Seminario de introducción a Bison

David Portolés Rodríguez

dporto@unizar.es

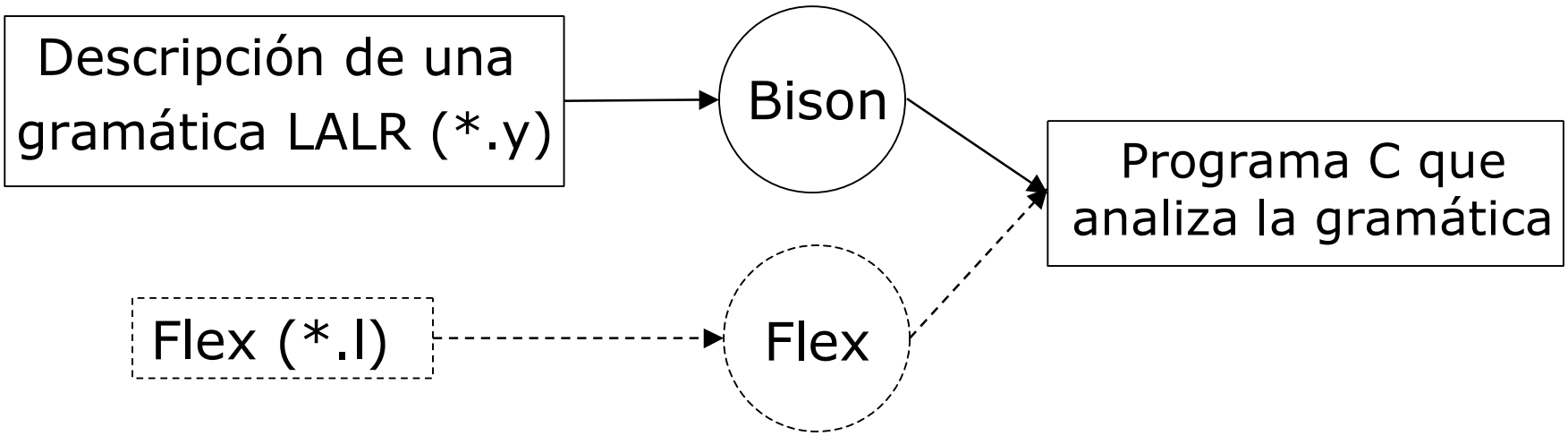
Lenguajes y Sistemas Informáticos

Dpto. de Informática e Ing. de Sistemas

Universidad de Zaragoza

