Luis Alejandro Urbina
Ma. Isabel Ortiz Naranjo

Código de 3 direcciones

Diseño del compilador

El compilador funciona utilizando ANTLR4 y Python3. Se parte de la gramática de YAPL definida en el archivo YAPL.g4, a partir de ella se construyen las clases de YAPLLexer.py, YAPLParser.py y YAPLVisitor.py. A partir de esta última, se construyen dos visitors que recorren el árbol de análisis sintáctico generado por YAPLParser añadiendo cada símbolo a la tabla de símbolos y a su vez verificando las reglas semánticas (definidas por el propio lenguaje e implementadas en el compilador) y de tipos (definidas más adelante en este documento). Luego, se recorre una vez más el árbol de análisis sintáctico pero esta vez se van almacenando objetos de tipo Quadrupĺe por cada expresión, estas Quadruples se guardan en un objeto de tipo QuadruplesStack, que es la representación del código intermedio asociado al archivo de entrada. El output de este programa es un string que contiene el código de 3 direcciones asociado a un programa de entrada con extensión .cl.

Tamaño de tipos de datos

Los tipos de datos se refieren a un extenso sistema utilizado para declarar variables o funciones de diferentes tipos. El tipo de una variable determina cuánto espacio ocupa en el almacenamiento y cómo se interpreta el patrón de bits almacenado.

Para este proyecto se consideraron únicamente los tipos de dato básicos: **Int, String y Bool.** A continuación se presenta una tabla con los tamaños de los diferentes tipos de datos (basado en la descripción de tamaños de datos para C y C++).

Tipo de dato	Tamaño en bytes
Int	4
String	1
Bool	1

Creación de Código de 3 direcciones

Para representar cada una de las instrucciones del código de 3 direcciones se utilizó un paradigma de programación orientada a objetos y 3 clases principales: **Quadruple,**

QuadruplesStack y CodeCreator. A continuación se presenta una descripción de cada una de las clases.

Clase: **Quadruple**

Descripción: Guarda el código de una instrucción de tres direcciones.

Atributos:

operator (str): operador asociado a la quadrupla
arg1 (str): primer argumento de la quadrupla
result (str): resultado de la operación de la quadrupla
arg2 (str): segundo argumento de la quadrupla

—

Clase: **QuadruplesStack**

Descripción: Guarda todas las quadruplas asociadas a la traducción de un programa

Atributos:

quadruples (Array): array que almacena objetos de tipo Quadruple
memAddr (str): dirección de memoria actual de la última quadrupla
type (str): tipo de la última dirección de memoria guardada
attr (str): atributos adicionales de una quadrupla

—-

Clase: **CodeCreator**

Descripción: Se encarga de la creación y el manejo de las variables temporales, condicionales y ciclos while

Atributos:

tempControl (int): guarda el número de temporal actual
ifControl (int): guarda el número de if actual
nextControl (str): guarda el número de next actual
whileControl (str): guarda el número de while actual

Definición del código de 3 direcciones

Palabras reservadas

- **DECLARE_{className}:** subrutina que declara una clase
- **{methodName}∈{className}:** subrutina que declara un método dentro de una clase
- **PROCEDURE_RETURN**: registro que guarda el valor de retorno de una función que ha sido invocada
- **OBJECT_{className}[offset]**: etiqueta para el atributo perteneciente a una clase
- **INNITIALIZE {className}:** crea una instancia de la clase dada
- **FUNCTION_{functionName}[offset]:** etiqueta para identificar al atributo/parámetro de una subrutina
- **INVOKE {subroutine}:** llama a una subrutina dentro del código de tres direcciones

¿Cómo funciona el código de tres direcciones?

Para operaciones aritméticas (+, -, *, /):

tx = val1 + val2, donde val1 y val2 pueden ser direcciones en memoria o escalares

Para variables (a: Int; a: Int <- 5;)

OBJECT_{className}[offset] = valor de la variable (o valor por defecto)

Estructuras de control (If)

Crear una subrutina:
[condición] goto [x]IF_TRUE
goto [x]IF_FALSE

[x]IF_TRUE:
ejecuta código del true
[x]NEXT

[x]IF_FALSE:
ejecuta código del false

[x]NEXT:

goto siguiente_instrucción fuera del if

Estructuras de control (While)

Crear subrutina [x]WHILE:
[condición] goto [x]WHILE_TRUE
goto [x]WHILE_FALSE

[x]WHILE_TRUE
ejecuta código del while

[x]WHILE_FALSE
ejecuta código después del while

Creación de funciones:

Crear subrutina con etiqueta **{methodName}∈{className}:**
Asigna parámetros de la función con la etiqueta **FUNCTION_{functionName}[offset]**
ejecuta el código de la función
PROCEDURE_RETURN = valor de retorno de la función

Llamadas de funciones

INVOKE **{subroutine}**
Dirección de destino = PROCEDURE_RETURN