

# Παρουσίαση εργαλείου Bison

& Μέρη B-1, B-2 της εργασίας εξαμήνου

# Εισαγωγή στο εργαλείο bison

- ▶ Μια γεννήτρια συντακτικών αναλυτών
- ▶ Δέχεται την περιγραφή μιας γραμματικής χωρίς συμφραζόμενα (context-free)
- ▶ Παράγει έναν συντακτικό αναλυτή γραμμένο σε C
- ▶ Το bison αποτελεί μια βελτιωμένη έκδοση του yacc (yet another compiler-compiler)

# Δομή προγράμματος bison

%{

Πρόλογος (εισαγωγικό τμήμα)

%}

Δηλώσεις bison (τμήμα δηλώσεων)

%%

Γραμματικοί κανόνες

(κανόνες παραγωγής γραμματικής)

%%

Επίλογος (κώδικας/συναρτήσεις C)

# Πρόλογος (εισαγωγικό τμήμα)

- ▶ Μπορεί να έχει δηλώσεις μακροεντολών, συναρτήσεων και μεταβλητών
- ▶ Ότι περιέχει αντιγράφεται χωρίς αλλαγές στην αρχή του παραγόμενου αρχείου .c
- ▶ Είναι προαιρετικό καθώς τα διαχωριστικά %`{` και %`}` μπορούν να αφαιρεθούν

# Δηλώσεις bison

- ▶ Σε αυτό το τμήμα δηλώνονται τα σύμβολα της γραμματικής και κάποια χαρακτηριστικά τους
  - Δήλωση τερματικών και μη τερματικών συμβόλων
  - Δήλωση αρχικού συμβόλου
  - Καθορισμός προτεραιότητας
- ▶ Επίσης δηλώνονται κάποιες παράμετροι που επηρεάζουν τον συντακτικό αναλυτή
  - Κυρίως σε σχέση με τα ονόματα των παραγόμενων αρχείων και των προσφερόμενων συναρτήσεων

# Δηλώσεις bison – Γραμματική

- ▶ `%token TOKEN`
  - Ορίζει το τερματικό σύμβολο `TOKEN`
- ▶ `%start symbol`
  - Ορίζει το αρχικό σύμβολο της γραμματικής
    - Αν δεν χρησιμοποιηθεί η `%start`, αρχικό σύμβολο θεωρείται το πρώτο μη τερματικό σύμβολο που εμφανίζεται στο τμήμα της περιγραφής της γραμματικής
- ▶ `%union`
  - Ορίζει τους τύπους που μπορούν να πάρουν τα σύμβολα (τερματικά και μη)

# Δηλώσεις bison – Γραμματική

- ▶ `%token <intVal> TOKEN`
  - Ορίζει το τερματικό σύμβολο `TOKEN` με τύπο αυτό που προκύπτει από το `union`
- ▶ `%token <intVal> expr`
  - Ορίζει το μη τερματικό σύμβολο `expr` με τύπο αυτό που προκύπτει από το `union`

# Δηλώσεις bison – Γραμματική

- ▶ `%destructor { code } symbols`
  - Ορίζει ένα τμήμα κώδικα που εκτελείται για τα δοθέντα σύμβολα όταν αυτά σταματήσουν να χρησιμοποιούνται
    - `%union { char *string; }`
    - `%token <string> STRING`
    - `%destructor { free($$); } STRING`
- ▶ `%expect n`
  - Δηλώνει ότι αναμένουμε η γραμματική μας να έχει `n` conflicts



# Δηλώσεις bison – Γραμματική

## ▶ Προτεραιότητες

### ◦ %left, %right

- Ορίζουν την προτεραιότητα στα token που ακολουθούν στη γραμμή με αυξανόμενη προτεραιότητα από πάνω προς τα κάτω
- Τα tokens που εμφανίζονται στην ίδια γραμμή έχουν την ίδια προτεραιότητα

