TEC | Tecnológico de Costa Rica

Tabla de contenidos

Tabla de contenidos **2**

Descripción del problema **2**

Diseño del programa **2**

Lista de Terminales 2

Palabras reservadas 2

Operadores aritméticos 3

Operadores relacionales 3

Operadores lógicos 3

Delimitadores y otros símbolos 3

Elementos léxicos 4

Lista de No Terminales 4

Símbolo Inicial 6

Análisis de resultados **6**

Objetivos 6

Lecciones Aprendidas 6

Bitácora **7**

Descripción del problema

Un grupo de desarrolladores desea crear un nuevo lenguaje imperativo, ligero, que le permita realizar operaciones básicas para la configuración de chips, ya que esta es una industria que sigue creciendo constantemente, y cada vez estos chips necesitan ser configurados por lenguajes más ligeros y potentes. Es por esto que este grupo de desarrolladores requiere desarrollar su propio lenguaje para el desarrollo de sistemas empotrados, y como primer paso necesitan desarrollar una gramática simple y poderosa.

Diseño del programa

Lista de Terminales

Palabras reservadas

```
if, elif, else, do, while, for, break, return, int, float, bool,
char, string, void, read, write, main, true, false
```

Operadores aritméticos

+	suma
-	resta
*	multiplicación
//	división
~	módulo
**	potencia
++	incremento
--	decremento

Operadores relacionales

<	menor que
<=	menor o igual que
>	mayor que
>=	mayor o igual que
==	igual a
!=	diferente de

Operadores lógicos

^	conjunción
#	disyunción
!	negación

Delimitadores y otros símbolos

( )	paréntesis
\ /	delimitador de bloque
?	delimitador de expresión
,	separador
|	asignación
@	inicio de comentario de línea
{ }	delimitador de comentario multilínea
[ ]	delimitadores para arreglos

Elementos léxicos

`Id`	identificadores: Nombres de variables/funciones
`entero`	constantes numéricas enteras
`flotante`	constantes numéricas de punto flotante
`caracter`	constantes de tipo caracter
`cadena`	constantes de tipo string
`booleano`	constantes booleanas: true, false

Lista de No Terminales

`program`	Estructura principal del programa
`block`	Bloques de código delimitados
`decl`	Declaraciones generales
`decl_var`	Declaración de variables
`decl_fun`	Declaración de funciones
`stmt`	Sentencias de código
`assign`	Sentencias de asignación
`exp`	Expresiones de código
`exp_arit`	Expresiones aritméticas
`exp_un`	Expresiones aritméticas unarias
`exp_rel`	Expresiones relacionales
`exp_log`	Expresiones lógicas
`exp_bool`	Expresiones booleanas
`exp_comb`	Combinación de expresiones
`ctrl`	Estructuras de control
`if`	Estructura condicional if-elif-else
`do_while`	Estructura iterativa do-while

`for`	Estructura iterativa for
`break`	Sentencia break
`return`	Sentencia return
`read`	Funciones de entrada
`write`	Funciones de salida
`tipo`	Tipos de dato
`tipo_arr`	Tipo de arreglo
`acc_arr`	Acceso a elementos de arreglo
`param`	Parámetros de función
`arg`	Argumentos de llamada a función
`comment`	Comentario de una línea
`comment_block`	Comentario de múltiples líneas
`lit`	Valores literales
`term`	Términos para expresiones aritméticas

Símbolo Inicial

`program`

Análisis de resultados

Objetivos

Objetivos	**Alcanzado**	**No alcanzado**
Diseño de una gramática completa	X	
Tipado explícito y fuerte	X	
Operaciones matemáticas con precedencia	X	
Modularidad	X	

Claridad y legibilidad	X	

1. Se ha logrado definir una gramática BNF que cumple con todos los requisitos especificados en la tarea, incluyendo tipos de datos, estructuras de control, operadores, y funcionalidades de entrada/salida.
2. La gramática implementa un sistema de tipado explícito donde todas las variables deben ser declaradas con su tipo antes de ser utilizadas, lo que permite la detección temprana de errores de tipo.
3. Se ha logrado implementar la precedencia matemática natural mediante la estructura jerárquica de las producciones, siguiendo un modelo similar al lenguaje C.
4. La gramática está organizada de manera modular, permitiendo una fácil extensión o modificación de componentes específicos sin alterar todo el diseño.
5. A pesar de la complejidad del lenguaje diseñado, la gramática se ha estructurado de manera clara, facilitando su comprensión y posible implementación.

Lecciones Aprendidas

1. **Complejidad de diseño de lenguajes:** Para lograr el diseño de una gramática formal, se requieren muchos casos a evaluar y además de eso tiene que ser consistente en todas las reglas, así que con cada característica que se añadía se aumentaba la complejidad.
2. **Importancia de la jerarquía en expresiones:**
3. **Balance entre flexibilidad y restricciones**
4. **Ambigüedades gramaticales**
5. **Valor de la notación BNF**
6. **Consideraciones para sistemas empotrados**

Bitácora

https://github.com/Bayronjrc/tarea1_Compi_Bayron_Gadyr.git