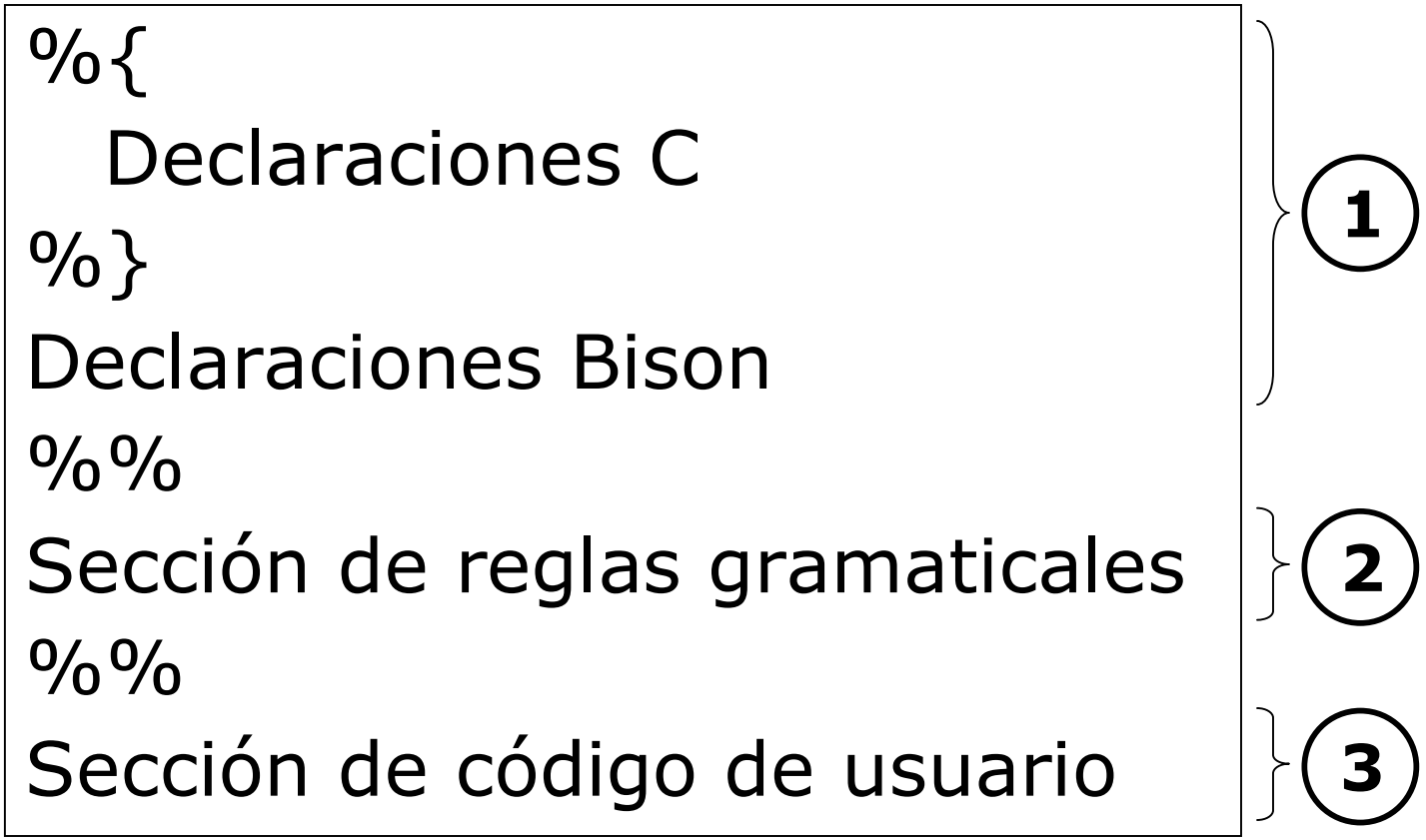
¿Qué es Bison?

- Bison es un una herramienta que permite generar analizadores sintácticos
- Compatible con Yacc
- Flex + Bison = Compilador
- Supondremos que siempre interactuamos con Flex



