Universidad De San Carlos De Guatemala
Facultad De Ingeniería
Escuela De Ciencias Y Sistemas
Lenguajes Organización de Lenguajes y Compiladores 1
Catedrático: Ing. Mario Bautista
Tutor académico: David Pascual

Proyecto 2

OBJETIVOS:

- Que el estudiante practique el desarrollo del análisis léxico y análisis sintáctico utilizando la herramienta Irony.
- Aplicar los conocimientos del curso de Organización de Lenguajes y Compiladores 1 en la creación de soluciones de software.
- Aplicar los conceptos de compiladores para implementar el proceso de Interpretación de código de alto nivel.

DESCRIPCIÓN DEL PROBLEMA:

La empresa de desarrollo de software OLCEE requiere la elaboración de una herramienta que Interpretará un código de programación de alto nivel y de esta forma será capaz de generar la Interpretación correspondiente así como los diagramas de flujos de los métodos y/o funciones solicitados. La aplicación recibirá el nombre de "Flow Charts IDE". La aplicación contará con un entorno de desarrollo, el cual contará con editor y un área gráfica para poder visualizar los diagramas de flujo, en la cual se mostrarán las imágenes creadas durante la Interpretación del código. Esta área gráfica se mostrará en una ventana independiente al editor para adaptarse al tamaño de la imagen a mostrar.

El editor contará con un directorio de Proyectos (Treeview) y varias funcionalidades que harán un uso más fácil para el usuario, las cuales serán: crear, modificar y guardar archivos, numeración de líneas y múltiples pestañas de edición en paralelo. El lenguaje que deberá ser Interpretado se basa en la programación estructurada.

MÓDULOS DE LA APLICACIÓN:

Interprete:

El entorno de trabajo tendrá un Intérprete de código, ejecutando las instrucciones del código analizado generando resultados obtenidos por consola, diagramas de flujos o el reporte de errores en caso que existan. El Intérprete deberá ser capaz de ejecutar todas las instrucciones del lenguaje.

Editor:

El editor será parte del entorno de trabajo, cuya finalidad será proporcionar ciertas funcionalidades, características y herramientas que serán de utilidad al usuario. La función principal es el manejo de archivos de código fuente extensión .olc que serán Interpretados.

El editor deberá contar con lo siguiente:

- **Directorio de Proyectos:** El editor deberá presentar una estructura de directorios.
- **Crear nuevo archivo**: El editor deberá ser capaz de crear archivos en blanco.
- **Abrir archivos**: El editor deberá abrir archivos en formato .olc
- **Guardar el archivo**: El editor deberá guardar el estado del archivo en el que se estará trabajando.
- **Múltiple pestañas**: El editor deberá ser capaz de crear todas las pestañas que el usuario desee.
- **Enumeración líneas**: El editor deberá tener un contador de líneas de código ingresado.
- **Ejecutar**: Invocará al Intérprete

Reportes:

La aplicación deberá poder generar reporte de errores, estos errores deben ser presentados en un archivo en formato HTM, los tipos de errores se detallara más adelante.

Área Grafica:

El área gráfica será la parte en blanco de una ventana en donde se mostrará los diagramas de flujo de todos los procedimientos graficado por la instrucción en alto nivel que diagrama el flujo de los procedimientos, la estructura de los diagramas de flujo se detallara más adelante.

Área de Resultados por Consola:

La aplicación deberá poder generar resultados por consola de acuerdo a la instrucción en alto nivel que imprime mensajes por consola.

DESCRIPCION DEL LENGUAJE:

El lenguaje de alto nivel es un lenguaje programación estructurada tomando como base el lenguaje de programación Pascal, **el cual es sensible a mayúsculas y minúsculas**. Los archivos en los que se guardan los programas escritos en este lenguaje tienen extensión .olc

Para una mejor compresión en la descripción, se usara una notación de colores para definir los diferentes tipos de sentencias, la cual se describe a continuación:

COLOR	DESCRIPCIÓN
NARANJA	Opcional, esta parte de la estructura puede o no venir en las sentencias del lenguaje. La parte opcional será encerrada entre corchetes "[ ]".
NEGRO	Obligatorio, esta parte de la estructura es obligatorio que aparezca en la sentencia.
AZUL	Palabras reservadas que pertenecen al lenguaje.
VERDE	Ejemplos.

Tipo de Datos:

Entendemos a un tipo de dato como la restricción que se le asigna a una variable, matriz o función sobre los datos que puede contener o retornar y las acciones que se pueden realizar sobre él. **La tipificación del lenguaje de alto nivel es tipificación fuerte**, se deben declarar los tipos de datos asociados a los elementos del lenguaje y este tipo de datos no cambiará en todo el código.