## ▶ %left ADD, SUB

- Το left σημαίνει ότι έχουμε αριστερή προσεταιριστικότητα

- $1 + 2 - 3 \quad \Rightarrow \quad (1 + 2) - 3$

# Δηλώσεις bison – Γραμματική

- ▶ %left MUL, DIV
- ▶ %right EXP, EQ
  - Το right σημαίνει ότι έχουμε δεξιά προσηταιριστικότητα
    - $2^3^4 \quad \Rightarrow \quad 2^{(3^4)}$
- ▶ Σύμφωνα με όλα τα παραπάνω:
  - $1 + 2 * 3^4^5 - 6$ 
    - $(1 + (2 * (3 ^ (4 ^ 5)))) - 6$

# Δηλώσεις bison – Γραμματική

## ▶ Προτεραιότητες

- %nonassoc EQ, UMINUS

- Χρησιμοποιείται για τελεστές που δεν μπορούν να συνδυαστούν μεταξύ τους

## ▶ %prec UMINUS

- Το %prec αλλάζει μια ήδη δηλωμένη προτεραιότητα μέσα σε έναν κανόνα

- Ar\_expr: ‘-’ ar\_expr %prec UMINUS;

# Δηλώσεις bison – Παράμετροι

- ▶ **%defines**
  - Παράγει ένα header file με τις δηλώσεις μακροεντολών για τα σύμβολα της γραμματικής
  - Αν το παραγόμενο αρχείο του συντακτικού αναλυτή είναι το `parser.c`, τότε το header file θα έχει όνομα `parser.h`
- ▶ **%output-file**
  - Ορίζει το όνομα του παραγόμενου αρχείου που θα περιέχει τον κώδικα του συντακτικού αναλυτή
- ▶ **%error-verbose**
  - Χρησιμοποιείται για να πάρουμε πιο αναλυτικά μηνύματα λάθους στην κλήση της `yerror`

# Γραμματικοί κανόνες (κανόνες παραγωγής)

- ▶ Η περιγραφή της γραμματικής της γλώσσας γίνεται με κανόνες παραγωγής διατυπωμένους σε BNF μορφή
- ▶ Γενική μορφή κανόνων
  - Αριστερό\_μέλος: δεξιό\_μέλος;
    - Το αριστερό μέλος είναι ένα μη τερματικό σύμβολο
    - Το δεξιό μέλος μπορεί να περιέχει μηδέν ή περισσότερα τερματικά και μη τερματικά σύμβολα
    - Τα τερματικά σύμβολα (tokens) παριστάνονται με κεφαλαία ενώ τα μη τερματικά με πεζά κατά σύμβαση
    - Μπορούν να δίνονται περισσότερα εναλλακτικά δεξιά μέλη χρησιμοποιώντας το | ως διαχωριστή

# Επίλογος (κώδικας/συναρτήσεις C)

- ▶ Περιέχει συναρτήσεις που χρησιμοποιούνται από τον παραγόμενο συντακτικό αναλυτή
- ▶ Πχ. Συνάρτηση `main`, συνάρτηση `yerror`
- ▶ Ότι προστίθεται σε αυτό το τμήμα αντιγράφεται χωρίς αλλαγές στο τέλος του παραγόμενου αρχείου `.c`
- ▶ Το τμήμα αυτό είναι προαιρετικό

# Περισσότερες πληροφορίες

- ▶ Επίσημη σελίδα του εργαλείου
  - <https://www.gnu.org/software/bison/>
- ▶ Επίσημες οδηγίες χρήσης του εργαλείου
  - <https://www.gnu.org/software/bison/manual/>

# Παραδείγματα

- ▶ 3-ΕΡΓΑΣΤΗΡΙΟ » 3-BISON » Παραδείγματα » 0-Απλοί αυτόνομοι συντακτικοί αναλυτές (χωρίς συνεργασία με ΛΑ)
  - **2 autonomous parsers.zip**
    - unzip 2\ autonomous\ parsers.zip [Linux]