Estructura de un fichero Bison

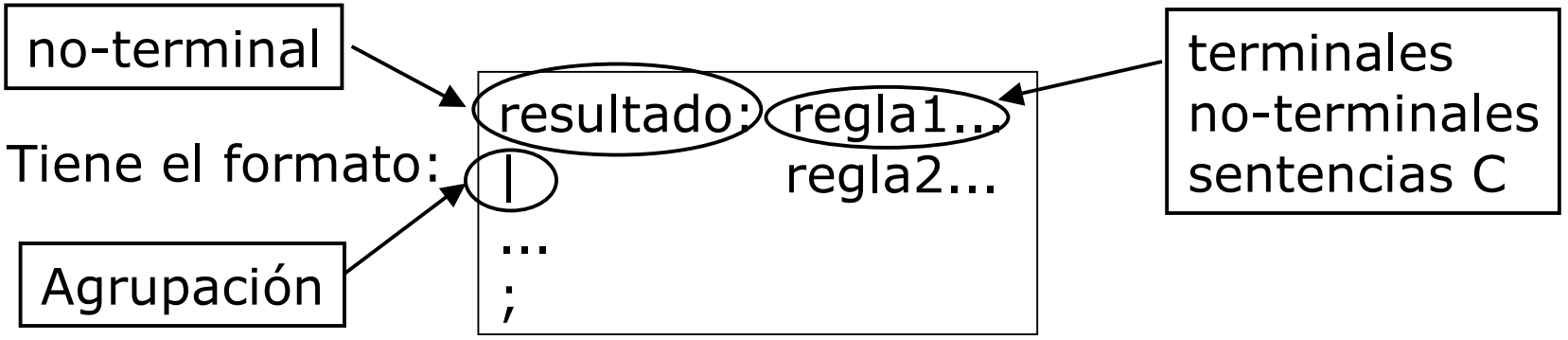
- Tres partes separadas por %%



Sección de declaraciones

- Declaraciones C
  - Entre %{ %}
  - Tipos y variables usadas en las acciones
  - Include's, define's
- Declaraciones Bison
  - Símbolos terminales (tokens) y no terminales
    - %token
  - Precedencia de operadores
    - %left, %right, %nonassoc, %prec
  - Tipos de datos de los valores semánticos de varios símbolos
    - %union, %type, %token
  - Símbolo inicial
    - %start

Sección de reglas gramaticales



- Tiene el formato:

- Símbolos terminales: tokens
  - Declarados con %token o bien el propio carácter
- Reglas gramaticales: producciones
  - Se permite la regla vacía (ε)
  - Recursiva: aparece la parte izq. en la parte dcha.
    - Tipos: Recursiva por la izq y por la dcha, indirecta
    - Preferible por la izquierda: ocupa menos espacio de pila
  - Pb. ambigüedad: conflicto shift-reduce, reduce-reduce (ver teoría COMPI: lección 5, página 69 y ss.)

Consejos para la sección de reglas

- Conviene incluir /* vacío */ en regla vacía (ε)
- Tokens en mayúsculas y no-terminales en minúsculas
- Conviene tabular y situar un token por línea
  - Muy extenso, pero más legible
  - Si hay código C => un tabulador más
  - Todos los símbolos alineados en la misma columna

```
P_i : S_1 { inst_1 ... inst_n } S_2 { inst_1 ... inst_n } .... S_k { inst_1 ...
inst_n } | T_1 { inst_1 ... inst_n } ... T_l { inst_1 ... inst_n };
```