Los tipos de datos permitidos son los siguientes:

- INT
- DOUBLE
- STRING
- CHAR
- BOOL

Especificaciones sobre los tipos de datos:

- **INT**: tipo de dato que acepta valores numéricos enteros en base decimal (base 10), su valor por default será 0.
  Ejemplos: -85, 0, 15, 1589.

- **DOUBLE**: tipo de dato que acepta valores numéricos con punto flotante de doble precisión en base decimal (base 10), su valor por default será 0.0.
  Ejemplos: -5.68, 3.141592, 0.008.

- **STRING**: tipo de dato que acepta cadenas de caracteres alfanuméricas. Un dato de tipo STRING debe estar escrito entre comillas dobles, su valor por default será cadena vacía "".
  Ejemplo: "Este es un ejemplo de una cadena STRING".

- **CHAR**: tipo de dato que acepta un carácter alfanumérico. Un dato de tipo CHAR puede estar escrito entre comillas simples o puede recibir un número entero entre 0 y 255, en este caso el número será convertido al carácter equivalente del código ASCII, su valor por default será el carácter de espacio ' '. Un tipo de dato CHAR también puede ser utilizado como un dato entero, en este caso el carácter deberá ser convertido a su código ASCII correspondiente, la decisión de la forma de usar un tipo de dato CHAR se tomará según el ámbito en el que se encuentre. Ejemplo de CHAR: 'a', '1', 23, '?', 55.

- **BOOL**: tipo de dato que acepta un valor lógico verdadero o falso. Un tipo de dato BOOL también puede ser utilizado como un dato entero.
  Datos aceptados como valores lógicos verdadero y su equivalencia: true, 1.
  Datos aceptados como valores lógicos falso y su equivalencia: false, 0.

Expresiones de Operaciones Aritméticas:

Una operación aritmética es un conjunto de reglas que permiten obtener otras cantidades o expresiones a partir de datos específicos. A continuación se definen las operaciones aritméticas soportadas por el lenguaje.

Suma:

Operación aritmética que consiste en reunir varias cantidades (sumandos) en una sola (la suma). El operador de la suma es el signo más +

<u>Especificaciones sobre la suma:</u>

- Al sumar dos datos numéricos (INT, DOUBLE, CHAR, BOOL) el resultado será numérico.
- Al sumar dos datos de tipo carácter (CHAR, STRING) el resultado será la concatenación de ambos datos.
- Al sumar un dato numérico con un dato de tipo carácter el resultado será la concatenación de del dato de tipo carácter y la conversión a cadena del dato numérico.
- Al sumar dos datos de tipo lógico (BOOL) el resultado será un dato lógico, en este caso utilizaremos la suma como la operación lógica or.

Sistema de tipos para la suma:

Operandos	Tipo de dato resultante	Ejemplos
INT + DOUBLE DOUBLE + INT DOUBLE + CHAR CHAR + DOUBLE BOOL + DOUBLE DOUBLE + BOOL DOUBLE + DOUBLE	DOUBLE	5 + 4.5 = 9.5 7.8 + 3 = 10.8 15.3 + 'a' = 112.3 'b' + 2.7 = 100.7 true + 1.2 = 2.2 4.5 + false = 4.5 3.56 + 2.3 = 5.86
INT + CHAR CHAR + INT BOOL + INT INT + BOOL INT + INT	INT	7 + 'c' = 106 'C' + 7 = 74 4 + true = 5 4 + false = 4 4 + 5 = 9
STRING + INT STRING + DOUBLE DOUBLE + STRING INT + STRING	STRING	"hola" + 2 = "hola2" "hola" + 3.5 = "hola3.5" 4.5 + "hola" = "4.5hola" 8+"hola" = "8hola"
STRING + CHAR CHAR + STRING STRING + STRING	STRING	"hola"+'t' = "holat"'u'+"hola" = "uhola" "hola"+ "mundo" = "holamundo"
BOOL + BOOL	BOOL	true + true = true false + false = false

Cualquier otra combinación es inválida y deberá arrojar un error de semántica.

