Manual de Usuario del Compilador

Benjamín Miranda
Vicente Ruiz
Universidad Católica del Norte
Ingeniería Civil en Computación e Informática
Fundamentos de la Computación

June 17, 2025

Contents

1 Introducción

Este documento describe el uso y diseño de un compilador desarrollado en C utilizando Flex y Bison. El compilador traduce un lenguaje propio a código C ejecutable, permitiendo así comprender el proceso completo de análisis léxico, sintáctico, semántico y generación de código.

2 Tipos de Datos y Sintaxis

2.1 Declaración de Variables

Permite definir el tipo y nombre de una variable antes de su uso en el programa. Toda variable debe ser declarada previamente indicando si es un número entero (`integer`), decimal (`floating`) o cadena de texto (`chain`).

```
integer edad;
floating altura;
chain nombre;
```

2.2 Operadores y Funcionalidades

Las expresiones pueden usar operadores para realizar cálculos o concatenaciones:

- **Aritméticos**: `+`, `-`, `*`, `/`, `%` – Para enteros y flotantes.
- **Concatenación**: `+` – Usado con cadenas de texto.
- **Comparación**: `==`, `!=`, `<`, `<=`, `>`, `>=`

2.3 Asignación y Concatenación

Permite almacenar un valor en una variable previamente declarada. Las variables pueden recibir valores literales, resultados de operaciones o concatenación de cadenas utilizando el operador `+`.

```
edad = 21;
altura = 1.75;
nombre = "Juan";
nombre = nombre + " P rez";
```

2.4 Entrada y Salida

La instrucción `write` permite capturar valores ingresados desde el teclado. Detecta automáticamente el tipo de la variable. La instrucción `print` muestra información en pantalla.

Nota: No se permite ingresar texto en variables numéricas. Actualmente, el sistema no controla errores como división por cero o entradas de tipo incorrecto.

```
integer edad;
chain nombre;

print "Ingrese su edad:";
write edad;

print "Ingrese su nombre:";
write nombre;

print edad;
print nombre;
```

2.5 Condicionales

Permiten ejecutar bloques de código dependiendo del resultado de una condición lógica.

```
if (edad > 18) {
    print "Adulto";
} else {
    print "Menor de edad";
}
```

2.6 Ciclo While

Ejecuta un bloque de código repetidamente mientras una condición se mantenga verdadera.

```
while (edad < 30) {
    edad = edad + 1;
}
```

2.7 Ciclo For

Permite ejecutar un bloque de código un número conocido de veces, con inicialización, condición y actualización.

```
for (x = 0; x < 5; x = x + 1) {
    print x;
}
```

3 Funciones

3.1 Definición

Las funciones permiten agrupar un conjunto de instrucciones bajo un identificador. Soportan parámetros y retorno de valores. El tipo de retorno y los tipos de parámetros deben coincidir

con las operaciones internas.

```
function sumar(integer a, integer b) {
    integer resultado;
    resultado = a + b;
    return resultado;
}
```

3.2 Llamado

Las funciones pueden ser llamadas directamente como una instrucción o dentro de una expresión.

```
integer x;
x = sumar(10, 5); // llamado como expresi n
sumar(2, 3);      // llamado como instrucci n
```

4 Estructura Interna del Compilador

- **Análisis léxico (Flex)**: convierte el texto fuente en una secuencia de tokens.
- **Análisis sintáctico (Bison)**: construye el árbol de sintaxis abstracta (AST) a partir de reglas gramaticales.
- **Análisis semántico**: verifica tipos, declaraciones y operaciones válidas entre variables.
- **AST y Tabla de símbolos**: estructura jerárquica del programa y registro de variables y funciones.
- **Generación de código (C)**: traduce el AST en un programa en C listo para compilar.

5 Errores Semánticos Detectados

- Uso de variables no declaradas
- División por cero no controlada (puede provocar fallo en ejecución)

6 Compilación y Ejecución

6.1 Pasos para compilar el compilador

```
bison -d parser.y
flex scanner.l
gcc -o dpp_compiler parser.tab.c lex.yy.c ast_c.c generarCodigo.c -
   lm
```

6.2 Ejecutar el compilador(En CMD)

```
dpp_compiler < test.dpp
gcc output.c -o programa
programa
```

7 Ejemplo Completo

Código de entrada

```
function potencia(integer base, integer exponente) {
    integer resultado;
    integer i;
    resultado = 1;
    i = 0;
    while (i < exponente) {
        resultado = resultado * base;
        i = i + 1;
    }
    print "Resultado de potencia:";
    print resultado;
    return resultado;
}

integer x;
integer y;

print "Ingrese base:";
write x;

print "Ingrese exponente:";
write y;

potencia(x, y);
```

Salida esperada

```
Ingrese base:
3
Ingrese exponente:
4
Resultado de potencia:
81
```