Práctica PLG, el lenguaje Ned

Mayra Alexandra Castrosqui Florián
Jaime Dan Porras Rhee

Abril 2015

Resumen

En esta entrega se especificará el léxico y la sintaxis del lenguaje. Se expondrán ejemplos típicos de programas en dicho lenguaje.

Descripción del lenguaje

Ned es un lenguaje de programación imperativo. Un programa escrito en Ned se divide en cuatro bloques. Los tres primeros para declaraciones y el último que contiene el código a ejecutar. En el primero se declaran las variables globales. En el segundo bloque se declaran las clases con sus métodos y atributos. En el tercero se declaran funciones y procedimientos. Y en el último, rodeado por un `Begin End` se encuentra el código principal. Esquemáticamente sería de la forma

```
Declare var
...
Declare class
...
Declare function
...
Begin
...
End
```

Características del lenguaje

Identificadores y ámbitos de definición
Clases, procedimientos y funciones. El lenguaje tendrá ámbitos de declaración dinámicos.

Tipos
Tipos con nombre y definición de tipos de usuario.

Instrucciones ejecutables
Instrucción case. Llamadas a procedimientos, funciones y métodos de clase.

Errores
Tratar de proseguir la compilación tras un error, a fin de detectar más errores.

1. Análisis léxico

Se tendrán las siguientes clases de unidades léxicas:

- Identificadores.
- Palabras reservadas.
- Constantes literales para los tipos int, float, char, string y bool.
- Símbolos.

Algunas palabras reservadas serán `break`, `continue`, `const` entre otras que irán apareciendo en este documento.

Los identificadores podrán admitir mayúsculas, minúsculas y "_". Los identificadores de variables y funciones empiezan por minúscula y los de clases por mayúscula.

$$id \rightarrow Min(Min \mid May \mid Num \mid \_)^*$$
$$Id \rightarrow May(Min \mid May \mid Num \mid \_)^*$$

Las constantes literales

$$int \rightarrow Num^*$$
$$float \rightarrow Num^+.[Num^+]$$
$$char \rightarrow \text{`}c\text{'}$$
$$string \rightarrow \texttt{"}c^*\texttt{"}$$
$$bool \rightarrow \texttt{True} \mid \texttt{False}$$

Auxiliares:

$$Num \rightarrow \texttt{0} \mid .. \mid \texttt{9}$$
$$Min \rightarrow \texttt{a} \mid .. \mid \texttt{z}$$
$$May \rightarrow \texttt{A} \mid .. \mid \texttt{Z}$$
$$c \rightarrow Num \mid Min \mid May \mid Sym$$

2. Análisis sintáctico

El lenguaje Ned se describe mediante una gramática incontextual G.

$$G = (V_N, V_T, P, Code)$$

Las variables terminales son las unidades léxicas que obtenemos del análisis léxico. Las no terminales irán apareciendo en las reglas, además $Code \in V_N$. Veamos las reglas para la gramática de nuestro lenguaje.

Reglas

Reglas para la estructura principal del código

$$Code \rightarrow Opc\ \texttt{Begin}\ S\ \texttt{End}$$
$$Opc \rightarrow [\texttt{Declare var}\ DV][\texttt{Declare class}\ DC][\texttt{Declare function}\ DF]$$
$$DV \rightarrow (tipo(id\texttt{;} \mid ASIG))^+$$
$$DC \rightarrow (\texttt{Class}\ Id\ \{\ \texttt{public}\ (DF \mid DV)^*\ \texttt{private}\ (DF \mid DV)^*\ \})^+$$
$$DF \rightarrow (P \mid F)^+$$
$$P \rightarrow id\ \texttt{(}[tipo\ id\ (\texttt{,}\ tipo\ id)^*]\texttt{)\{}\ S\ \texttt{\}}$$
$$F \rightarrow tipo\ id\texttt{(}[tipo\ id\ (\texttt{,}\ tipo\ id)^*]\texttt{)\{}\ S\ \texttt{\}}$$