Resta:

Operación aritmética que consiste en quitar una cantidad (sustraendo) de otra (minuendo) para averiguar la diferencia entre las dos. El operador de la resta es el signo menos –

Especificaciones sobre la resta:

- Al restar dos datos numéricos (INT, DOUBLE, CHAR, BOOL) el resultado será numérico.
- No es posible restar datos numéricos con tipos de datos de tipo carácter (STRING).
- No es posible restar tipos de datos carácter (CHAR, STRING) entre sí.
- No es posible restar tipos de datos lógicos (BOOL) entre sí.

Sistema de tipos para la resta:

Operandos	Tipo de dato resultante	Ejemplos
INT - DOUBLE DOUBLE - INT DOUBLE - CHAR CHAR - DOUBLE BOOL - DOUBLE DOUBLE - BOOL DOUBLE - DOUBLE	DOUBLE	5-4.5 = 0.5 7.8-3 = 4.8 15.3 - 'a' = -81.7 'b' - 2.7 = 95.3 true - 1.2 = -0.2 4.5 - false = 4.5 3.56 - 2.3 = 1.26
INT - CHAR CHAR - INT BOOL - INT INT - BOOL INT - INT	INT	7 - 'c' = -92 'C' - 7 = 60 4 - true = 3 4 - false = 4 4 - 5 = -1

Cualquier otra combinación es inválida y deberá arrojar un error de semántica.

División:

Operación aritmética que consiste en partir un todo en varias partes, al todo se le conoce como dividendo, al total de partes se le llama divisor y el resultado recibe el nombre de cociente. El operador de la división es la diagonal /

Especificaciones sobre la división:

- Al dividir dos datos numéricos (INT, DOUBLE, CHAR, BOOL) el resultado será numérico.
- No es posible dividir datos numéricos con tipos de datos de tipo carácter (STRING).
- No es posible dividir tipos de datos carácter (CHAR, STRING) entre sí.
- No es posible dividir tipos de datos lógicos (BOOL) entre sí.
- Al dividir un dato numérico entre 0 deberá arrojarse un error de ejecución.

Sistema de tipos para la división:

Operandos	Tipo de dato resultante	Ejemplos
INT / DOUBLE DOUBLE / INT DOUBLE / CHAR CHAR / DOUBLE BOOL / DOUBLE DOUBLE / BOOL DOUBLE / DOUBLE INT / CHAR CHAR / INT BOOL / INT INT / BOOL INT / INT	DOUBLE	5/4.5 = 1.11111 7.8/3 = 2.6 15.3 / 'a' =0.1577 'b' / 2.7 = 28.8889 true / 1.2 = 0.8333 4.5 / false = error 3.56 / 2.3 = 1.5478 7 / 'c' = 0.7070 'C' / 7 = 9.5714 4 / true = 4.0 4 / false = error 4 / 5 = 0.8

Multiplicación:

Operación aritmética que operación aritmética que consiste en sumar un número (multiplicando) tantas veces como indica otro número (multiplicador). El operador de la multiplicación es el asterisco *

Especificaciones sobre la multiplicación:

- Al multiplicar dos datos numéricos (INT, DOUBLE, CHAR, BOOL) el resultado será numérico.
- No es posible multiplicar datos numéricos con tipos de datos carácter (STRING).
- No es posible multiplicar tipos de datos carácter (CHAR, STRING) entre sí.
- Al multiplicar dos datos de tipo lógico (BOOL) el resultado será un dato lógico, en este caso usaremos la multiplicación como la operación AND entre ambos datos.

Sistema de tipos para la multiplicación:

Operandos	Tipo de dato resultante	Ejemplos
INT * DOUBLE DOUBLE * INT DOUBLE * CHAR CHAR * DOUBLE BOOL * DOUBLE DOUBLE * BOOL DOUBLE * DOUBLE	DOUBLE	5*4.5 = 22.5 7.8*3 = 23.4 15.3 * 'a' = 1484.1 'b' * 2.7 = 264.6 true * 1.2 = 1.2 4.5 * false = 0 3.56 * 2.3 = 8.188
INT * CHAR CHAR * INT BOOL * INT INT * BOOL INT * INT	INT	7 * 'c' = 693 'C' * 7 = 469 4 * true = 4 4 * false = 0 4 * 5 = 20
BOOL * BOOL	BOOL	true * true = true false * false = false

Cualquier otra combinación es inválida y deberá arrojar un error de semántica.