# 0-9digits.y

- ▶ Ο κώδικας στο αρχείο αυτό επιτρέπει τη δημιουργία συντακτικού αναλυτή που
  - (Μέσω Λ.Α. / συνάρτηση yylex) Αναγνωρίζει ένα αριθμητικό ψηφίο (0-9)
  - (Μέσω Λ.Α. / συνάρτηση yylex) Αναγνωρίζει τον χαρακτήρα νέας γραμμής \n
  - (Μέσω Λ.Α. / συνάρτηση yylex) Για οποιονδήποτε άλλο χαρακτήρα ο Λ.Α. τυπώνει invalid character
  - Για οποιαδήποτε γραμμή εισόδου που περιλαμβάνει αριθμητικό ψηφίο και \n ο Σ.Α. τυπώνει Digit: και το ψηφίο αυτό
  - Αν η γραμμή δεν περιλαμβάνει αριθμητικό ψηφίο και \n ο Σ.Α. τυπώνει syntax error

# 0-9digits.y – Πρόλογος & Δηλώσεις bison

## Πρόλογος

```
%{  
    #include <stdio.h>  
    int yylex(void);  
    void yyerror(char *);  
%}
```

## Δηλώσεις bison

```
%token DIGIT NEWLINE
```

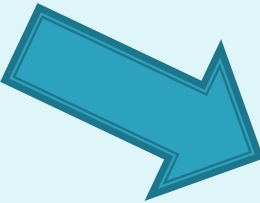
# 0-9digits.y – Γραμματικοί κανόνες

```
program:
    program expr NEWLINE { printf("Digit: %d\n", $2); }
    |
    ;
expr:
    DIGIT          { $$ = $1; }
    ;
```

# 0-9digits.y – Επίλογος (κώδικας/συναρτήσεις C)

- ▶ Η συνάρτηση yylex

```
yylex() {  
    char c;  
    c = getchar();  
  
    // Process all digits  
    if (c >= '0' && c <= '9')  
    {  
        yylval = c - '0';  
        return DIGIT;  
    }  
    if (c == '\n') return NEWLINE;  
    yyerror("invalid character");  
}
```



ASCII Table

Decimal	Hex	Char
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9

# 0-9digits.y – Επίλογος (κώδικας/συναρτήσεις C)

- ▶ Οι συναρτήσεις yyerror και main

```
void yyerror(char *s) {  
    fprintf(stderr, "%s\n", s);  
}  
  
int main(void) {  
    yparse();  
    return 0;  
}
```

# Οδηγίες εκτέλεσης

- ▶ Δημιουργία αρχείου εξόδου με τον κώδικα της συντακτικής ανάλυσης σε γλώσσα C
  - `bison -o 0-9digits.c 0-9digits.y`
- ▶ Δημιουργία εκτελέσιμου κώδικα του συντακτικού αναλυτή
  - `gcc -o 0-9digits 0-9digits.c`
- ▶ Εκτέλεση (αν όλα πάνε καλά) του ΣΑ
  - `./0-9digits` [Linux]
  - `0-9digits` [Windows]

# Εκτέλεση παραδείγματος

- ▶ ./0-9digits [ή 0-9digits αν χρησιμοποιείτε Windows!]

- 1
- Digit: 1
- 9
- Digit: 9
- 25
- syntax error

## Κανόνες παραγωγής

```
program:
    program expr NEWLINE { printf("Digit: %d\n", $2); }
    |
    ;
expr:
    DIGIT { $$ = $1; }
    ;
```

- ▶ ./0-9digits [ή 0-9digits]

- +
- invalid character
- syntax error

## Κώδικας λεκτικού αναλυτή

```
char c;
c = getchar();

// Process all digits
if (c >= '0' && c <= '9')
{
    yylval = c - '0';
    return DIGIT;
}
if (c == '\n') return NEWLINE;
if (c == EOF) return 0;
yyerror("invalid character");
```

# Debugging

- ▶ Προσθήκη `#define YYDEBUG 1` στον πρόλογο

```
%{  
    #include <stdio.h>  
    int yylex(void);  
    void yyerror(char *);  
    #define YYDEBUG 1  
%}
```

- ▶ Προσθήκη `yydebug=1` στον κώδικα της `main`

```
int main(void) {  
    yydebug=1;  
    yyparse();  
    return 0;  
}
```



# Debugging

- ▶ `bison -o 0-9digits.c 0-9digits.y --verbose`
- ▶ `gcc -o 0-9digits 0-9digits.c`
- ▶ `cat 0-9digits.output`