Reglas para las instrucciones

$$S \rightarrow DV \mid ASIG \mid IF \mid WHILE \mid FOR \mid FOREACH \mid REPEAT \mid SWITCH \mid RET$$
$$ASIG \rightarrow id\ \texttt{=}\ VAL[\ \texttt{[}Num\texttt{]}\ ]\texttt{;}$$
$$IF \rightarrow \texttt{if}\ B\ \texttt{then}\ S\ \texttt{else}\ S\ \texttt{endif}$$
$$WHILE \rightarrow \texttt{while}\ B\ \texttt{do}\ S\ \texttt{endwhile}$$
$$FOR \rightarrow \texttt{for (}\ ASIG\texttt{;}[B(\texttt{,}B)^*]\texttt{;}ASIG\ \texttt{)}\ S\ \texttt{endfor}$$
$$FOREACH \rightarrow \texttt{foreach}\ id\ \texttt{in}\ id\ \texttt{do}\ S\ \texttt{endforeach}$$
$$REPEAT \rightarrow \texttt{repeat}\ S\ \texttt{until}\ B\ \texttt{endrepeat}$$
$$SWITCH \rightarrow \texttt{switch (}\ id\ \texttt{)}\ \ (\texttt{case (}\ VAL\ \texttt{)}\ S)^+\ [\texttt{Default}\ S]\ \texttt{endswitch}$$
$$RET \rightarrow \texttt{return}\ VAL\texttt{;}$$

Otras reglas

$$tipo \rightarrow \texttt{int} \mid \texttt{float} \mid \texttt{char} \mid \texttt{string} \mid \texttt{bool}$$
$$VAL \rightarrow char \mid string \mid A \mid B \mid id$$

Reglas para expresiones aritméticas y booleanas

$$A \to id \mid int \mid float \mid A\texttt{+}A \mid A\texttt{-}A \mid A\texttt{*}A \mid A\texttt{\^{}}A \mid A\texttt{\%}A \mid A\texttt{:}A \mid A\texttt{/}A \mid \texttt{(}A\texttt{)}$$
$$B \to bool \mid B\texttt{\&\&}B \mid B\texttt{||}B \mid \texttt{!}B \mid \texttt{(}B\texttt{)} \mid C$$
$$C \to A\ comp\ A$$
$$comp \to \texttt{==} \mid \texttt{!=} \mid \texttt{<=} \mid \texttt{>=} \mid \texttt{<} \mid \texttt{>}$$

Los símbolos de *comp* son operadores infijos. Tienen todos la misma prioridad y no son asociativos. Con respecto a los operadores para la expresiones aritméticas y booleanas tenemos los siguientes cuadros.

Operador	Tipo	Prio	Asociatividad
^	binario infijo	0	asoc. a derechas
*	binario infijo	1	asoc. a izquierdas
/	binario infijo	1	asoc. a izquierdas
:	binario infijo	2	asoc. a izquierdas
%	binario infijo	3	asoc. a izquierdas
+	binario infijo	4	asoc. a izquierdas
-	binario infijo	4	asoc. a izquierdas

Operador	Tipo	Prio	Asociatividad
&&	binario infijo	0	asoc. a derechas
||	binario infijo	1	asoc. a izquierdas
!	unario prefijo	1	asociativo

```
Declare Var
float r=3.14;

Declare Class

Class Racional{
        public
        Racional(int n1, int n2){
                num = n1;
                den = n2;
        }
        int divEq(int n){
                num = num/n;
                den = den/n;
        }
        float toFloat(){
                return num/den;
        }
        int getNum(){
                return num;
        }
        int getDen(){
                return den;
        }
        private
        int num;
        int den;

}

Declare Function
        int mcd(int a, int b){
                int mcd = 1;
                for (i=1; i<min(a,b);i=i+1)
                        if a%i == 0 && b%i == 0 then
                                if i>mcd then mcd = i;
                                endif
                        endif
                endfor
        return mcd;
        }

        int min(int a, int b){
                if a<b then return a;
                else return b;
        }

Begin
        Racional r1 = Racional(1,2);
        Racional r2 = Racional(1,3);
        r1=r1+r2;
        r1.diveq(mcd(r1.getnum, r1.getden));
        r=r1.tofloat()*r*2;
End
```

```
Declare function
int factorial(int n){
        int fact=1;
        foreach i in {1..n}
                fact=fact*i;
        endforeach
        return fact;
}

Begin
        int v=8;
        int resultado=factorial(v);
End
```