**Proyecto - Fase #3 - Tabla de símbolos y comprobación de tipos**

**Mini C# y Mini Java**

Fecha de entrega: sábado 21 de noviembre, 07:30 am

**El objetivo**

La tercera fase del proyecto consistirá en generar la tabla de símbolos, realizar la asignación de valores a variables y constantes, comprobar los tipos de ciertas expresiones para los lenguajes que están trabajando.

**Tabla de símbolos**

La tercera fase debe generar la tabla de símbolos para el lenguaje mini C# o mini Java. Como la tabla se va generando y eliminando en forma dinámica, se deberá ir manejando y generando un archivo de salida que contenga la tabla completa. La estructura de la tabla se deja a discreción del alumno, pero se calificará la presentación de la misma, así como la mejor manera de almacenar y representar los elementos en la tabla.

**Asignación de variables y constantes**

Durante esta fase también debe realizarse las operaciones y la asignación de valores de las constantes y variables en el lenguaje. De tal manera que, si tenemos una definición de constante de la siguiente forma:

const int a;

a = 35 \* 2;

El compilador debe almacenar la constante en la tabla de símbolos y el valor de la misma debe indicar 70. También se podrán hacer operaciones y referencias a valores de constantes y variables ya definidas, es decir:

double circun;

const double pi;

const double radio;

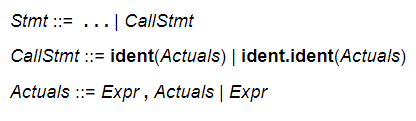
pi = 3.1416; /\*Se guarda la constante pi con valor 3.1416 \*/

radio = 20.5; /\* Se guarda la constante radio con valor 20.5 \*/

circun = pi\*2\*radio; /\*Se guarda la variable circun con valor 128.8056\*/

**Comprobación de tipos**

El compilador debe evaluar los tipos de las variables, constantes y parámetros de los procedimientos y funciones. Deberá indicar si las operaciones de expresiones, asignación y paso de parámetros está permitida en el lenguaje según los tipos de dato que se manejen. Para ello a su gramática debe agregarle las producciones para poder realizar invocaciones de funciones, las cuales se detallan a continuación:



Por ejemplo, se lee el siguiente archivo de prueba:

class Program {

int f1(int a, int b, int c){

if (a>b) c=2; else c=3;

}

void main(){

int m1,

int m2;

int m3;

Parser MyParser;

MyParser.f1(1,2,m3); /\* Paso de parámetros correcto \*/

MyParser.f1(1,“2.2”,m3); /\*Debe indicar error de argumento 2 inválido\*/

}

}

El resultado en pantalla debería ser, por ejemplo:

line 11, column 26: \*\*ERROR\*\* An argument for function f1 is invalid

Sin embargo, su compilador debe considerar la coerción de tipos: Por ejemplo:

int A;

double B;

A = 9;

B = 34.35 \* A;

Lo anterior es un ejemplo de coerción de tipos y es válido ya que debería poder convertir el valor entero a un doble. Pero si tenemos:

string A;

int B;

A = “hola”;

B = 34 \* A;

En este caso, el compilador debe indicar que no es válida la operación por tipos no compatibles:

line 2, column 9: \*\*ERROR\*\* Cannot operate type 'int' and 'string'

De encontrar errores, el compilador debe continuar hasta el final de archivo. Los errores deberá reportarlos en pantalla indicando:

Línea y columna, Error: <descripción del error>.

De no hallar errores en el archivo de entrada, indicarlo también en pantalla. Su compilador siempre deberá poder reconocer errores léxicos o sintácticos en los archivos de entrada.

**Requerimientos generales**

* El ejecutable debe ser llamado minic o minijava y debe pedir el archivo de entrada y escribir todos los resultados en pantalla.
* El proyecto se entrega en el mismo repositorio en GitHub de la fase anterior, sobre la rama master o principal. Por tanto, se deberá crear una nueva rama donde incorpore esta fase. Se calificará el proyecto hasta el último commit realizado antes de la hora de entrega. Se revisará el historial del repositorio. No se aceptarán cambios en el momento de la calificación.
* En el repositorio deberá incluir un archivo README.txt donde se detalle la estructura de su tabla de símbolos, cómo le da mantenimiento, así como detallar su estrategia de manejo de errores semánticos.

**¡Buena suerte!**