{ Manual Técnico- IDE Pascal }

— SymPascal —

Nombre:

Maria Monserrat Gomez Rabatu

Registro Académico:

202030849

Curso:

Organización de Lenguajes y Compiladores 2

{

Introducción

Este manual describe el funcionamiento interno y la implementación del analizador para archivos en lenguaje Pascal. El sistema utiliza las herramientas JFlex para el análisis léxico y CUP para el análisis sintáctico, junto con estructuras en Java para realizar el análisis semántico.

Estructura del Proyecto

El proyecto está dividido en las siguientes partes:

- **Análisis Léxico:** Utiliza JFlex para generar un analizador léxico que convierte el código fuente en una secuencia de tokens.
- **Análisis Sintáctico:** Utiliza CUP para verificar que los tokens generados sigan las reglas gramaticales de Pascal.
- **Análisis Semántico:** Implementado en Java, analiza la coherencia lógica y el uso correcto de tipos y variables en el código.
- **IDE:** Interfaz gráfica desarrollada en Java para facilitar la interacción con los archivos Pascal.

- **Directorios del proyecto:**
  1. `simpascal/léxico:` Contiene los archivos generados por JFlex.
  2. `simpascal/sintáctico:` Contiene los archivos de CUP.
  3. `simpascal/interfaz:` Contiene la implementación del entorno gráfico que permite cargar y editar archivos Pascal.
  4. `simpascal/instrucciones:` Contiene todo el análisis semántico realizado con clases Java.

}

{

Tecnologías Utilizadas

- **Java (JDK 20 o superior):** Utilizado como lenguaje principal para implementar la lógica del compilador y el IDE.
- **JFlex:** Herramienta para generar analizadores léxicos. Convierte el código fuente en una secuencia de tokens.
- **CUP (Java-based parser generator):** Herramienta para generar analizadores sintácticos, basada en la gramática definida para Pascal.
- **Swing:** Utilizado para construir la interfaz gráfica del usuario (IDE).

Análisis Léxico

El análisis léxico se implementa utilizando JFlex. La definición de los tokens léxicos de Pascal (palabras reservadas, identificadores, operadores, etc.) está en el archivo lexer.flex.

- **Definición de token** En el archivo lexico.flex, se definen los patrones regulares para los tokens de Pascal, incluyendo:
- Palabras reservadas.
- Operadores.
- Identificadores: Cualquier cadena que comience con una letra y pueda contener letras o números.
- Comentarios: Se reconocen comentarios de una línea { ... } y de múltiples líneas (* ... *).

El archivo lexer.flex genera la clase Lexico.java, que es utilizada para transformar el código fuente Pascal en una secuencia de tokens.

{

Análisis Sintáctico

El análisis sintáctico está implementado con CUP. La gramática de Pascal está definida en el archivo parser.cup. CUP toma los tokens generados por el léxico y verifica que sigan las reglas de la sintaxis de Pascal.

- **Gramática de Pascal:** La gramática define las reglas de construcción de un programa Pascal, incluyendo:
- **Programas:** El bloque `program` y las secciones de declaración (`type`, `const`, `var`), así como los subprogramas (`function`, `procedure`) y el bloque principal `begin-end`.
- **Expresiones y declaraciones:** Definiciones de variables, operaciones aritméticas y lógicas, y sentencias de control y ciclos.

El archivo `parser.cup` genera las clases `Parser.java` y `Sym.java`, que implementan el analizador sintáctico del compilador.

Análisis Semántico

El análisis semántico se encarga de verificar que el código sea lógico y coherente, como asegurarse de que las variables estén correctamente declaradas antes de usarse y que los tipos sean compatibles en las expresiones.

- **Tabla de Símbolos:** Se utiliza una tabla de símbolos para registrar todas las variables, constantes, tipos y subprogramas definidos en el programa

}

Gramática BNF:

<inicio> ::= "PROGRAM" <ID> ";" <i_declaracion> <i_fun_proc> <i_instrucciones> "." ;

<i_declaracion> ::= <dec_type_opt> <dec_const_opt> <dec_var_opt> ;

<dec_type_opt> ::= <dec_type> | ε ;

<dec_const_opt> ::= <dec_const> | ε ;

<dec_var_opt> ::= <dec_var> | ε ;

<dec_type> ::= "TYPE" <types_list> ;

<types_list> ::= <types_list> <types_aux> | <types_aux> ;

<types_aux> ::= <variable> "=" <tipos> ";"

| <variable> "=" "ARRAY" "[" <rango> "]" "OF" <tipos> ";" ;

<dec_const> ::= "CONST" <constantes_list> ;