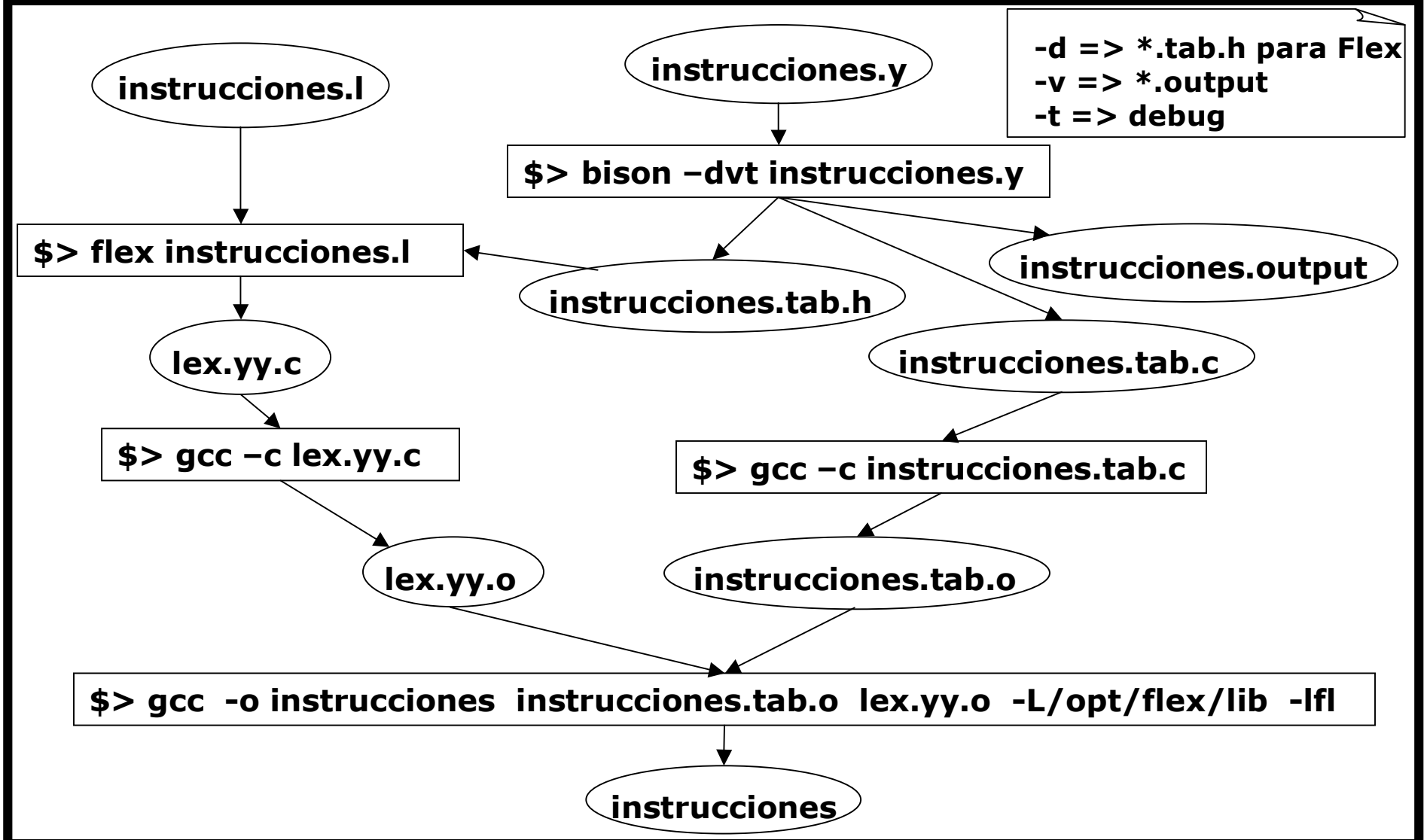
vs

```
P_i :
   S_1
      { inst_1
        ...
        inst_n
      }
   S_2
      { inst_1
        ...
        inst_n
      }
   ....
   S_k
      { inst_1
        ...
        inst_n
      }
|  T_1
      { inst_1
        ...
        inst_n
      }
   ...
   T_l
      { inst_1
        ...
        inst_n
      }
;
```

Sección de código de usuario

- Código C creadas por necesidad del programador
- Se copian literalmente en el *.tab.c
- main() está en el Bison (.y) y nunca en el Flex (.l)
- Bison necesita definir yyerror() (excepto –ly)
- yyparse()
  - Realiza el análisis sintáctico
  - Devuelve 0 si ok, 1 si fallo
  - Se crea por defecto, pero puede redefinirse
  - Es la análoga a yylex() de Flex
  - No lee directamente de la stdin => recibe tokens de yylex()
- yylex()
  - Lo habitual es usar la de Flex, pero puede redefinirse
  - Es la que proporciona tokens a Bison
  - En Flex: acciones terminan con return()

Proceso de compilación



Ficheros generados

- El fichero *.tab.h sirve como cabecera para Flex
  - Crea tipo enumerado para los tokens que se usarán en Flex
  - Ahorra definir los tokens en Flex
- El fichero *.output sirve para ver la gramática generada y depurarla si hay conflicto
- Opción –y de Bison
  - Comportamiento como Yacc
  - El resultado siempre se llama: y.tab.c y.tab.h y.output

