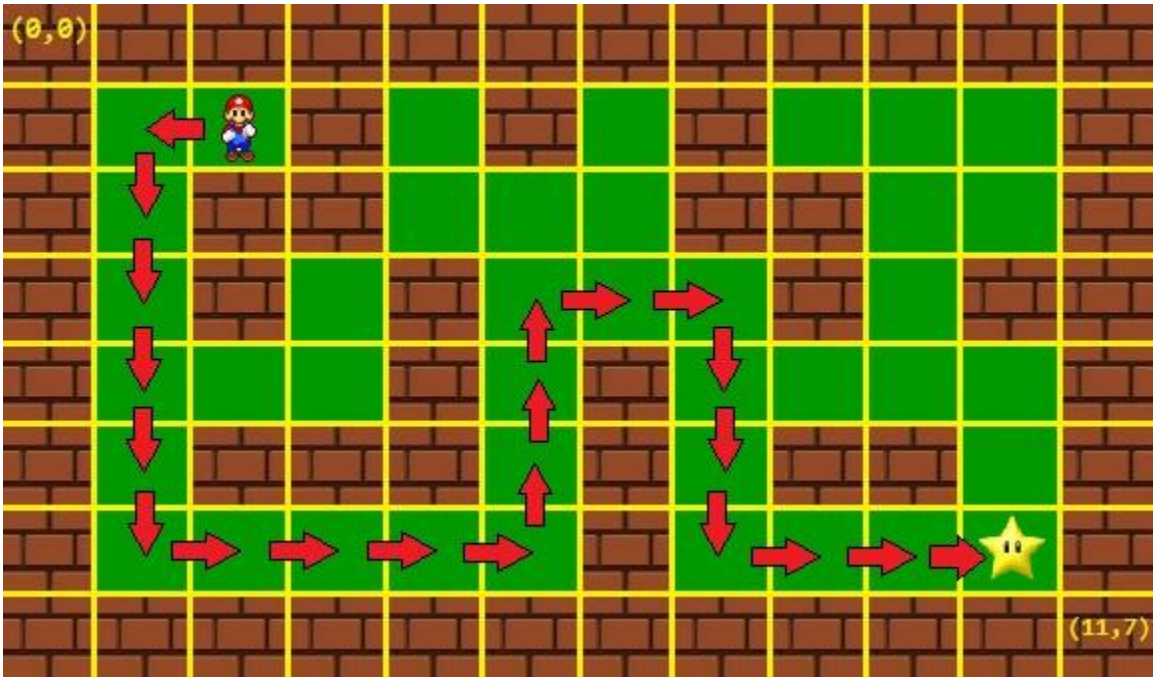
```
[Principal]: {
             [Intervalo]:(1000);
       [Nivel]: {
             [Dimensiones]:(12, 8);
              [Inicio_personaje]:(2, 1);
```

```
            [Ubicación_Salida]:(10, 6);
              [Pared]: {
                  [Variable]:a,b,c,d;
                   a:=0;
                   [Rango_Casillas]:(a..11, a);
                   [Rango_Casillas]:(a..11, 7);
                   a:=a+11;
                   [Rango_Casillas]:(0, 1..6);
                  [Rango_Casillas]:(a, 1..6);
                   b:=a-11;
                   c:=b+3;
                   d:=c-1;
                   [Casilla]:(c, 1);
                   [Casilla]:(c, d);
                   [Casilla]:(d, d);
                   [Casilla]:(d, c);
                   [Casilla]:(5, 1);
                   [Casilla]:(10, 3);
                   [Casilla]:(8, 2);
                   [Casilla]:(8, 3);
                   [Casilla]:(7, 1);
                   [Casilla]:(7, 2);
                   b:=b+5;
                   [Rango_Casillas]:(6, 4..6);
                   [Rango_Casillas]:(4, 3..b);
                   [Rango_Casillas]:(2..3, b);
                   [Rango_Casillas]:(8..9, b);
            }
      }
    [Personaje]:{
             [Variable]:e;
              e:=1;
             [Paso]:(e, e);
             [Caminata]:(e, 2..6);
             [Caminata]:(2..5, 6);
             [Caminata]:(5, 5..3);
             [Paso]:(6, 3);
             [Paso]:(7, 3);
             [Caminata]:(7, 4..6);
             [Caminata]:(8..9, 6);
             [Paso]:(10, 6);
      }
}
```

Al ejecutar las instrucciones anteriores en el editor, se generaría un laberinto como el siguiente:



Al ejecutar la opción recorrer ruta, el personaje tendría que caminar siguiendo la ruta definida en el archivo de entrada, a continuación, se muestra una imagen con la ruta:



Aplicación:

La aplicación debe analizar el texto del editor, de encontrar errores léxicos o sintácticos, la aplicación debe informar al usuario y mostrar un reporte de errores en formato HTML, en este caso no debe generar la página de salida. En la tabla con el reporte de errores se deben mostrar los siguientes campos:

No.	Error	Tipo	Descripción	Fila	Columna
1	%	Léxico	Elemento léxico desconocido	3	4
2	(	Sintáctico	Se esperaba [	6	10

De no encontrar error alguno, se debe generar el laberinto correspondiente. La aplicación debe pintar dentro del editor cada uno de los lexemas, según su tipo, los colores se definen a continuación:

Tipo	Color
Palabras reservadas	Azul
Números	Rojo
Dos puntos	Amarillo
Punto y coma	Morado
Llaves	Rosado
Corchetes	Celeste
Otros	Negro

Además de colorear los elementos léxicos y generar el laberinto de salida, se debe generar otra página HTML que contenga una Tabla de tokens con los siguientes campos:

No.	Lexema	Tipo	Fila	Columna
1	Nivel	Reservada	1	1
2	0	Numero Entero	1	8
3	:	Dos puntos	1	12

Interfaz gráfica:

Componentes mínimos de la interfaz:

1. **Editor de texto:** Debe ser un área de texto en el cual se puedan escribir nuevas instrucciones de entrada o bien cargar texto desde archivos con extensión .DIS
2. **Abrir:** La interfaz debe proveer la capacidad de abrir archivos con extensión .DIS.
3. **Guardar:** La aplicación debe proveer la capacidad de guardar el texto contenido en el editor. Si no ha sido guardada nunca, debe proveer una opción para ingresar el nombre del nuevo archivo y la ubicación en la que se desea guardar.
4. **Guardar como:** Está opción permite guardar el archivo de entrada con otro nombre, se debe preguntar el nombre del nuevo archivo.
5. **Analizar:** Debe realizar el análisis léxico y sintáctico del lenguaje que se encuentra actualmente en el editor y generar la salida correspondiente.
6. **Tablero de juego:** Es una ventana en la que aparece el laberinto cargado y en la que se ejecuta el recorrido del personaje según la ruta definida en el archivo de entrada.
   Nota: Las gráficas mostradas en el presente enunciado son solamente un ejemplo, el estudiante puede escoger los gráficos del laberinto (personaje, Pared y camino) que desee.
7. **Acerca de:** Debe desplegar una ventana con los datos del estudiante y del curso.
8. **Salir:** Debe terminar la ejecución de la aplicación.

Entregables:

- Manual de Usuario
- Manual Técnico
    - Dentro de este manual debe incluirse el autómata finito determinista que se utilizó para la implementación del analizador léxico. Para la obtención del autómata se debe de utilizar expresiones regulares y el método del árbol los cuales también deben de adjuntarse en este documento.
    - Deben de adjuntarse de igual manera la gramática independiente del contexto utilizada para la implementación del analizador sintáctico.
- Código fuente.
- Ejecutable del proyecto

Documentación a entregar de forma física el día de la calificación:

- Hoja de calificación (Original y una copia)

Notas importantes

- El proyecto se debe desarrollar de forma individual.
- Este proyecto se deberá desarrollar utilizando C# con Visual Studio.
- No se deberá de utilizar ninguna herramienta auxiliar.
  El proceso de obtener tokens, se debe hacer a través de la implementación del autómata finito determinista desarrollado por el propio estudiante.
- El analizador sintáctico debe ser programado por el estudiante, y el código fuente debe coincidir con la gramática tipo 2 que presente en el manual técnico.
- No se puede agregar o quitar algún símbolo en el archivo de entrada. El proyecto deberá funcionar con los archivos de prueba que se disponga para la calificación, sin modificación.
- La calificación del proyecto será personal y durará como máximo 30 minutos, en un horario que posteriormente será establecido. Se debe tomar en cuenta que durante la calificación no podrán estar terceras personas alrededor, de lo contrario no se calificará el proyecto.
- No se dará prórroga para la entrega del proyecto.
- **Copia parcial o total del proyecto tendrá una nota de 0 puntos, y se notificará a la escuela de sistemas para que se apliquen las sanciones correspondientes.**
- **En el caso de no cumplir con alguna de las indicaciones antes mencionadas, NO se calificará el proyecto; por lo cual, se tendrá una nota de cero puntos.**

Fecha de entrega: 27 de octubre de 2018

Anexos

- Información sobre el algoritmo de Tremaux para encontrar la salida de un laberinto, se recomienda el uso de este algoritmo porque para implementarlo no hace falta utilizar estructuras de datos complicadas, sino estructuras que ya conocen, como matrices y pilas:
    - http://es.wikipedia.org/wiki/Algoritmo_de_Tremaux