Potencia:

Operación aritmética que consiste en multiplicar varias veces un mismo factor. El operador de la potencia es el acento circunflejo ^

Especificaciones sobre la potencia:

- Al potenciar dos datos numéricos (INT, DOUBLE, CHAR, BOOL) el resultado será numérico.
- No es posible potenciar datos numéricos con tipos de datos de tipo carácter (STRING).
- No es posible potenciar tipos de datos carácter (CHAR, STRING) entre sí.
- No es posible potenciar tipos de datos lógicos (BOOL) entre sí.
- **La potencia tiene asociatividad derecha, el resto de operaciones aritméticas tienen asociatividad izquierda.**

Sistema de tipos para la potencia:

Operandos	Tipo de dato resultante	Ejemplos
INT ^ DOUBLE DOUBLE ^ INT DOUBLE ^ CHAR CHAR ^ DOUBLE BOOL ^ DOUBLE DOUBLE ^ BOOL DOUBLE ^ DOUBLE	DOUBLE	5^4.5 = 1397.54 7.8^3 = 474.55 'b' ^ 2.7 = 237853.96 true ^ 1.2 = 1.0 4.5 ^ false = 1.0 3.56 ^ -2.3 = 0.0539
INT ^ CHAR CHAR ^ INT BOOL ^ INT INT ^ BOOL INT ^ INT	INT	'C' ^ 7 = 6060711605323 4 ^ true = 4 4 ^ false = 1

Cualquier otra combinación es inválida y deberá arrojar un error de semántica.

Expresiones de Operaciones Relacionales:

Una operación relacional es una operación de comparación entre dos valores, el cuál siempre devuelve un valor de tipo lógico (BOOL) según el resultado de la comparación. Una operación relacional es binaria, es decir tiene dos operandos siempre.

Especificaciones sobre las operaciones relacionales:

- Es válido comparar datos numéricos (INT, DOUBLE) entre sí, la comparación se realizará utilizando el valor numérico entero con signo de cada dato.
- Es válido comparar cadenas de caracteres (CHAR, STRING) entre sí, la comparación se realizará sobre el resultado de sumar el código ASCII de cada uno de los caracteres que forman la cadena.
- Es válido comparar datos numéricos con datos de carácter ('CHAR') en este caso se hará la comparación del valor numérico entero con signo con el valor del código ASCII del carácter.

- No es válido comparar cadenas de caracteres (STRING) o datos numéricos (DOUBLE, INT, CHAR) con valores lógicos (BOOL).
- No es válido comparar valores lógicos (BOOL) entre sí.

Operaciones relacionales:

Operación Relacional	Ejemplos
>	5>4 = true 4.5>6 = false “abc” > “abc” = false ‘a’ > “a” = false 97 > ‘a’ = false
<	5<4 = false 4.5<6 = true “abc” < “abc” = false ‘a’ < “a” = false 97 < ‘a’ = false
>=	5>=4 = true 4.5>=6 = false “abc” >= “abc” = true ‘a’ >= “a” = true 97 >= ‘a’ = true
<=	5<=4 = false 4.5<=6 = true “abc” <= “abc” = true ‘a’ <= “a” = true 97 <= ‘a’ = true
==	5==4 = false 4.5==6 = false “abc” == “abc” = true ‘a’ == “a” = true 97 == ‘a’ = true
!=	5!=4 = true 4.5!=6 = true “abc” != “abc” = false ‘a’ != “a” = false 97 != ‘a’ = false

Expresiones de Operaciones Lógicas:

Las operaciones lógicas son expresiones matemáticas cuyo resultado es un valor lógico (BOOL). Las operaciones lógicas se basan en el álgebra de Boole.

Especificaciones sobre las operaciones lógicas:

- No es válido realizar una operación lógica sobre dos datos de tipo numérico (INT, DOUBLE, CHAR) **a menos que su equivalencia a entero sea 0 o 1 realizando su equivalencia a false o true respectivamente.**
- No es válido realizar una operación lógica sobre dos datos de tipo cadena (STRING).
- Si es válido realizar una operación lógica sobre dos datos de tipo lógico.
- La operación lógica NOT tiene solo un operando, el resto de operaciones lógicas debe tener dos operandos.
- **La operación lógica NOT tiene asociatividad derecha, el resto de operaciones lógicas tienen asociatividad izquierda.**

Operaciones lógicas:

<table>
<tr><th>Operación lógica</th><th>Operador</th><th>Tabla de Verdad</th></tr>
<tr><td>OR</td><td>||</td><td>
<table>
<tr><th>Op1</th><th>Op2</th><th>Op1 || Op2</th></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
</table>
</td></tr>
<tr><td>AND</td><td>&&</td><td>
<table>
<tr><th>Op1</th><th>Op2</th><th>Op1 && Op2</th></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
</table>
</td></tr>
<tr><td>NOT</td><td>!!</td><td>
<table>
<tr><th>Op1</th><th>!!Op1</th></tr>
<tr><td>1</td><td>0</td></tr>
<tr><td>0</td><td>1</td></tr>
</table>
</td></tr>
</table>