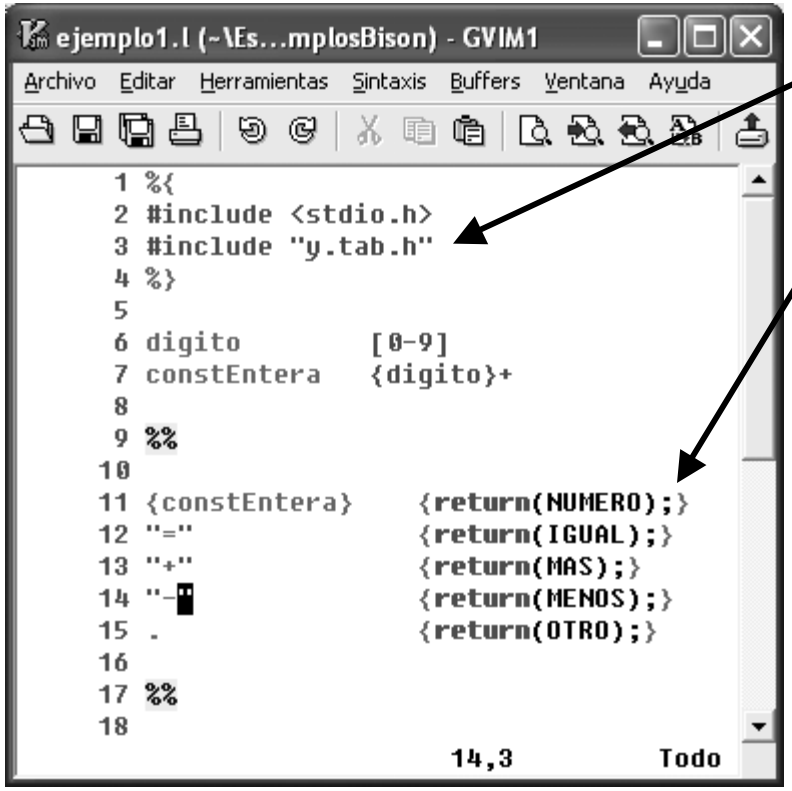
Primer ejemplo en Bison

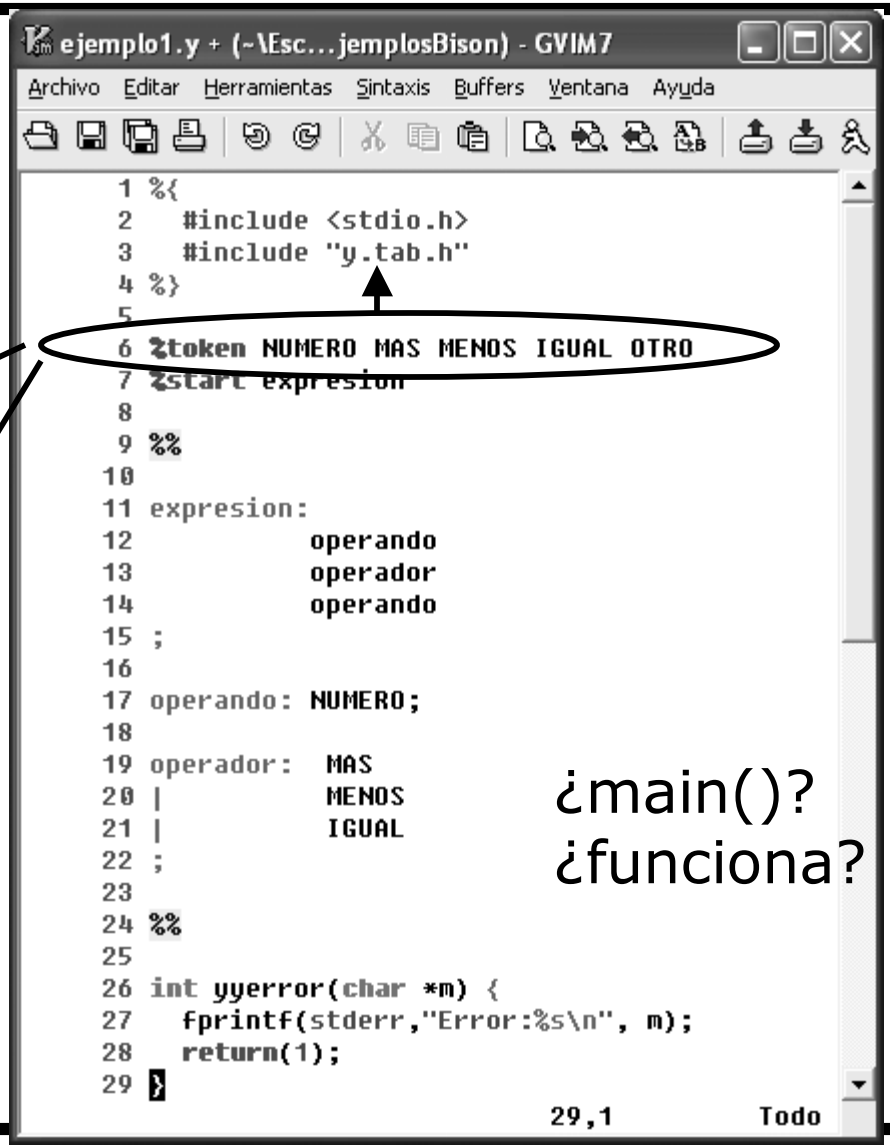
- Reconocedor de:

  op1 + op2

  | op1 – op2

  | op1 = op2





Segundo ejemplo en Bison

- Ídem que antes
- Los tokens carácter se tratan de forma más simple
- main() => yyparse()

ejemplo2.l (~\Esc...mplosBison) - GVIM2

Archivo Editar Herramientas Sintaxis Buffers Ventana Ayuda

```
 1 %{
 2 #include <stdio.h>
 3 #include "y.tab.h"
 4 %}
 5
 6 digito         [0-9]
 7 constEntera    {digito}+
 8
 9 %%
10
11 {constEntera}     {return(NUMERO);}
12 [+-=]             {return(yytext[0]);}
13 .                 {return(OTRO);}
14
15 %%
16
                              12,13-17      Todo
```

ejemplo2.y + (~\E...emplosBison) - GVIM

Archivo Editar Herramientas Sintaxis Buffers Ventana Ayuda

```
 1 %{
 2   #include <stdio.h>
 3   #include "y.tab.h"
 4 %}
 5
 6 %token NUMERO OTRO
 7 %start expresion
 8
 9 %%
10
11 expresion:
12           operando
13           operador
14           operando
15 ;
16
17 operando: NUMERO;
18
19 operador: '+'
20 |         '-'
21 |         '='
22 ;
23
24 %%
25
26 int yyerror(char *m) {
27   fprintf(stderr,"Error:%s\n", m);
28   return(1);
29 }
30
31 int main()
32 {
33   yyparse();
34   return(0);
35 }
                              7,16          Todo
```

Accediendo al valor del token

- **Objetivo: Reconocer y evaluar op1 = op2**
- **"Pasar" el valor desde el léxico al sintáctico**
- **$x es el símbolo x-ésimo de la parte derecha (empieza en 1)**
- **$$ es la parte izquierda**
- **Por defecto son *int***

ejemplo3.l (~\Escrito...\ejemplosBison) - GVIM5

Archivo Editar Herramientas Sintaxis Buffers Ventana Ayuda

```
 1 %{
 2 #include <stdio.h>
 3 #include "y.tab.h"
 4 int var;
 5 %}
 6
 7 digito         [0-9]
 8 constEntera    {digito}+
 9
10 %%
11
12 {constEntera}     {sscanf(yytext,"%d", &yylval);
13                     return(NUMERO);}
14 [+-=]              {return(yytext[0]);}
15 .                  {return(OTRO);}
16
17 %%
18
                                    9,0-1         Todo
```

ejemplo3.y + (~\Escritorio\u...006\compi\ejemplosBison) - GVIM

Archivo Editar Herramientas Sintaxis Buffers Ventana Ayuda

```
 1 %{
 2   #include <stdio.h>
 3   #include "y.tab.h"
 4 %}
 5
 6 %token NUMERO OTRO
 7 %start expresion
 8
 9 %%
10
11 expresion:
12           operando
13           '='
14           operando
15           { if ($1==$3) {
16               printf("operandos son =: %d\n", $1);
17             } else {
18               printf("operandos son != : %d != %d\n", $1, $3);
19             }
20           }
21 ;
22
23 operando: NUMERO
24         { $$=$1; }
25 ;
26
27 %%
28
29 int yyerror(char *m) {
30   fprintf(stderr,"Error:%s\n", m);
31   return(1);
32 }
                                    10,0-1        Todo
```