## Grammar

```
0 $accept: program $end
1 program: program expr NEWLINE
2       | %empty
3 expr: DIGIT
```

## Terminals, with rules where they appear

```
$end (0) 0
error (256)
DIGIT (258) 3
NEWLINE (259) 1
```

## Nonterminals, with rules where they appear

```
$accept (5)
  on left: 0
program (6)
  on left: 1 2, on right: 0 1
expr (7)
  on left: 3, on right: 1
```

# Debugging – 0–9digits [1]

- ▶ Εκτέλεση του συντακτικού αναλυτή μετά την ενεργοποίηση του debugging
- ▶ ./0–9digits [ή 0–9digits]

Starting parse

Entering **state 0**

Reducing stack by rule 2 (line 16):

→ \$\$ = nterm program ()

Stack now 0

Entering **state 1**

Reading a token: 1

...

## State 0

0 \$accept: . program \$end

\$default reduce using rule 2 (program)

program go to state 1

0: **program**

## RULES

0 \$accept: program \$end

1 program: program expr NEWLINE

2 | %empty

3 expr: DIGIT

## State 1

0 \$accept: program . \$end

1 program: program . expr NEWLINE

\$end shift, and go to state 2

DIGIT shift, and go to state 3

expr go to state 4

# Debugging – 0–9digits [2]

...

Reading a token: 1

Next token is token **DIGIT** ()

Shifting token **DIGIT** ()

Entering **state 3**

Reducing stack by rule 3 (line 19):

\$1 = token **DIGIT** ()

-> \$\$ = nterm **expr** ()

Stack now 0 1

State 3

3 **expr**: **DIGIT** .

\$default reduce using rule 3 (**expr**)

1: **expr**  
0: **program**

Entering **state 4**

Reading a token: Next token is token **NEWLINE** ()

Shifting token **NEWLINE** ()

Entering **state 5**

Reducing stack by rule 1 (line 15):

\$1 = nterm **program** ()

\$2 = nterm **expr** ()

\$3 = token **NEWLINE** ()

Digit: 1

-> \$\$ = nterm **program** ()

Stack now 0

State 4

1 **program**: **program** **expr** . **NEWLINE**

**NEWLINE** shift, and go to state 5

State 5

1 **program**: **program** **expr** **NEWLINE** .

\$default reduce using rule 1 (**program**)

0: **program**

4: **NEWLINE**  
1: **expr**  
0: **program**

# calc-0.y

- ▶ Ο κώδικας στο αρχείο αυτό επιτρέπει τη δημιουργία συντακτικού αναλυτή που καταλαβαίνει απλές αριθμητικές πράξεις (άθροισμα και πολ/σμο) με postfix notation
  - πχ. 5 4 +

# calc-0.y – Πρόλογος & Δηλώσεις bison

```
%{  
    #include <stdio.h>  
    int yylex(void);  
    void yyerror(char *);  
%}  
  
%token INTEGER PLUS MULT NEWLINE
```

# calc-0.γ – Γραμματικοί κανόνες

program:

```
    program expr NEWLINE { printf("%d\n", $2); }  
    |  
    ;
```

expr:

```
    INTEGER      { $$ = $1; }  
    | expr expr PLUS { $$ = $1 + $2; }  
    | expr expr MULT { $$ = $1 * $2; }  
    ;
```

# calc-0.y – Επίλογος (κώδικας/συναρτήσεις C)

## ► Η συνάρτηση yylex

```
yylex() {  
    char num = 0;  
    char c;  
    c = getchar();  
  
    // Ignore spaces and tabs  
    while (c == ' ' || c == '\t') { yylval = 0; c = getchar(); }  
  
    // Process all digits  
    while (c >= '0' && c <= '9')  
    {  
        yylval = (yylval * 10) + (c - '0');  
        num = 1;  
        c = getchar();  
    }  
    if (num) { ungetc(c, stdin); return INTEGER; }  
  
    if (c == '+') return PLUS;  
    if (c == '*') return MULT;  
    if (c == '\n') { yylval = 0; return NEWLINE;}  
    yyerror("invalid character");  
}
```

# calc-0.y – Επίλογος (κώδικας/συναρτήσεις C)

- ▶ Οι συναρτήσεις yyerror και main

```
void yyerror(char *s) {  
    fprintf(stderr, "%s\n", s);  
}  
  
int main(void) {  
    yparse();  
    return 0;  
}
```



