Lenguajes de programación

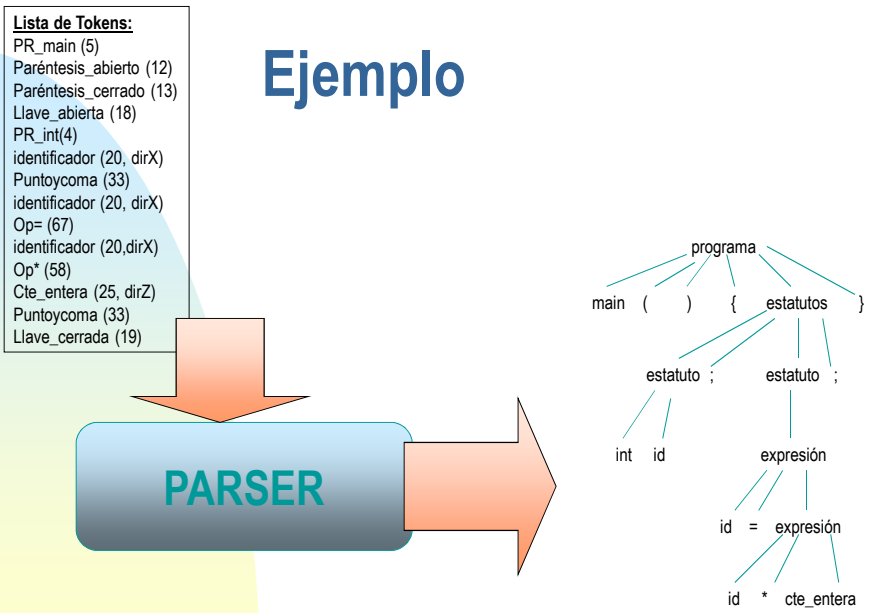
¿Cómo se construye un compilador?

=> ANÁLISIS DE SINTAXIS

Analizador de SINTAXIS

- Verifica que la secuencia de tokens cumpla con el orden definido por la gramática del lenguaje.
- Se forman frases gramaticales (estatutos) en forma de árbol.
- La gramática debe de ser LIBRE DE CONTEXTO para poder reconocer las estructuras típicas de un lenguaje de programación.

Ejemplo



¿Cómo se construye un Analizador de Sintaxis?

- Diseñar la gramática del lenguaje y quitarle ambigüedades. Un diagrama de sintaxis puede facilitar el diseño.
- Implementar por medio de algún método:
  - Descenso recursivo
  - LL – Descendente predictivo
  - LR - Ascendente
- Utilizar un generador automático de parser: Yacc, Bison, Javacc
  - Gramática LR -> AEF -> Método LR

¿Cómo distinguir un elemento de LEXICO de uno de SINTAXIS?

- El analizador de léxico se debe ocupar de las estructuras sintácticas de más bajo nivel, correspondientes a los elementos básicos del lenguaje.
- Si la definición de un elemento es INHERENTEMENTE RECURSIVA, es tarea del analizador sintáctico reconocer ese elemento.
- *Ejemplo*: Definición de una expresión.

El método de Descenso recursivo

- Consiste en programar un **módulo por cada símbolo No Terminal** de la gramática a reconocer.
- La programación del módulo se basa en la secuencia de la producción del No Terminal, incluyendo :
  - La verificación de la presencia de un símbolo terminal en la entrada (con un if).
  - La llamada al módulo del símbolo No terminal correspondiente (incluso recursivamente si se requiere).

Ejemplo

Gramática:

No Terminales: {A, B}

Terminales: {a, b, c}

Producciones:

A ::= a B a

B ::= b B b

B ::= c

```
procedure A;
 begin
   if (token == 'a')
      token = dame_token();
      B;
      if (token == 'a')   cout << "Entrada exitosa";
     else   cerr << "Error, esperaba 'a'";
     end if
  else  cerr << "Error, esperaba 'a'";
  end if;
end A;

procedure B;
begin
   if (token =='b')
      token = dame_token();
      B();
      if (token == 'b') token = dame_token();
      else  cerr << "Error, esperaba 'b'";
      end if
   else
       if (token == 'c')  token = dame_token();
      else  cerr << "Error, esperaba 'b' o 'c'";
      end if
   end if
end B
```

Método Básico Recursivo Descendente

- En el ejemplo se puede simplificar mucho si introducimos un procedimiento *match* que empate con los token de previsión.
- Si el procedimiento *match* resulta verdadero, la entrada es válida y se obtiene el siguiente token; si no, se marca un error.

```
procedure match( tokenEsperado)
begin
        if token = tokenEsperado then
                dame_token();
        else
                cerr << "Error, esperaba"
                        << tokenEsperado;
        end if;
end match;
```

Nuevo Código

```
procedure A;
 begin
   match('a');
   B;
   match('a')
   cout << "Entrada exitosa";
end A;

procedure B;
begin
   case token of
   'b': match('b')
        B;
        match('b');
    'c': match('c');
     else cerr << "Error, esperaba 'b' o 'c'";
    end case;
end B
```

Características del Analizador DR

- Fácil y rápido de implementar.
- Fácil de marcar los errores, pero difícil de recuperarse de ellos.
- Requiere que la gramática no tenga ambigüedades:
  - No debe haber producciones con recursividad izquierda: A::= A a | b
  - No debe haber producciones con factores comunes: A::= a B | a C
- Sólo para gramáticas LL.

EJEMPLO en C++

EXP -> EXP op EXP
EXP -> (EXP)
EXP -> cte

EXP -> cte EXP1
EXP -> (EXP) EXP1
EXP1 -> op EXP EXP1
EXP1 -> ε

```
#include <iostream>
using namespace std;

char token; // siguiente token

//Declaracion de prototipos para tener recursividad mutua
void exp(void);
void exp1(void);

// empata y obtiene el siguiente token
void match(char tokenEsperado) {
    if (token == tokenEsperado) {
            token = getchar();
            cout << token;
    } else {
            cerr << "Error: se esperaba " << tokenEsperado << endl;
            exit(1);
    }
}

int main(void) {
    token = getchar();  // inicializa con el primer token
    cout << token;
    exp();
    if (token == '\n')
            cout << "Expresion bien construida\n";
    else
            cerr << "Expresion mal construida\n";
}
```

EJEMPLO en C++

EXP -> cte EXP1
EXP -> (EXP) EXP1
EXP1 -> op EXP EXP1
EXP1 -> ε

```
// reconoce expresiones
void exp(void)
{
     switch (token) {
            case '0':   match('0'); // reconoce CTE
                        exp1();
                        break;
            case '(':   match('('); // reconoce (
                        exp();
                        match(')'); // reconoce )
                        exp1();
                        break;
            default:    cerr << "Error: se esperaba CTE o (\n";
                        exit(0);
     }
}

// auxilia el reconocimiento de expresiones
void exp1(void)
{
     if (token == '+') {
            match('+'); // reconoce operador
            exp();
            exp1();
     }
}
```