Accediendo al valor del token (completo)

- Reconocer y evaluar las tres operaciones

ejemplo4.y (~\Escritorio\uniz...006\compi\ejemplosBison) - GVIM3

Archivo Editar Herramientas Sintaxis Buffers Ventana Ayuda

```
 1 %{
 2   #include <stdio.h>
 3   #include "y.tab.h"
 4 %}
 5
 6 %token NUMERO OTRO
 7 %start expresion
 8
 9 %%
10
11 expresion:
12           operando
13           operador
14           operando
15           {
16             if ($2 == '=') {
17               if ($1==$3) {
18                 printf("operandos son =: %d\n", $1);
19               } else {
20                 printf("operandos son !=: %d != %d\n", $1, $3);
21               }
22             } else if ($2 == '+') {
23               printf("la suma de los operandos es: %d\n", ($1+$3));
24             } else if ($2 == '-') {
25               printf("la resta de los operandos es: %d\n", ($1-$3));
26             } else {
27               printf("la resta de es: %d\n", ($2));
28               }
29
30           }
31 ;
                                                    20,16-24 Comienzo
```

ejemplo4.y (~\Es...mplosBison) - GVIM3

Archivo Editar Herramientas Sintaxis Buffers Ventana Ayuda

```
32
33 operando: NUMERO
34           { $$=$1; }
35 ;
36
37 operador: '+'
38           { $$='+'; }
39 |         '-'
40           { $$='-'; }
41 |         '='
42           { $$='='; }
43 ;
44
45 %%
46
47 int yyerror(char *m) {
48   fprintf(stderr,"Error:%s\n", m);
49   return(1);
50 }
51
52 int main()
53 {
54   yyparse();
55   return(0);
56 }
                                                    37,14         Final
```

Utilizando la pila

ejemplo5.y (~\E...mplosBison) - GVIM

Archivo Editar Herramientas Sintaxis Buffers Ventana Ayuda

```
 1 %{
 2   #include <stdio.h>
 3   #include "y.tab.h"
 4 %}
 5
 6 %token NUMERO OTRO
 7 %start expresion
 8
 9 %%
10
11 expresion:
12         operando1
13         operador
14         operando2
15 ;
16
17 operando1: NUMERO
18           { $$=$1; }
19 ;
20
21 operador: '+'
22           { $$='+'; }
23 |         '-'
24           { $$='-'; }
25 |         '='
26           { $$='='; }
27 ;
28
                      7,16      C
```

$0 es operador, $-1 es operando1

ejemplo5.y (~\Escritorio\uniza..._2006\compi\ejemplosBison) - GVIM

Archivo Editar Herramientas Sintaxis Buffers Ventana Ayuda

```
29 operando2: NUMERO
30           {
31            if ($0 == '=') {
32              if ($-1==$1) {
33                printf("operandos son =: %d\n", $-1);
34              } else {
35                printf("operandos son !=: %d != %d\n", $-1, $1);
36              }
37            } else if ($0 == '+') {
38              printf("la suma de los operandos es: %d\n", ($-1+$1));
39            } else if ($0 == '-') {
40              printf("la resta de los operandos es: %d\n", ($-1-$1));
41            }
42           }
43 ;
44
45 %%
46
47 int yyerror(char *m) {
48   fprintf(stderr,"Error:%s\n", m);
49   return(1);
50 }
51
52 int main()
53 {
54   yyparse();
55   return(0);
56 }
                                          31,21-27      Final
```

Utilizando la pila: consideraciones

- Un bloque C también cuenta como un $x

```
      $1       $2   $3              $4                $5    $6
a : token1  token2  b  { acción a media regla; }   c   token3
;
```

