Instituto Tecnológico de Buenos Aires

Trabajo Práctico Especial:

Primera Entrega

Autoría:
Manuel Ader #63587 – mader@itba.edu.ar

Materia:
(2024-2Q) 72.39 - Autómatas,
Teoría de Lenguajes y Compiladores

Profesores:
Arias Roig, Ana Maria
Golmar, Mario Agustin

Fecha de entrega: 5/09/2024

Índice

Dominio

Desarrollar un GPL (General Purpose Language) imperativo Turing completo con el objetivo de proporcionar una herramienta de propósito general que permita la computación de cualquier tipo de programas. El lenguaje está diseñado para tener las funcionalidades básicas de un lenguaje de bajo nivel, con el fin de traducirse posteriormente a instrucciones equivalentes en C. Esto permite a los distintos usuarios desarrollar programas escalables de complejidad arbitraria, ya sea para resolver problemas generales de programación, cálculos matemáticos, manipulación de datos, o lógica condicional.

Este lenguaje puede ser usado para desarrollar programas que realicen desde simples operaciones aritméticas y lógicas, hasta implementar ciclos y funciones complejas. La idea es que este lenguaje sea capaz de soportar la construcción de cualquier programa computacionalmente posible, debido a su naturaleza Turing completa.

Construcciones

El lenguaje ofrece las siguientes construcciones y funcionalidades básicas:

1. **Declaración de Tipos de Datos y Variables:**

El lenguaje soporta los tipos de datos enteros (`INT`), booleanos (`BOOL`), y cadenas de texto (`STRING`).

Se pueden declarar variables de cualquiera de estos tipos usando la siguiente sintaxis: `type var_name;`

Por ejemplo:

```
INT contador;

BOOL esVerdad;

STRING nombre;
```

2. Asignación de Valores:

Las variables pueden ser asignadas utilizando la siguiente regla:

```
var_name = expression;
```

Por ejemplo:

```
contador = 5;

esVerdad = TRUE;

nombre = "Manu";
```

3. Expresiones:

El lenguaje soporta tres tipos de expresiones: aritméticas, booleanas y de cadenas.

- **Expresiones aritméticas**: permiten las operaciones básicas (`+`, `-`, `*`, `/`).
- **Expresiones booleanas**: permiten operadores lógicos (`AND`, `OR`, `NOT`) y comparaciones (`>`, `<`, `==`, `!=`, `>=`, `<=`).
- **Expresiones de cadenas**: son literales de texto encerrados entre comillas dobles.

4. Impresión de Datos:

Se puede imprimir una expresión utilizando la función `PRINT`:

```
PRINT(expression);
```

Por ejemplo:

```
PRINT(contador);

PRINT("Hola, Mundo!");
```

5. Funciones:

El lenguaje permite la definición de funciones con tipos de retorno y parámetros. Las funciones tienen la siguiente estructura:

```
type function_name(parameters) {

    instructions

}
```

Ejemplo de función:

```
INT sumar(INT a, INT b) {

     RETURN a + b;

}
```

Las funciones pueden ser invocadas mediante llamadas simples que pasan los argumentos correspondientes: `function_name(arguments);`

Por ejemplo:

```
sumar(5, 10);
```

6. Condicionales:

Las estructuras condicionales permiten realizar decisiones basadas en expresiones booleanas, con las construcciones:

```
IF (bool_exp) {

  instructions

} ELSE {

  instructions

}
```

7. Bucles:

El lenguaje incluye una estructura de bucle `FOR`, que itera en un rango de valores:

```
FOR var_name IN (arit_exp, arit_exp) {

  instructions

}
```

Por ejemplo:

```
FOR i IN (1, 10) {

       PRINT(i);

}
```

8. **Operadores Relacionales y Lógicos:**

Se proveen operadores relacionales como `>`, `<`, `==`, `!=`, `>=`, `<=` y lógicos como `AND`, `OR` y `NOT`, que pueden ser utilizados en expresiones booleanas.

Casos de Prueba

A continuación se presentan los casos de prueba para validar el comportamiento esperado del lenguaje:

Casos de Aceptación

1. **Declaración y uso de variables:**

Programa que declara una variable, asigna un valor y la imprime.

```
INT contador;

contador = 10;

PRINT(contador);
```

2. **Operaciones Aritméticas:**

Programa que realiza una operación aritmética y muestra el resultado.

```
INT a;
INT b;
a = 5;
b = 3;
PRINT(a + b);
```

3. **Uso de una función:**

Programa que define una función, la llama y muestra el resultado.

```
INT sumar(INT a, INT b) {
  RETURN a + b;
}
PRINT(sumar(2, 3));
```

4. **Bucle:**

Programa que imprime números del 1 al 10.

```
FOR i IN (1, 10) {
  PRINT(i);
}
```

5. **Condicional:**

Programa que decide si un número es mayor que otro.

```
INT a;
```

```
INT b;

a = 5;

b = 10;

IF (a > b) {

  PRINT("A es mayor");

} ELSE {

  PRINT("B es mayor o igual");

}
```

Casos de Rechazo

1. **Programa malformado:**

Faltan los puntos y comas al final de las declaraciones:

```
INT contador

contador = 10

PRINT(contador)
```

2. **Incompatibilidad de tipos:**

Intento de asignar una cadena de texto a una variable entera.

```
INT numero;

numero = "texto";
```

Gramática

G = <NT, T, S, P>

No-Terminales:

NT = {S, block, instructions, instruction, declaration, assignation, expression, print, function_call, function, conditional, loop, type, var_name, arit_exp, bool_exp, string_exp, compare_op, literal_int, literal_bool, literal_string, function_name, parameters, parameter, arguments, argument, return_statement}

Terminales:

T = {PROGRAM, INT, BOOL, STRING, AND, OR, NOT, PRINT, IF, ELSE, FOR, DO, TRUE, FALSE, RETURN} U C

C = El conjunto de caracteres disponibles en el alfabeto ASCII (letras, operaciones, comillas, comillas, etc)

Producción:

P = { S → PROGRAM block

block → { instructions }

instructions → instruction
| instruction instructions

instruction → declaration ;
| assignation ;
| expression ;
| print ;
| function_call ;
| return_statement ;
| function
| conditional
| loop

declaration → type var_name

```
type → INT
     | BOOL
     | STRING

assignation → var_name = expression

expression → arit_exp
           | bool_exp
           | string_exp
           | var_name

arit_exp → arit_exp + arit_exp
         | arit_exp - arit_exp
         | arit_exp * arit_exp
         | arit_exp / arit_exp
         | var_name
         | literal_int

bool_exp → bool_exp AND bool_exp
         | bool_exp OR bool_exp
         | NOT bool_exp
         | arit_exp compare_op arit_exp
         | var_name
         | literal_bool

string_exp → literal_string

print → PRINT (expression)

function → type function_name ( parameters ) block

parameters → parameter
           | parameter, parameters

parameter → type param_name                    .

function_call → function_name ( arguments )
```

```
arguments → argument
          | argument, arguments

argument → expression

conditional → IF (bool_exp) block ELSE block
            | IF (bool_exp) block

loop → FOR var_name IN (arit_exp, arit_exp) block

var_name → [a-zA-Z_][a-zA-Z0-9_]*

function_name → [a-zA-Z_][a-zA-Z0-9_]*

compare_op → >
           | <
           | ==
           | !=
           | >=
           | <=

return_statement → RETURN expression

literal → literal_int
        | literal_bool
        | literal_string

literal_int → [0-9]+

literal_bool → TRUE | FALSE

literal-string → \" " [^\"]* " \"

}
```