PRÁCTICA DE
PROCESADORES DEL LENGUAJE II
Curso 2021 – 2022
Entrega de Junio

APELLIDOS Y NOMBRE: Carrillo Álvarez, Miguel

DNI: 53364711L

CENTRO ASOCIADO MATRICULADO: UNED Valencia

CENTRO ASOCIADO DE LA SESIÓN DE CONTROL: UNED Valencia

EMAIL DE CONTACTO: mcarrillo95@alumno.uned.es

TELÉFONO DE CONTACTO: 622886358

¿REALIZAS LA PARTE OPCIONAL? (SÍ o NO): NO

1. El analizador semántico y la comprobación de tipos.

1.1. Descripción del manejo de la Tabla se Símbolos y de la Tabla de Tipos.

En el archivo *parser.cup* he creado una tabla de símbolos y una tabla de tipos para cada ámbito. En la tabla de Símbolos guardo las variables, parámetros, constantes y funciones. Esto me ayuda para buscar con el nombre en la tabla el tipo de Símbolo que es y para comprobar la unicidad del símbolo, evitando guardar dos símbolos con el mismo nombre. He tratado los procedimientos como funciones, diferenciándolos en que estos primeros no tienen valor de retorno.

En la tabla de tipos guardo los tipos primitivos entero y booleano para los cuales he creado una clase para cada uno de ellos que hereda de *TypeSimple*. También guardo los tipo Array con los atributos de clase dimensión que guarda el tamaño, *rangoInicio* que guarda cual es el valor de la primera posición del array, *rangoFinal* que guarda su última posición y *tipoArray* que guarda si es un array de enteros o de booleanos. Con esta tabla compruebo que a la hora de asignar un valor coinciden los tipos o que en el paso por parámetros coincide el tipo del parámetro con el tipo esperado. En el caso de los vectores compruebo que la posición a la que intenta acceder está dentro del rango del vector.

2. Generación de código intermedio.

2.1. Descripción de la estructura utilizada.

Para el código intermedio he utilizado cuádruplas basándome en las transparencias del material de estudio de Javier Vélez Reyes. También he tenido que crear cuádruplas propias por ejemplo en los vectores, para posicionar el acceso a la memoria o en la sentencia *for* para gestionar la salida del bucle. He implementado la generación de código intermedio para todo el lenguaje a excepción de lo relativo a los subprogramas. El motivo ha sido que me ha faltado tiempo para implementarlo.

He gestionado los tipo booleano con *Value*, dándole el valor cero en caso de ser falso y uno en caso de ser cierto.

He añadido métodos de acceso para el código intermedio generado y la lista de temporales ampliando la clase *nonTerminal* añadiendo un *TemporalIF* con sus métodos *get* y *set*. Estos temporales me han servido para almacenar los valores o las direcciones de las variables.

He utilizado etiquetas para las sentencias de if o for y para comprobar expresiones como MENOR o IGUALDAD.

Para las cadenas de caracteres he creado una lista llamada *listaCadenas* para almacenar estas cadenas e introducirlas al final.

3. Generación de código final.

3.1. Descripción de hasta donde se ha llegado.

Para el código final he usado un entorno de ejecución estático, sin registro de activación, el motivo es que al no implementar la parte opcional me he decantado por este entorno. He asignado posiciones de memoria estática para las variables y los temporales. Para ello he cogido la posición de memoria más alta (65535) y he ido asignando posiciones de memoria en orden decreciente según el tamaño de la variable. Tamaño 1 para variables de tipo primitivo y para los *arrays* he tomado su dimensión para reservar el espacio de memoria necesario para acceder a todas sus posiciones. Para los temporales les he dado una posición de memoria para cada uno.

Inicializo las variables globales a cero y todo el código intermedio generado se lo paso al código final. En la clase ExecutionEnvironmentENS2001 he generado la traducción de las cuádruplas de código intermedio a código final.

Para recuperar las posiciones de memoria de las variables, en Variables de código intermedio guardo el objeto de *SymbolVariable* de la tabla de símbolos. Con él llamo al método *getDireccion()* y obtengo posición de memoria donde se guarda el valor de esta.

Al tener las posiciones de memoria asignadas a las variables o temporales para acceder a una dirección de memoria de una variable o temporal utilizo un direccionamiento directo a memoria.

4. Indicaciones especiales

He intentado realizar todas las operaciones en tiempo de ejecución, por ejemplo si tengo que guardar en la posición 3 de un vector que tiene como rango [3..5], me traigo la posición de memoria de la variable tipo vector. Le resto el rango inicio (3) a la posición 3 del vector y obtengo la posición 0 dentro del espacio de memoria reservado para la variable vector.

Esta decisión la he tomado, porque al principio intentaba guardar todos los valores dentro de la tabla de Símbolos, pero creo que es más efectivo calcularlo en el código final, es decir en tiempo de ejecución.

Por último, solo me queda hacer una valoración de la práctica. Me ha parecido todo un reto superarla y he necesitado más horas de las que tenía pensado dedicarle. Tanto es así que muy a mi pesar he tenido que dejar la parte opcional porque me estaba quedado sin tiempo para preparar otras asignaturas. Si bien me he quejado del trabajo que supone realizar la práctica, tengo que admitir que me ha encantado hacerla. No solo por la satisfacción de haberla terminado sino por haberla acabado con la sensación de haber aprovechado todo el tiempo invertido para aprender esta apasionante asignatura.

He utilizado como bibliográfica complementaria el libro:

Compiladores: teoría e implementación de Jacinto Ruiz Catalán