- Cuidado...

```
tontería: uno fin | dos fin
;
uno: ‘A’ ‘B’ {printf(“uno”);} ‘C’ ‘D’
;
dos: ‘A’ ‘B’                          ‘C’ ‘E’
;
fin:  ‘F’  {    ¿Qué $x es ‘B’?  }
                                  ¿Solución?
```

Añadiendo semántica (.l)

- Objetivo: evaluar suma y resta de enteros y reales

ejemplo6.l (~\Escritorio\unizar\docencia\2005_2006\compi\ejemplosBison) - GVIM

Archivo Editar Herramientas Sintaxis Buffers Ventana Ayuda

```
 1 %{
 2 #include <stdio.h>
 3 #include "y.tab.h"
 4 #define tpINT 10
 5 #define tpREAL 11
 6 %}
 7
 8 digito         [0-9]
 9 constEntera    {digito}+
10 constReal      {digito}+"."{digito}*
11
12 %%
13
14 {constEntera}    {sscanf(yytext,"%d", &yylval.tpOperando.valorI); yylval.tpOperando.elTipo = tpINT;
15                   return(NUMERO);}
16 {constReal}      {sscanf(yytext,"%f", &yylval.tpOperando.valorF); yylval.tpOperando.elTipo = tpREAL;
17                   return(NUMERO);}
18 [+-=]            {return(yytext[0]);}
19 .                {return(OTRO);}
20
21 %%
22
                                                                    1,1           Todo
```

Añadiendo semántica (.y)

```
1 %{
2   #define tpINT  10
3   #define tpREAL 11
4   #define SUMA   12
5   #define RESTA  13
6
7   #include <stdio.h>
8   #include "y.tab.h"
9 %}
10
11 %union{
12   struct {
13     float valorF;
14     int valorI;
15     int elTipo;
16   }tpOperando;
17   int tpOperacion;
18 }
19
20 %token NUMERO OTRO
21 %type <tpOperando> NUMERO
22 %type <tpOperando> operando1
23 %type <tpOperando> operando2
24 %type <tpOperacion> operador
25 %start expresion
26
27 %%
28
29 expresion:
30           operando1
31           operador
32           operando2
33 ;
34
35 operando1: NUMERO
36       { $$=$<tpOperando>1; }
37 ;
38
39 operador: '+' { $$ = SUMA; }
40 |         '-' { $$ = RESTA; }
```

```
42 operando2: NUMERO
43       {
44
45         if ($<tpOperando>-1.elTipo==$<tpOperando>1.elTipo) {
46           if ($<tpOperacion>0==SUMA) {
47             if ($<tpOperando>-1.elTipo==tpINT) {
48               printf("+ ENTERA: %d\n", ($<tpOperando>-1.valorI + $<tpOperando>1.valorI));
49             } else {
50               printf("+ REALES: %f\n", ($<tpOperando>-1.valorF + $<tpOperando>1.valorF));
51             }
52           } else {
53             if ($<tpOperando>-1.elTipo==tpINT) {
54               printf("- ENTERA: %d\n", ($<tpOperando>-1.valorI - $<tpOperando>1.valorI));
55             } else {
56               printf("- REALES: %f\n", ($<tpOperando>-1.valorF - $<tpOperando>1.valorF));
57             }
58           }
59         } else {
60           printf("Deben coincidir los tipos de los operandos\n");
61         }
62       }
63 ;
64
65 %%
66
67 int yyerror(char *m) {
68   fprintf(stderr,"Error:%s\n", m);
69   return(1);
70 }
71
72 int main()
73 {
74   yyparse();
75   return(0);
76 }
```

Para saber más

- Consultar la página web de las asignaturas:
  - Compiladores I:

    http://webdiis.unizar.es/~ezpeleta/COMPI/compiladoresI.htm
  - Lenguajes, Gramáticas y Autómatas:

    http://webdiis.unizar.es/asignaturas/LGA/
- En ellas aparecen enlaces a documentación de utilidad (manuales online, apuntes, etc)

Preguntas