Precedencia de operadores aritméticos, relacionales, lógicos y signos de agrupación:

El único símbolo de agrupación válido para operaciones son los paréntesis ( ) los cuales sirven para dar mayor jerarquía y orden a las operaciones, la precedencia de los operadores va de menor a mayor según su aparición en la siguiente tabla:

Operadores
OR
AND
NOT
> < >= <= == !=
+ -
* /
^

Expresiones:

Consideraremos para este enunciado que cuando se haga referencia a una 'expresión', se hace referencia a cualquier operación que devuelve un valor, ya sea una operación aritmética, una operación relacional, una operación lógica, un atributo, variable, llamada a una función, acceso a una posición de una matriz, etc. Los tipos de datos para cada expresión vendrán dados por los sistemas de tipos definidos para las operaciones y del tipo de datos asociados al resto de elementos del lenguaje.

Ejemplo:

```
(!!(a>=b+llamada() ) && true || ((c==5) || false) || d<=4 && !!1  ) && 5>=g
```

Estructura General del Lenguaje:

En un archivo .olc solo puede ser declarado solamente un programa. El archivo .olc tiene la siguiente estructura general:

```
PROGRAM <id> ;
        [ importaciones o declaración variables o matrices globales en desorden ]
BEGIN
        [ sentencias del método main ]
END

[ lista de métodos y funciones ]
```

Visibilidad:

Para el lenguaje estructurado no existe visibilidad, puesto que todos los métodos y variables son públicas y no hay excepción.

Comentarios:

El lenguaje como Java permite dos opciones para comentar en el código. La sintaxis para realizar los comentarios es la siguiente:

Comentarios de una sola línea
// **<CUALQUIER_TEXTO menos salto de línea>** **<SALTO_LINEA>**
Comentarios de una sola línea
{ **<CUALQUIER_TEXTO>** **}**

Importaciones:

El lenguaje soporta la inclusión de métodos definidos en otros archivos .olc, en este sentido llamaremos "archivo origen" al archivo .olc que realiza la importación y "archivo importado" al archivo .olc que es llamado desde la importación.

Especificaciones sobre la importación de archivos**:**

- Cuando importamos un archivo solo podemos utilizar sus procedimientos (métodos o funciones).
- Cuando importamos un archivo NO podemos utilizar sus variables globales (pero un procedimiento importado sí puede hacer usos de variables globales de su origen).
- Cuando realizamos una importación y encontramos un método que es igual en nombre, tipo y parámetros que otro que ya ha sido definido en el archivo origen, ignoramos el método importado y utilizamos el método del archivo origen.
- Cuando importamos un archivo NO podemos utilizar su método main.
- Solo podemos importar archivos .olc que se encuentren dentro del mismo proyecto.
- Podemos importar la cantidad de archivos .olc que deseemos.

La sintaxis es la siguiente:

USES **<id>;**

Declaración de Variables Globales y Locales:

Una variable es un espacio de memoria reservado para almacenar un valor que corresponde a un tipo de dato. El lenguaje admite la declaración de variables globales y locales.

Especificaciones sobre las variables globales y locales:

- Las variables globales deben ser declaradas en el encabezado del programa para poder ser utilizadas.
- Una variable global puede ser accedida y asignada desde cualquier método, incluyendo el método main.
- Una variable local puede ser accedida y asignada únicamente desde el método en el que fue declarada, respetando el ámbito en la que fue declarada.
- No pueden existir variables globales con el mismo nombre.
- No pueden existir variables locales con el mismo nombre, pero sí puede existir una local con el mismo nombre que una variable global dándole mayor prioridad a la variable local.
- No se puede asignar valor a las variables al momento de declararlas.

La sintaxis es la siguiente:

```
VAR <TIPO> : <id> [ ,<id_1>, <id_2, …, <id_n> ] ;
```

Declaración de Matrices Globales y Locales:

Una matriz es una estructura de datos, o más técnicamente, un espacio de memoria que permite almacenar una colección de elementos. El lenguaje admite la declaración de matrices globales y locales.

Especificaciones sobre las matrices globales y locales:

- Una matriz siempre tendrá el mismo tipo de datos para todos sus elementos.
- Una matriz puede ser definida de una o varias dimensiones.
- Una matriz tiene un tamaño estático, es decir no varía su tamaño una vez este haya sido definido.
- Una matriz puede ser declarada como global o local, siguiendo las mismas restricciones que las variables.

