Programación III

Teoría y Práctica de Compilación

ISC-314

Especificación de Proyecto Final

Módulo de Generación de Código Intermedio

Objetivos Generales

Este mini-proyecto parcial les permitirá a los alumnos:

1. Poner en práctica los conocimientos aprendidos hasta ahora en lo que se refiere a:
    a. Representación Intermedia del Código Fuente
    b. Representación de Código de Tres Direcciones
    c. Representación de Tétradas o Cuádruplas
    d. Acciones Semánticas Para La Generación de Código Intermedio
    e. Código Intermedio Para Representar
        i. Expresiones
        ii. Estatuto de Asignación
        iii. Estatuto de Condicional
        iv. Estatuto de Ciclos

2. Modificar el Módulo de Tabla de Símbolos
    a. Integrar Segmento de Constantes Enteras
    b. Integrar Segmento de Constantes de Coma Flotante
    c. Integrar Segmento de Constantes de Cadena
3. Codificar el módulo de Generación de Código Intermedio

Conceptos Preliminares

Tal cual como se explicó en clase, nuestro compilador generará código intermedio en formato de *cuádruplas* o *tétradas*. A continuación, se da una especificación de los operadores del lenguaje intermedio usado para este proyecto:

Tabla 1 - Operadores del Código Intermedio

Operador	Ejemplo	Descripción[1]
BUNC	`BUNC 0 0 dest`	Salto no condicionado a la tétrada numerada con dest
CALL	`CALL n 0 dest`	Salto no condicionado a la tétrada numerada con dest llamando al procedimiento cuyo ID se haya en la posición "n" de la tabla de símbolos
PLUS, SUB, MULT, DIV	`OP n m dest`	Aplicación de la operación aritmética OP a los identificadores, constantes y/o temporales que se hallan en las posiciones "n" y "m" de la tabla de símbolos y acumular el resultado en "dest".
BEQ, BLT, BGT, BGE, BLE, BNE	`OP n m dest`	Aplicación de la operación de comparación OP a los identificadores y/o constantes que se halla en las posiciones "n" y "m" de la tabla de símbolos
ASSGN	`ASSGN n 0 dest`	Asignar el valor de la constante, identificador y/o temporal que se halla en la posición "n" en el identificador y/o temporal denotado por "dest".
READ	`READ 0 0 dest`	Leer un valor del flujo de entrada estándar (consola) y asignarlo en el identificador que se encuentra en la posición "dest" de la tabla de símbolos.
WRITE	`WRITE n 0 0`	Escribir en el flujo de salida estándar (consola) el valor de a constante o identificador denotado por la posición "n" de la tabla de símbolos

A continuación, se da una sugerencia para el formato de registro que debería ser usado para representar una cuádrupla o tétrada:

Tabla 2 - Posible formato de registro para representar una cuádrupla

Nombre	Tipo
`lineNumber`	`Int`
`OperatorCode`	`OpEnum`[2]
`fstOperand`	`Short`
`sndOperand`	`Short`
`trdOperand`	`Short`

[1] Esta descripción se da por motivos ilustrativos. Recuerden que el código intermedio no necesariamente es ejecutable.

[2] Sería bueno que uds. acumulasen las constantes nombradas que representan operadores en una una enumeración *(enum*).

Instrucciones de Programación

1. Lean detenidamente el capítulo 8 de (Louden 2005)

2. Diseñe e implemente el módulo de generación de código intermedio. El mismo deberá de, al menos, ofrecer las siguientes primitivas[3]:
    a. Inicialización del módulo
    b. Construir una tétrada y guardarla en un arreglo de tétradas
    c. Actualizar una tétrada ya generada

3. Revise la gramática de MiniP usando la especificación de Yacc e identifique el fragmento ejecutable del lenguaje, es decir, los estatutos (*statements*) y las expresiones.
    a. Es posible que tengan que hacer ciertos cambios (p. ej. fragmentar reglas) para poder generar código intermedio para los estatutos condicionales y de ciclos.

4. Para cada tipo de estatuto (expresión) identifique las acciones semánticas necesarias para generar las tétradas correspondientes. Presten atención a los siguientes casos:

    a. Generación de código para un estatuto "if-then-else"
    b. Generación de código para un estatuto "for"

    En estos casos deberán de actualizar tétradas ya generadas para determinar el destino de los saltos condicionales/incondicionales generados. Se les sugiere que lean detenidamente la sección 8.4 del capítulo 8 pp. 428-436 de (Louden 2005)

5. Al final de la compilación el programa deberá reportar:
    a. Las acciones del analizador sintáctico
    b. El contenido de la tabla de símbolos
        i. Segmento de identificadores
        ii. Segmento de constantes enteras
        iii. Segmento de constantes de coma flotante
        iv. Segmento de constantes de cadena
    c. El listado de tétradas de código intermedio generadas

    En el archivo "Ejemplo_Salida.txt" se provee un ejemplo de la salida esperada

6. Asegúrense de que el proyecto compile satisfactoriamente y que no haya ningún error de ligadura (*error de linker*).

[3] Es muy posible que necesiten funciones de utilería que el implementen el código para tareas intermedias (p. ej. obtener la representación de cadena del código numérico de un operador, desplegar el arreglo de tétradas)

Parte Extra

Se asignarán puntos adicionales a aquellos que puedan adaptar el compilador para que éste genere código intermedio para una sentencia/estatuto "do-until". Para propósitos ilustrativos, más abajo, se sugiere la regla gramatical para una sentencia de este tipo:

```
STMT --> DO STMT UNTIL EXPR
```

Donde STMT y EXPR representan no-terminales para estatutos/sentencias (ejecutables) y expresiones (booleanas), respectivamente. DO y UNTIL representan **nuevas unidades léxicas** (tokens) que este tipo de sentencia/estatuto introduce.

Criterios de Prueba

Deberán probar la nueva versión del proyecto para los siguientes casos de prueba:

Nombre	**Descripción**
TestSBT1.txt	Este caso de prueba presenta varias instancias de errores de variables no declaradas y errores de tipo
TestSBT2.txt	Este caso de prueba presenta varias instancias de errores de definición-uso
TestSBT3.txt	Este caso de prueba no exhibe errores de definición-uso ni de tipos

Nota: Deberán corregir los errores de definición-uso y errores de tipos en los casos de prueba. Sin esto (suponiendo que su compilador esté bien implementado) no podrán generar código intermedio.

Estructura del Reporte de Experiencia

Con el fin de evidenciar que los estudiantes hayan tenido una experiencia de aprendizaje al menos satisfactoria, la entrega de este mini-proyecto exige un reporte de experiencia. El mismo deberá contar con las siguientes partes

- Introducción y apreciaciones previas a resolver el problema
- Descripción del proceso de resolución de la problemática
    - Interfaz del módulo de generación de código intermedio
    - Detalles de la implementación
- Lecciones aprendidas

Entrega

La evaluación de este proyecto requerirá que se entreguen (vía la PVA y en la fecha indicada) los siguientes entregables

- La especificación sintáctica en Yacc/Bison modificada
- El código fuente del módulo de tabla de símbolos (modificado)
- El código fuente del módulo de generación de código intermedio
- El proyecto completo compilado
- Salidas para cada caso de prueba
- Reporte/informe de experiencia
- Presentación de los puntos más relevantes del informe de experiencia

Referencias

Louden, Kenneth (2005). *Construcción de Compiladores: Principios y Práctica.* México, D.F.: Thomson Learning