# Εκτέλεση calc-0

- ▶ `bison -o calc-0.c calc-0.y`
- ▶ `gcc -o calc-0 calc-0.c`
- ▶ `./calc-0` [ή `calc-0` στα Windows]
  - `2 3 +`  $(2+3)$ 
    - 5
  - `3 4 5 + *`  $3 * (4+5)$ 
    - 27
  - `5+4`
    - syntax error
  - `$`
    - invalid character
    - syntax error

Τι θα συμβεί αν  
φτάσουμε στο EOF  
(πατήσουμε Control+D  
[Linux] ή Control+Z [Win])

???

# Εκτέλεση calc-0

► 2 3 +

◦ 2

• Λ.Α.: INTEGER

• Bison: expr: INTEGER

• Stack: program =shift=> program INTEGER =reduce=> program expr

◦ 3

• Λ.Α.: INTEGER

• Bison: expr: INTEGER

• Stack: program expr =shift=> program expr INTEGER =reduce=> program expr expr

◦ +

• Λ.Α.: PLUS

• Bison: expr: expr expr PLUS

• Stack: program expr expr =shift=> program expr expr PLUS =reduce=> program expr

◦ \n

• Λ.Α.: NEWLINE

• Bison: program: program expr NEWLINE

• Stack: program expr =shift=> program expr NEWLINE =reduce=> program

program:

```
program expr NEWLINE { printf("%d\n", $2); }
```

```
|  
;
```

expr:

```
INTEGER      { $$ = $1; }
```

```
| expr expr PLUS { $$ = $1 + $2; }
```

```
| expr expr MULT { $$ = $1 * $2; }
```

```
;
```

# Εργασία B-1

- ▶ Εργασία Μεταγλωττιστών
  - Μέρος B-1
    - Δοκιμή του πρότυπου κώδικα Bison
      - `simple-bison-code.zip`

• ΧΩΡΙΣ ΥΠΟΒΟΛΗ ΤΟ ΜΕΡΟΣ B-1!

# Σύνδεση εργαλείου Flex με Bison



# 1<sup>ο</sup> παραδείγμα

- ▶ 3-ΕΡΓΑΣΤΗΡΙΟ » 3-BISON » Παραδείγματα »  
1-Απλοί συντακτικοί αναλυτές  
(συνεργασία με ΛΑ)
  - 1-intro-calc.zip
    - Εισαγωγικός ΣΑ (πρόσθεση – πολ/σμος δύο ακεραίων) σε συνεργασία με flex
    - unzip 1-intro-calc.zip
      - calc-0\_with\_flex.l
      - calc-0\_with\_flex.y
      - Makefile

# calc-0\_with\_flex.l – Ορισμοί

- ▶ Με %option noyywrap αποφεύγουμε τη συγγραφή της συνάρτησης yywrap().

```
%option noyywrap  
%x error
```

- ▶ Με %x error δηλώνουμε την αποκλειστική (eXclusive) start condition με όνομα error
  - Με BEGIN(error) μεταβαίνουμε στην κατάσταση error και εκτελούνται μονάχα οι κανόνες που ξεκινάνε με <error>

# calc-0\_with\_flex.l – Ορισμοί

- ▶ Πέρα από τις «κλασικές» βιβλιοθήκες `stdio.h`, `string.h` και `stdlib.h` υπάρχει και η αναφορά στην `calc-0_with_flex.tab.h`
- ▶ Αυτή παράγεται από τον `bison`

```
#include <stdio.h>  
#include <string.h>  
#include <stdlib.h>
```

```
#include "calc-0_with_flex.tab.h"
```

# calc-0\_with\_flex.l – Ορισμοί

- ▶ Ορισμός συναρτήσεων yyerror & prn

```
int line=1;  
  
void yyerror (const char *msg);  
void prn(char *s);
```