La sintaxis es la siguiente:

```
VAR <id> OF <TIPO>  <tamDim1> [ < tamDim2 > < tamDim3 > … < tamDimN > ] ;
```

Nota: El valor de la las dimensiones de una matriz debe ser una expresión de valor entero, si el valor es un entero negativo o cero se deberá mostrar un error semántico.

Método Main:

Es el punto de entrada para que nuestra aplicación cobre vida, en este método se inicia la ejecución de todas las sentencias que definirán el comportamiento de la aplicación que se ha escrito. Un programa solo puede tener un método main, este método no sigue la misma sintaxis que el resto de métodos, no posee parámetros y está formado por todas las instrucciones que se encuentran encerradas entre el BEGIN y END del programa.

Procedimientos (Métodos y Funciones):

Un procedimiento puede ser un método él cual es una subrutina con instrucciones, una función es una subrutina que retorna un valor de acuerdo a su tipo de dato, un programa puede contener uno o más procedimientos. **Los métodos y funciones del lenguaje de alto nivel soportan llamadas recursivas.**

Para un método la sintaxis es la siguiente:

```
PROCEDURE <id> ( [ <TIPO> : <id_1>, <TIPO> : <id_2>, …, <TIPO> : <id_n> ] ) ;
BEGIN
        <sentencias del método>
END
```

Para una función la sintaxis es la siguiente:

```
PROCEDURE <TIPO> : <id> ( [ <TIPO> : <id_1>, <TIPO> : <id_2>, …, <TIPO> : <id_n> ] ) ;
BEGIN
        <sentencias de la función>
END
```

Nota: Dentro de un programa podrán existir más de procedimientos, pero deberán diferenciarse por el tipo de procedimiento así como por la cantidad o tipo de parámetros, si no es el caso reportar error semántico.

Llamada a Funciones:

Cuando queremos llamar a una función en una expresión basta con hacer referencia al nombre del mismo y pasar los parámetros si fuera necesario.

La sintaxis es la siguiente:

```
<id> ( [ <PARAMETRO1>, <PARAMETRO2>, …., <PARAMETRON> ] )
```

Acceso a una Posición de una Matriz:

Obtienen el valor de una posición de una matriz en una expresión.

La sintaxis es la siguiente:

```
<id> <posicion1> [ < posicion2 > … < posicionN > ];
```

Nota: El valor de la posición de una dimensión de una matriz debe ser de valor entero. Si se Intenta acceder a una posición con un dato entero negativo o mayor a los límites definidos en el tamaño de la matriz se deberá mostrar un error semántico.

Sentencia de Retorno:

La sentencia de retorno tendrá dos funciones diferentes:

- Dentro de los métodos servirá para alterar el flujo de ejecución del mismo, si dentro de un método se encuentra la sentencia de retorno se deberá terminar la ejecución del método, esta sentencia también puede venir en el método main.

- Dentro de las funciones, la sentencia de retorno terminar la ejecución de la función e indicará el valor que se debe retornar en el momento de la ejecución, por ello dentro de las funciones deberá venir acompañado de un parámetro que será una expresión.

Dentro de un método la sintaxis es la siguiente:

```
RETURN ;
```

Dentro de una función la sintaxis es la siguiente:

```
RETURN <EXPRESION_A_RETORNAR> ;
```

Sentencia para Ejecutar un Procedimientos:

Cuando queremos llamar a un procedimiento basta con hacer referencia al nombre del mismo y pasar los parámetros si fuera necesario.

La sintaxis es la siguiente:

```
<id> ( [ <PARAMETRO1>, <PARAMETRO2>, …., <PARAMETRON> ] ) ;
```

Sentencia de Asignación de Variables:

Una variable puede recibir cualquier valor a asignar que sea del mismo tipo del que fue declarada. Una variable global puede ser asignada desde cualquier método, incluyendo al método main. Una variable local sólo puede ser asignada desde el método y ámbito desde el que fue declarada.

La sintaxis es la siguiente:

```
<id> := <EXPRESION_A_ASIGNAR> ;
```

Sentencia de Asignación de una Posición de una Matriz:

Luego de ser declarada una matriz podemos asignarle un valor a una posición (valor entero) en específico. Una posición de una matriz global puede ser asignada desde cualquier método, incluyendo al método main. Una posición de una matriz local sólo puede ser asignada desde el método y ámbito desde el que fue declarada.

