INSTITUTO TECNOLÓGICO DE COSTA RICA

INGENIERÍA EN COMPUTACIÓN

SEDE DE ALAJUELA

COMPILADORES E INTERPRETES

PROFESOR: FABIAN FALLAS MOYA

ESTUDIANTES: YORLEY AGUILAR MENDOZA
FERNANDA ALVARADO VARGAS

Contenido

1. Introducción.

El proyecto a desarrollar pretende describir un lenguaje de programación con la intención de desarrollar un compilador, "un compilador es un programa encargado de traducir un lenguaje de programación a otro". Con lo anterior mencionado, es importante recalcar que el proyecto se desarrolla con el fin de poner en práctica el conocimiento que los estudiantes han obtenido hasta el momento.

El objetivo principal de este proyecto es que los estudiantes se introduzcan en el diseño e implementación de lo compiladores; para ello se planea crear un compilador para el lenguaje llamado C-0.

En dicho proyecto se realizarán tres check points; para el último check point se pretende abarcar los siguientes puntos:

- Análisis léxico.
- Análisis sintáctico
- Análisis semánticos.
- Generación de código.

Además en el último check point cada grupo debe realizar una adaptación al lenguaje a desarrollar; en este caso, se han agregado la implementación de métodos propios y switch.

2. Objetivos

Objetivo general.

-Desarrollar un compilador completo para el lenguaje C-0 con, implementando diferentes herramientas para su desarrollo.

Objetivos específicos.

➢ Implementar herramientas para la creación de un compilador. ➢ Conocer el funcionamiento de un compilador.

3. Herramientas a utilizar.

Para la creación de las primeras dos fases del un compilador se utilizarán las siguientes herramientas:

- Flex.
- Cup.
- IDE de NetBeans.

Además el programa se desarrollará utilizando java como lenguaje de programación. Para verificar que el código generado este funcionando, se utilizará el emulador ENS 2001 para windows.

4. Descripción del programa.

C-0 es un lenguaje de programación que está formado por los siguiente tokens:

- Constantes.
- Operadores.
- Delimitadores.
- Identificadores.
- Palabras reservadas.
- Tipos de datos.

A continuación se muestra una breve descripción de cada token.

4.1 Contantes: En C-0 se utilizará solo un tipo de constantes, las constantes de cadena. Una constante de cadena es cualquier secuencia de caracteres tipo ASCII entre comillas dobles. El carácter «\» sirve para destacar caracteres especiales como puede ser \n que representa un salto de línea.

4.2 Operadores: En esta sección se tienen tres operadores diferentes:

4.2.1 Operadores aritméticos: Implementados para realizar operaciones aritméticas; entre ellos se tienen: +(suma), -(resta), * (producto), / (división)

4.2.2 Operadores relacionales: Este tipo de operadores serán utilizados para expresiones donde se necesite realizar una comparación; entre ellos se encuentran: < (menor), > (mayor), ==(igual), ¡= (distinto).

4.2.3 Operadores lógicos: Son implementados para realizar comparaciones lógicas; estos operadores son los siguientes: || (or) && (and).

4.3 Delimitadores: Son que indican el inicio o fin del programa o de una expresión; entre ellos se encuentran: "()" , ";" , "{}".

4.4 Identificadores: Los identificadores serán las variables a utilizar en el lenguaje; es decir son las variables que el desarrollador podrá declarar.

4.5 Palabras reservadas: Las palabras reservadas son aquellas que son propias del lenguaje; entre ellas se encuentran: main, if, while , else, putw, puts, int, break, switch, case.

4.6 Tipos de datos: En C-0 se utilizará un tipo de dato y ese será int.

- Otra de las características a considerar en dicho lenguaje son las sentencias de control de flujo, las instrucciones de entrada-salida y la declaración de variables. Sentencias de control de flujo:

En esta sección se realiza una pequeña descripción del funcionamiento de las sentencias de control de flujo; para ello es necesario utilizar las palabras reservadas if, else, while, switch seguidas de un delimitador en cuyo caso son las llaves. En dicha estructura se debe escribir una expresión de comparación que de como resultado un true o false; esta expresión debe de estar entre paréntesis. Dentro de la estructura de control se puede definir un subprograma.

- Instrucciones de entrada y salida.

  Entre las instrucciones de entrada y salida se tienen: puts y putw.

  - puts: Salida de cadenas de caracteres.
  - putw: Salida de expresiones enteras.

- Declaración de variables.
  Las declaraciones de variables se pueden realizar al inicio del programa; para ello debe escribir el tipo de dato seguido del nombre de la variable, a estas se les puede asignar un valor o no.

5. Partes de un compilador

Anteriormente se mencionó que en el último check point deben estar concluidas las etapas del compilador; a continuación se dará una breve descripción de cada etapa.