<constantes_list> ::= <constantes_list> <constante> | <constante> ;

<constante> ::= <ID> "=" <expresion> ";" ;

<dec_var> ::= "VAR" <variables_list> ;

<variables_list> ::= <variables_list> <var_aux> | <var_aux> ;

<var_aux> ::= <variable> ":" <tipos> ";"

| <variable> ":" "ARRAY" "[" <rango> "]" "OF" <tipos> ";" ;

<rango> ::= <ENTERO> ".." <ENTERO> ;

<variable> ::= <variable> "," <ID> | <ID> ;

<i_fun_proc> ::= <i_funciones> <i_procedimientos> | <i_funciones> |
<i_procedimientos> | ε ;

<i_instrucciones> ::= "BEGIN" <instrucciones> "END" ;

<instrucciones> ::= <instrucciones> <instruccion> | <instruccion> ;

<instruccion> ::= <i_write> ";"

| <i_read> ";"

| <i_asignar> ";"

| <i_if> ";"

| <i_for> ";"

| <i_while> ";"

| <i_repeat> ";"

| <llam_proc> ";" ;

<tipos> ::= "INTEGER"

| "REAL"

| "CHAR"

| "STRING"

| "BOOLEAN"

| <ID> ;

<i_write> ::= "WRITELN" "(" <write_aux> ")" ;

<i_read> ::= "READLN" "(" <variable> ")" ;

<write_aux> ::= <write_aux> "," <expresion> | <expresion> ;

<i_asignar> ::= <ID> ":=" <expresion> ;

<i_if> ::= "IF" "(" <expresion> ")" "THEN" <ins_statements> <i_elseIfsB> <i_elseB>
;

<i_elseIfsB> ::= <i_elseIfs> | ε ;

<i_elseIfs> ::= <i_elseIfs> <i_elseIf> | <i_elseIf> ;

<i_elseIf> ::= "ELSEIF" "(" <expresion> ")" "THEN" <ins_statements> ;

<i_elseB> ::= <i_else> | ε ;

<i_else> ::= "ELSE" <ins_statements> ;

<ins_statements> ::= <instruccionU>

| "BEGIN" <instrucciones> <instruccion> "END" ;

<instruccionU> ::= <i_write>

| <i_read>

| <i_asignar>

| <i_if>

| <i_for>

| <i_while>

| <i_repeat>

| <llam_proc> ;

<i_for> ::= "FOR" <ID> ":=" <expresion> "TO" <expresion> "DO" <ins_statements>
;

<i_while> ::= "WHILE" "(" <expresion> ")" "DO" <ins_statements> ;

<i_repeat> ::= "REPEAT" <ins_statements> "UNTIL" <expresion> ;

<i_funciones> ::= <i_funciones> <funcion> | <funcion> ;

<funcion> ::= "FUNCTION" <ID> "(" <parametros> ")" ":" <tipos> ";"
<i_declaracion> <i_instrucciones> ";" ;

<llam_fun> ::= <ID> "(" <llam_par> ")" ;

<llam_proc> ::= <ID> "(" <llam_par> ")" ;

<llam_par> ::= <llam_par> "," <expresion> | <expresion> ;

<i_procedimientos> ::= <i_procedimientos> <procedimiento> |
<procedimiento> ;

<procedimiento> ::= "PROCEDURE" <ID> "(" <parametros> ")" ";"
<i_declaracion> <i_instrucciones> ";" ;

<parametros> ::= <parametros> "," <variable> ":" <tipos> | <variable> ":" <tipos> ;

<expresion> ::= <e_arit>

| <e_relac>

| <e_logic>

| <e_nativ>

| <casteo>

| <ID>

| <ID> "[" <expresion> "]"

| "(" <expresion> ")"

| <llam_fun> ;

<e_arit> ::= "-" <expresion>

| <expresion> "*" <expresion>

| "DIV" "(" <expresion> "," <expresion> ")"

| "MOD" "(" <expresion> "," <expresion> ")"

| <expresion> "+" <expresion>

| <expresion> "-" <expresion> ;

<e_relac> ::= <expresion> "=" <expresion>

| <expresion> "<>" <expresion>

| <expresion> ">" <expresion>

| <expresion> "<" <expresion>

| <expresion> ">=" <expresion>

| <expresion> "<=" <expresion> ;

<e_logic> ::= "NOT" <expresion>

| <expresion> "OR" <expresion>

| <expresion> "AND" <expresion> ;

<e_nativ> ::= <ENTERO>

| <DECIMAL>

| <CADENA>

| <CARACTER> ;

```
}
```