La sintaxis es la siguiente:

```
<id> <posicion1> [ < posicion2 > … < posicionN > ] := <EXPRESION_A_ASIGNAR>;
```

Nota: El valor de la posición de una dimensión de una matriz debe ser de valor entero. Si se Intenta acceder a una posición con un dato entero negativo o mayor a los límites definidos en el tamaño de la matriz se deberá mostrar un error semántico.

Sentencia de Asignación por Incremento y Decremento:

Aumento: Esta sentencia consiste en añadir una unidad a una variable o posición de una matriz de tipo numérico (INT, DOUBLE, CHAR). El operador del aumento es el doble signo más ++.

La sintaxis es la siguiente:

```
<id> | <id> <posicion1> [ < posicion2 > … < posicionN > ] ++ ;
```

Decremento: Esta sentencia consiste en quitar una unidad a una variable o posición de una matriz de tipo numérico (INT, DOUBLE, CHAR). El operador del decremento es el doble signo menos --.

La sintaxis es la siguiente:

```
<id> | <id> <posicion1> [ < posicion2 > … < posicionN > ] -- ;
```

Sentencias de Flujo de Control:

IF:

Sentencia cuya función es determinar el flujo que el programa debe seguir entre una acción u otra dependiendo de una o varias condiciones establecidas por el programador.

La sintaxis es la siguiente:

```
IF ( <CONDICION> ) THEN
BEGIN
        [ sentencias del IF ]
END
```

IF ELSE:

Se debe utilizar cuando se desea tener un set de instrucciones en el caso de que la condición sea verdadera y otro set cuando la condición sea falsa.

La sintaxis es la siguiente:

```
IF ( <CONDICION> ) THEN
BEGIN
        [ sentencias del IF ]
END ELSE
BEGIN
        [ sentencias del ELSE ]
END
```

IF ELSEIF:

Cuando se necesitan diferentes acciones para diferentes casos (condiciones), es posible utilizar esta sentencia, esta sentencia puede agregar un set de instrucciones que se ejecuten en el caso que no se cumpla ninguna condición.

La sintaxis es la siguiente:

```
IF ( <CONDICION> ) THEN
BEGIN
        [ sentencias del IF ]
END ELSEIF ( <CONDICION> ) THEN
BEGIN
        [ sentencias del ELSEIF1 ]
END [ ELSEIF ( <CONDICION> ) THEN
BEGIN
        [ sentencias del ELSEIF2 ]
END
....
ELSEIF ( <CONDICION> ) THEN
BEGIN
        [ sentencias del ELSEIF_N ]
END ] [ ELSE
BEGIN
        [ sentencias del ELSE ]
END ]
```

SELECT CASE OF:

Es una sentencia de anidamiento de múltiples instrucciones IF...ELSEIF sobre un caso en particular aplicado sobre una expresión, cuando la expresión evaluada es igual a un determinado valor de un caso **se debe ejecutar el set de instrucciones contenidas dentro del caso hasta encontrar una sentencia BREAK; que indicara la salida, de lo contrario la**

ejecución seguirá Interpretando las sentencias de los casos siguientes, en caso no se cumpla ninguna condición el CASE OF puede incluir opcionalmente un set de instrucciones que se ejecutarán que se ejecutarán por default (ELSE), estas se deben ejecutar hasta encontrar una sentencia de que indique su salida. **No se pueden comparar valores lógicos.**

La sintaxis es la siguiente:

```
SELECT CASE ( <EXPRESION> ) OF
BEGIN
        <VALOR1> : BEGIN
            [ sentencias del caso 1 ]
        END
        [ <VALOR2> : BEGIN
            [ sentencias del caso 1 ]
        END
        …
        <VALORN> : BEGIN
            [ sentencias del caso N ]
        END ]
        [ ELSE: BEGIN
            [ sentencias default ]
        END ]
END
```

Sentencias Cíclicas:

Sentencia BREAK:

Esta sentencia se utilizara para salir de cualquier bucle.

La sintaxis es la siguiente:

```
BREAK ;
```

Sentencia CONTINUE:

Esta sentencia se utilizará para indicarle al bucle que termine la secuencia en la que se encuentra y hace que se evalúe la condición para determinar si debe iniciar una nueva ejecución del ciclo. **Dentro de un FOR antes de evaluar la condición hace el incremento o decremento respectivamente a la variable de control**.

La sintaxis es la siguiente:

```
CONTINUE ;
```

WHILE:

Es un ciclo que se ejecuta mientras una condición se siga cumpliendo.

