Universidad de Oviedo

Diseño de Lenguajes de Programación

Grado en Ingeniería Informática del Software

Escuela de Informática de Oviedo

Análisis Sintáctico

- El objetivo de esta práctica es definir el analizador sintáctico para nuestra especificación de lenguaje.
- Para ayudarnos a entender lo que estamos haciendo, leeremos el fichero **2Sintactico.pdf** del capítulo 2 del Tutorial.
- Abriremos el Eclipse.
- Creamos un Nuevo Proyecto Java que denominaremos **DLP2015-2016v2** a partir del proyecto **DLP2015-2016v1** (copiamos el proyecto anterior con el nuevo nombre)

Requisitos Sintácticos

- Los tipos de datos permitidos son tipos básicos: **character**, **integer**, **real** y estructurados: **struct** y vectores multidimensionales.
- El lenguaje soportará el operador de casting **()** a alguno de los tipos básicos (de **integer** a **real** y de **real** a **integer**).
- Al principio del programa (sección types) podrán aparecer definiciones de tipos definidos por el usuario (estructuras).

Requisitos Sintácticos (y II)

```
struct miestructura

  integer campo1;

  character campo2;

endstruct
```

- A continuación podrán aparecer definiciones de variables globales (sección **globals**).
- Las variables locales se declararán al principio de las funciones definidas por el usuario antes que cualquier sentencia.
- No se permite la declaración de variables al principio de cualquier bloque o sentencia múltiple.

Requisitos Sintácticos (y III)

- El lenguaje debe permitir la declaración múltiple de variables de un tipo en una sentencia única, de la forma:

```
integer a, b , c;
```

- La declaración de tipos es explícita. Así pues, tanto a nivel global, como local a una función, podremos encontrar declaraciones como estas:

```
integer a;
character c0,c1,c2;
```

Requisitos Sintácticos (y IV)

```
real f1,f2;
struct miestructura r1;
integer [5][5] matriz;
```

- La entrada y salida de datos se realizará mediante las funciones **read** y **write**.

```
read (var1, var2,…, varn);
write (exp1, exp2,…, expn);
```

Requisitos Sintácticos (y V)

- Las expresiones podrán estar formadas por literales enteros, literales reales y variables combinados mediante operadores aritméticos o lógicos.
- Se podrán agrupar expresiones mediante paréntesis.
- Todas las instrucciones del lenguaje terminan en punto y coma.

Requisitos Sintácticos (y VI)

- El lenguaje permitirá sentencias condicionales con parte **else** no obligatoria.

```
if (a gt 3) then
   b=0;
endif
if (a eq 2) then
 write(k);
else
 write(m);
endif
```

Requisitos Sintácticos (y VII)

- Bucles while.

```
while ( i gt 0)
   i= i-1;
endwhile
```

- Llamadas a funciones definidas por el usuario.

```
k = f1(x,y+5,23);
```

Requisitos Sintácticos (y VIII)

```
types

struct tipo1
  integer edad;
  real sueldo;
endstruct

globals

integer j, [10]k;
character c;
real i;
struct tipo1 [10]r1;

procedures

function inicia (integer [3]vparam) as
integer
   integer [3]vlocal;

   k[1]=79;
   vlocal[1]=79;
   vparam[2]=80;
   return(2);
endfunction
```

```
main ()

  j=0;
  if (j gt 5)
  then
    write(1);
  else
    write(2);
  endif
  while (j lt 5)
    write(j);
    j=j+1;
  endwhile
  r1[0].edad=15;

endmain
```

¿Por dónde empezamos?

- En esta fase se requiere un metalenguaje que permita describir de manera precisa las estructuras que forman un programa válido, qué elementos las componen y en qué orden.
- Se usará para ello una **Gramática Libre de Contexto** que será expresada mediante la notación **BNF**.
- Una vez modificados los ficheros **lexico.l** y **sintac.y**, ir a la carpeta **sintáctico** del proyecto y ejecutar el fichero de proceso por lotes **genera.bat**.
- Desde **genera.bat** se invoca a **JFlex** y **BYaccJ** los cuales generan el código java del analizador léxico y sintáctico respectivamente.
- Para probarlo crear un fichero de entrada (**entrada.txt**) dentro de la carpeta **src** del proyecto y ejecutar la clase **main.Main**.

Conflictos y su resolución

- La **ambigüedad en la gramática** conduce a que **yacc**, ante una cadena de tokens, pueda agruparlos de diferentes formas ⇒ **diferentes árboles sintácticos**.
- La **ambigüedad genera conflictos**, pero en cambio **facilita** la **escritura de la gramática**.
- **Yacc** muestra los conflictos detectados en la especificación de la grámática en la **salida estándar** (opción **–v**) y en el fichero **y.output** de la carpeta **sintactico** del proyecto.

Conflictos y su resolución (y II)

- Tipos de conflictos:
  - **Shift/Reduce**
    - Ej.:

```
programa:       x
          |     y R;
x: A ↑ R;
y: A ↑;
```

    - Lugares típicos donde ocurre:
      - Parte de la gramática usada para definir expresiones (ej.: **expr**: **expr op expr**).
      - IF–THEN-ELSE
      - Bucles anidados.

Conflictos y su resolución (y III)

- Tipos de conflictos…:
  - **Shift/Reduce…**
    - **Posibles soluciones**:
      - Reescritura de la gramática.
      - Usar declaraciones de precedencia y asociatividad de operadores (**%left,%right,%nonassoc**) en la sección de definiciones del fichero **sintac.y**

Conflictos y su resolución (y IV)

- Tipos de conflictos…:
  - **Reduce/Reduce**
    - Ej.:

```
programa:     x
            | y ;
x: A ↑ ;
y: A ↑;
```

    - **Lugares típicos donde ocurre**:
      - Reglas con el mismo RHS o que se solapan parcialmente.
    - **Posibles soluciones**:
      - Reescritura de la gramática haciendo las alternativas disjuntas (sin solapamientos).
      - Si no es posible, dejar conflicto **reduce/reduce** ⇒**Yacc** resuelve el conflicto por el orden de definición de cada regla en el fichero **sintac.y**

Anexo. Precedencia y asociatividad de los operadores

Precedencia	Operador	Asociatividad
Mayor	not, - (menos unario)	no asociativo
	*,/,%	izquierda
	+,-	izquierda
	lt,le,gt,ge	izquierda
	eq,ne	izquierda
	and	izquierda
Menor	or	izquierda