5.1 Análisis léxico.

Durante el desarrollo de esta etapa se utilizó la herramienta lex; esta es una herramienta en librería en java que permite declarar todos los tokens que se van a implementar en el lenguaje; se debe crear un archivo .lex con la definición de los tokens a implementar; en el momento de ejecutar el programa se genera un .java con el código que verifica los tokens, además deben verificar la prioridad; en caso de que se ingrese un símbolo que no esté declarado en esta sección se debe mostrar un error léxico.

5.2 Análisis sintáctico.

En el desarrollo de esta etapa se utilizó la herramienta .cup; en esta herramienta se declaran todas las expresiones regulares; es importante recalcar que en esta etapa se verifica que las instrucciones estén escritas correctamente; al igual que el .lex, el . cup genera un .java con todas las funciones para verificar la sintaxis del programa; este es generado según las expresiones declaradas en el .cup. En el caso de que exista algún error sintáctico se debe reportar el error.

5.3 Análisis semántico.

Para realizar el análisis semántico también se utiliza el .cup; en esta etapa lo que se hace es que verifica que la expresión declarada tenga lógica; es decir que la instrucción realmente realice una instrucción válida; para ello se declaran una serie de registros y se van guardando en una pila desde el .cup, luego en una clase diferente se realizan los pop() de la pila y se verifica que la instrucción sea correcta; al igual que en las etapas anteriores si existe algún error este lo reporta. Esta etapa es esencial para poder generar el código del programa.

5.4 Generación de código.

Una vez verificada la semántica del programa, se debe generar el código asm para que este funcione; hay dos formas de generar el código.

- Crear una tabla intermedia.
- Escribir el código inmediatamente después del análisis semántico.

En el caso del lenguaje desarrollado por nuestro grupo, se hace inmediatamente después del análisis semántico. Lo que hace esto es escribir el código asm en un archivo de texto.

En esta parte es importante aclarar que para la ejecución del código se utiliza ENS 2001; por lo tanto se han utilizado las instrucciones válidas para dicho emulador y se escribe en un archivo .ens.

6. Conclusión

Cada una de las etapas desarrolladas tuvo sus complicaciones principalmente por la curva de aprendizaje que representaba cada herramienta implementada. Además se debe tomar tiempo para analizar cada instrucción que se va a desarrollar para que el compilador funcione correctamente, ya que en muchos casos o en todos, se debe hacer el análisis en papel para evitar una complicación a la hora de programarlo. Además realizar esta práctica permite que se entienda lo que se está haciendo.

En cuanto al grupo de trabajo, no hubo ninguna complicación en cuanto a la organización, además ha logrado administrar el tiempo para cada check point.

7. Anexos.

A continuación se adjuntará la definición de cada estructura al igual que el nombre:

7.1 Expresiones regulares.

```
EXP_ALPHA = [A-Za-z]

DIGIT = [0-9]

SPACE = " "

JUMP = \n|\r|\r\n

WHITESPACE = {JUMP} | [ \t\f] | {SPACE}

EXP_ALPHANUMERIC = {EXP_ALPHA}|{DIGIT}

SIGN = ("+"|"-")

INTEGER = ([1-9]{DIGIT}*)(("u"|"l"|"ul")?) | "0"

IDENTIFIER = {EXP_ALPHA}(_*)({EXP_ALPHANUMERIC}|(_))*

CHARACTER = \'{EXP_ALPHANUMERIC}?\'

DELIMITERS = "(" | ")" | "{" | "}"

ARITHMETIC = "+" | "-" | "*" | "/"

RELATIONAL = "<" | ">" | "==" | "¡="

LOGIC = "||" | "&&"

ASSIGN= "="

CONSTANT=  \"[^\n\r\"]*\" | \'[^\n\r\']*\'

KEYWORD = "main" | "else" | "puts" | "int" | "break"

TYPE_INT = "int"

TYPE_FLOW_CONTROL = "while" | "if" | "else" | "then"

TYPE_NUM = "int"

INVALID_CHARACTERS     =

"á"|"é"|"í"|"ó"|"ú"|"Á"|"É"|"Í"|"Ó"|"Ú"|"ñ"|"Ñ"|"¿"

LITERAL_NUM = {INTEGER}

LITERAL_CONSTANT = {CONSTANT}

ERROR = {DIGIT}({EXP_ALPHA}+) | {CONSTANT_ERROR}|

{INVALID_CHAR_ERROR}

CONSTANT_ERROR = \"[^\n\r\"]*{JUMP} | \'[^\n\r\']*{JUMP}

INVALID_CHAR_ERROR = {INVALID_CHARACTERS}
```