La sintaxis es la siguiente:

```
WHILE ( <CONDICION> ) DO
BEGIN
        [ sentencias del WHILE ]
END
```

DO WHILE:

Es un ciclo que se debe ejecutar al menos una vez, luego debe seguir la ejecución mientras la condición se siga cumpliendo.

La sintaxis es la siguiente:

```
DO BEGIN
        [ sentencias del DO WHILE ]
END WHILE ( <CONDICION> ) ;
```

REPEAT UNTIL:

Es un ciclo el cual debe ejecutarse al menos una vez y continuará su ejecución mientras la condición NO se cumpla.

La sintaxis es la siguiente:

```
REPEAT BEGIN
        [ sentencias del REPEAT UNTIL ]
END UNTIL ( <CONDICION> ) ;
```

FOR:

Es un bucle que permite inicializar o establecer una variable como variable de control e indicar un valor final, la variable de control que en cada iteración irá aumentando en una unidad si el valor inicial es menor al final o disminuyendo en una unidad si el valor inicial es menor al final y se ejecutará mientras el valor de la variable de control aun siga siendo menor o mayor respectivamente. La variable de control y el valor final deben ser valores enteros. La variable de control debe ser declarada previamente.

La sintaxis es la siguiente:

```
FOR <asignación inicial> TO <valr final> DO BEGIN
        [ sentencias del FOR ]
END
```

Funciones Propias del Lenguaje:

Estas funciones sólo pueden ser llamadas desde dentro de algún procedimiento.

GRAF:

Este método tomará parámetros el tipo de dato (si es una función si no este no se ingresa), nombre y los tipos de datos de los parámetros del procedimiento que se desea graficar, para poder hacer el diagrama de flujo de dicho procedimiento.

La sintaxis es la siguiente:

```
GRAF ( [ <TIPO> ] <id> [<TIPO1> <TIPO2> … <TIPO_N> ] ) ;
```

OUT:

Este método tomará solamente un único parámetro el cual una expresión la cual se desea mostrar en una nueva línea en la consola.

La sintaxis es la siguiente:

```
OUT ( <expresion a imprimir> ) ;
```

MANEJO Y RECUPERACION DE ERRORES:

Para generar los resultados las instrucciones ejecutada por el Intérprete de alto nivel, l**a aplicación debe estar en la capacidad de poder recuperarse de errores léxicos, sintácticos y semánticos** para corregir de forma eficiente el archivo.

Reporte de errores léxicos, sintácticos y semánticos:

No.	Descripción	Tipo de error	Línea	Columna
1	^ no pertenece la lenguaje	Léxico	12	45
2	Se esperaba … (se encontró id …)	Sintáctico	45	12
3	El tipo a asignar no es compatible	Semántico	53	10
4	La variable var1 no existe	Semántico	78	12
N	…	…	…	…

El reporte debe ser en formato HTML y poder ser visualizado al momento de terminar el análisis. **Considerar que en él enunciado se describió lo más importante de lo permitido o no en el lenguaje, es responsabilidad del alumno capturar la mayor cantidad de casos posibles de errores semánticos (errores lógicos, de control y excepciones).**

NOTAS IMPORTANTES:

- **Se debe realizar de manera individual.**
- Las herramienta utilizada para realizar análisis léxico y sintáctico serán Irony.
- La aplicación se desarrollara en el lenguaje de programación C# utilizando el IDE Visual Studio.NET.
- El lenguaje es case sensitiva, por lo cual distingue entre minúsculas y mayúsculas.
- La herramienta para generar el diagrama de flujo será únicamente Graphviz.
- La imagen generada debe contener todos los diagramas generados en la Interpretación del lenguaje en el cual se deben presentar los diagramas identificados de manera correcta y en orden.
- Deben utilizar el Árbol Sintáctico para la Interpretación y ejecución de todas las en caso contrario no se tendrá derecho a calificación.
- **Copias parciales o totales tendrán una nota de cero (0) puntos y se notificará a la escuela para que se apliquen las sanciones correspondientes.**
- **En el caso de no cumplir con alguna de las indicaciones antes mencionadas o con los entregables descritos a continuación NO se calificará; por lo cual, se tendrá una nota de cero puntos.**

ENTREGABLES:

Para el tener derecho se contara con los siguientes entregables para su calificación:

- Aplicación Funcional.
- Código fuente.
- Gramática realizada en Irony.

FECHA LÍMITE DE ENTREGA MARTES 2 DE ENERO DE 2018
ANTES DE LAS 11:59 PM