PreProyecto
Compiladores 2023

Integrantes:

- Romero Juan Ignacio,
- Rojo Jonathan Jair.

Etapas en el pre-proyecto:

1) **Analisis Lexico:** Acá definimos como reconocer ciertas cadenas, de un código c, asociando un token a cada una de ellas. Este token es un identificador único, que nos ayudará en el análisis semántico para asignarles un significado.

   Dado que los tokens pueden tomar diferentes tipos de valores, en cal-sintaxis.y, específicamente en el bloque de %unión, definimos que pueden tomar cadenas, enteros o estructura de árbol.

   Cuando alguna secuencia de cadenas de entradas son de tipo caracter,tuvimos que pedir memoria para poder luego al yy.val de tipo caracter,poder asignarlo al yy.text para luego que el token recibiera algo de tipo cadena.

2) **Analisis Sintactico:** Nos enfocamos en describir cómo debía escribirse nuestro programa utilizando una gramática, que fue extendida y utilizando expresiones regulares.

   Uso de un no terminal type dónde se busca darle un tipo a la palabra restringida “int” y “bool” para diferenciarlos de los tipos enteros y caracteres.

   Se agregó un no terminal prog1, para permites tener programas sin sentencias.

3) **Análisis Semántico:** En esta instancia tuvimos que garantizar el significado único de cada una de las reglas de la gramática, utilizando el chequeo de tipos que nos garantiza que las cosas que comparamos o asignamos no difieran nunca.

   Se chequeó que el tipo de las declaraciones coinciden así como el de las asignaciones.

   En las operaciones, no se diferenció entre la suma y el OR, ni entre la multiplicación y el AND, sino que aquí, se chequeaba los tipos y se daba el valor correspondiente según la operación.

   Si algunos de los tipos difieren directamente cortamos el programa.

4) **Tabla de Símbolos:** En esta etapa para cada declaración agregamos cada variable, siendo esta única e irrepetible, a la Tabla de Símbolos. Cuando era usada en una

sentencia, primero se chequea si la variable estaba en dicha tabla, sino el uso de ella era erróneo y se cortaba el programa.

Tuvimos que realizar dos funciones, una para la existencia de la variable, y si esta nos devolvía que existía, invocamos a la otra función donde nos devolvió dicha variable.

5) **Árbol Sintáctico Abstracto:** En esta etapa quizás otros grupos utilizaron el ASA para realizar el chequeo de tipos, nosotros lo realizamos sobre la gramática, por lo que el propósito final del ASA es para la siguiente etapa, la de construir un código intermedio para la generación de assembler.

   Para formar el árbol necesitábamos tener en cuenta de que los no terminales debían "subir" valores para ir construyendo el árbol, debido a esto, en los types tuvimos que asignarles la estructura del árbol, para que tomen esa forma.

   Implementamos una función para escribir un .dot que nos permita chequear si la forma del árbol es la correcta de acuerdo al programa ingresado.

6) **Código de tres direcciones:** Utilizamos el Árbol Sintactico de la etapa anterior para recorrerlo y poder agregar a la lista todas aquellas sentencias de código, de manera secuencial para poder ser ejecutadas en un orden específico. Tuvimos que darle a cada código un enumerado para saber que ejecución deberá realizar dependiendo de ese valor.

7) **Generación de Assembler:** Con la lista del código de tres direcciones, solo quedaba escribir en un archivo el código assembler según el enumerado que traía ese nodo de la lista 3AC.