|  |  |
| --- | --- |
| **Instituto Tecnológico de Costa Rica**  **Sede Regional San Carlos** | Primera Tarea Programada  Compiladores e Intérpretes |
| Prof. Oscar Víquez Acuña. | Análisis Sintáctico  Compilador MiniC# |

**Introducción**

Un compilador es una herramienta de trabajo muy importante para cualquier estudiante de computación y normalmente dichas herramientas se utilizan sin conocer detalles de su implementación que podrían ser de ayuda para comprender mejor el funcionamiento de la herramienta. Se espera que después de este curso salga con una mejor comprensión de los compiladores que utiliza.

Debe notarse que la definición puede contener algunas ambigüedades que el alumno debe resolver como parte propiamente del trabajo.

**Cualquier detalle extra que no sea aclarado en esta documentación y que surja debido a las dudas de los grupos, será aclarado en su momento por el profesor sin necesidad de elaborar otro enunciado.**

**Definición**

El proyecto consiste en crear un Analizador Sintáctico para el lenguaje MiniC# cuya sintáxis se detalla posteriormente en documento adjunto. Este lenguaje es un subconjunto de lenguaje al estilo C# para dotNET y por ende tiene similitudes marcadas pero también diferencias significativas. El objetivo en esta etapa es realizar la fase de análisis sintáctico del compilador para dicho lenguaje en donde se espera concluir con las subfases de “scanning” “parsing” y AST. Para el desarrollo de esta etapa será necesario la implementación de la misma en el lenguaje C# así como el uso obligatorio de la herramienta/librería ANTLR4.

A continuación se detallan las características del lenguaje MiniC#.

**Convenciones léxicas**

Comentarios y caracteres ignorados

Los comentarios deben permitirse desde que inicie con “/\*” hasta que termine “\*/” y los de luna línea que comienzan con “//”. Deben además permitir comentarios anidados (unos dentro de otros).

Otros caracteres ignorados incluyen el cambio de línea, return-carry, tab, y el espacio en blanco.

**Tokens**

Los símbolos entre comillas de la gramática son símbolos terminales, cualquier otra secuencia de caracteres denota el nombre de una clase léxica, p.e *id.*

Asumiremos entonces las siguientes clases léxicas:

letter 🡪 [a-z A-Z \_]

digit 🡪 [0-9]

id 🡪 letter {letter | digit}

Observe que el “\_” está tratado como una letra por cuestiones propias del formato de los nombres de identificadores en muchos lenguajes de programación.

Constantes numéricas

El literal para números enteros es un número que no debe comenzar con 0 a menos de que sea el número 0

Debe permitir además el uso de punto flotante para aquellas constantes double.

Constantes carácter

Un *charconst* como una ‘ seguida de un carácter ASCII (puede ser ASCII extendido) y posteriormente otra ’ de cierre. Además las contantes char solamente deben permitir un carácter a excepción de caracteres especiales como el TAB o la comilla o cualquier otro de uso normal en C#.

Operadores

* Operadores relacionales (relops): ==, !=, <, <=, > y >=
* Los addops son + o –
* Los mulops son \*, /, %
* El lexema para el token assign es = y los operadores lógicos son || y &&

**Scanner**

El Scanner debe realizarse con ANTLR4 y debe incluir todos los tokens solicitados.

**Parsing**

La gramática que se adjunta con este documento se encuentra en formato EBNF podrá ser modificada por el estudiante si lo desea sin que se altere las capacidades sintácticas del lenguaje. A partir de esta gramática se deberá escribir el archivo de gramática .g4 para el ANTLR

En caso de error el parser debe reportarlo en la interfaz gráfica que se implemente como editor de texto para el compilador.

Los errores que se muestren deben ser lo suficientemente significativos y además deben contar con información de posicionamiento de dicho error (línea y columna).

**AST**

Se debe construir el Árbol de Sintaxis Abstracto (AST) utilizando la estructura de datos que genera ANTLR4.

**Manejo de Errores**

Se debe utilizar el lenguaje Español para los errores que se muestren.

**Detalles de interfaz de usuario**

El editor contará con las opciones que el programador crea convenientes e incorporará una única opción de compilación llamada “compilar”. Esta opción será única para el resto del desarrollo del programa. Debe contar además con el despliegue de la fila y la columna para el cuadro de texto principal, de manera que cuando se despliegue algún error con su respectiva ubicación, pueda ser sencillo encontrarlo.

El editor debe contemplar una opción para abrir más de un archivo fuente a la vez y que estos puedan organizarse en pestañas y pueden ser compilados de forma independiente.

Para el desarrollo del editor de texto pueden utilizar soluciones previamente creadas y disponibles en la Web pero deben documentar adecuadamente la fuente y los cambios realizados a la misma.

**Puntos de evaluación**

La evaluación de todo el proyecto se basa en la detección de errores y en la generación del código correctamente. Para esta primera tarea, la evaluación consiste en realizar el proceso de escaneo del texto fuente (Scanner), la verificación de los tokens con respecto a la gramática (Parser) y la posterior creación de la estructura intermedia AST.

Los puntos que serán evaluados para este primer proyecto son los siguientes:

|  |  |
| --- | --- |
| **Interfaz de Usuario** | **15** |
| Editor amigable. Opciones de compilación funcionales.  Identificación de código por línea y columna. | 10 |
| Manejo de archivos | 5 |
| **Scanner** | **30** |
| Tokens válidos, Comentarios y caracteres especiales | 20 |
| Detección de errores léxicos (Claridad y Ubicación) | 10 |
| **Parser** | **45** |
| Declaraciones (vars, const, métodos, clases) | 10 |
| Statements (if, while, method calls, etc) | 12 |
| Expresiones (simples, complejas) | 13 |
| Detección de errores sintácticos (Claridad y Ubicación) | 10 |
| **Documentación** | **10** |
| Formato y Contenido | 5 |
| Ortografía y Gramática | 5 |
|  | **100** |

**Documentación**

La documentación deberá incluir las siguientes partes y no debe exceder las 7 hojas por un lado sin incluir la portada:

* Portada formal.
* Análisis del lenguaje (análisis de la gramática, del código que puedo crear o no puedo crear con ella)
* Soluciones e implementación. (cuestiones técnica de la implementación del proyecto)
* Resultados obtenidos. (cuadro **explicativo** de lo realizado y lo no realizado – énfasis en esto último)
* Conclusiones **del trabajo**.
* Bibliografía. (documentos con autor)

**Aspectos Administrativos**

* La tarea se desarrollará en grupos de máximo dos personas.
* El lenguaje de programación será *C#* de la plataforma *dotNet Framework.*
* La fecha de entrega será el Viernes 31 de Marzo de 2023 hasta las 10:00 pm.
* El medio de entrega será el TEC-DIGITAL solamente.
* Cualquier intento de plagio, copias totales o parciales de otras personas o de Internet, serán castigados con nota de 0.

**Programa de** **Ejemplo Tentativo (deben complementarlo y crear más ejemplos)**

using otraClase;

class P

{

class Table {

int[] pos;

int[] neg;

}

Table val;

void Main ()

{

int x, i;

*//---------- Initialize val ----------*

val = new Table;

val.pos = new int[]; val.neg = new int[size];

i = (int)0.0;

while (i < size) {

add(val.pos,0); val.neg[i] = 0;

i++;

}

*/\*---------- Read /\*values\*/ ---------- \*/*

read(x);

while (-size < x && x < size) {

if (0 <= x) val.pos[x]++;

else val.neg[-x]++;

read(x);

write("Valor de x: "); write(x,2);

}

}

}