- ▶ Ορισμός DELIMITER & INTEGER

DELIMITERS	[ \t]+
INTEGER	[0-9]+



# calc-0\_with\_flex.l – Κανόνες

- ▶ Κανόνες για την λεκτική ανάλυση

```
{DELIMITERS}    {}
"+"             {prn("+"); return '+';}
"*"             {prn("*"); return '*'};
\n              {line++; return NEWLINE;}
{INTEGER}        {prn("INTCONST"); return INTEGER;}
.                {yyerror ("Unrecognized token error!");
                  BEGIN(error); return TOKEN_ERROR;}
<error>.         {}
<error>\n        {BEGIN(0);}
```

# calc-0\_with\_flex.l – Κώδικας χρήστη

- ▶ Κώδικας των συναρτήσεων yyerror & prn

```
void yyerror (const char *msg)
{
    fprintf(yyout, "\tFlex -> ERROR, line %d at lexeme '%s' :
%s\n",line, yytext, msg);
}
```

```
void prn(char *s)
{
    printf("\tFlex -> Matched token: %s\n", yytext);
}
```

# calc-0\_with\_flex.y – Πρόλογος & δηλώσεις bison

- ▶ Με την YYSTYPE ορίζουμε τον τύπο δεδομένων

```
#include <stdio.h>
#include <string.h>
int line;
int flag=0;
extern char * yytext;
#define YYSTYPE int

%token INTEGER NEWLINE TOKEN_ERROR
%start program
```

# calc-0\_with\_flex.y – Γραμματικοί κανόνες

- ▶ Κανόνες παραγωγής για τον συντακτικό αναλυτή

```
program:
    program expr NEWLINE { printf("%d\n", $2); }
    |
    ;
expr:
    INTEGER      { $$ = atoi(yytext); }
    | expr expr '+' { $$ = $1 + $2; }
    | expr expr '*' { $$ = $1 * $2; }
    ;
```

# calc-0\_with\_flex.y – Επίλογος

- ▶ Η `yyparse()` επιστρέφει 0 αν η συντακτική ανάλυση ολοκληρώθηκε με επιτυχία

```
int main(int argc, char **argv)
{
    int i;

    int ret = yyparse();

    if (flag==0 && ret==0)
        printf("\t\tBison -> PARSING SUCCEEDED.\n");
    else
        printf("\t\tBison -> PARSING FAILED.\n");

    return 0;
}
```

# calc-0\_with\_flex – Οδηγίες εκτέλεσης

- ▶ Για τη δημιουργία του κώδικα της ΣΑ:
  - `bison -d calc-0_with_flex.y`
    - `calc-0_with_flex.tab.h`
    - `calc-0_with_flex.tab.c`
- ▶ Για τη δημιουργία του κώδικα της ΛΑ:
  - `flex calc-0_with_flex.l`
    - `lex.yy.c`
- ▶ Για τη συνένωση των κωδικων ΣΑ & ΛΑ και τη δημιουργία του εκτελέσιμου:
  - `gcc -o calc-0_with_flex calc-0_with_flex.tab.c lex.yy.c`
    - `calc-0_with_flex`

# 2<sup>ο</sup> παραδείγμα

- ▶ 3-ΕΡΓΑΣΤΗΡΙΟ » 3-BISON » Παραδείγματα »  
1-Απλοί συντακτικοί αναλυτές  
(συνεργασία με ΛΑ)
  - 2-calc\_full
    - Πλήρης calculator calc\_full.y, calc\_full.l και σημειώσεις
    - unzip calc\_full.zip
      - calc\_full.l
      - calc\_full.y
      - Makefile
      - calc\_full Notes.txt

# calc\_full.l – Ορισμοί

```
%{  
    #define YYSTYPE double  
    #include "calc_full.tab.h"  
    #include <stdlib.h>  
%}  
  
digit          [0-9]  
integer        {digit}+  
exponent       [eE][+-]?{integer}
```



# calc\_full.l – Κανόνες

```
[ \t]+      { }  
{integer}("."{integer})?{exponent}? { yylval=atof(yytext);  
                                         return NUMBER;  
                                         }
```

```
"+"      return PLUS;  
"-"      return MINUS;  
"*"      return TIMES;  
"/"      return DIVIDE;  
"^"      return POWER;  
"("      return LEFT;  
")"      return RIGHT;  
"\n"     return END;
```

# calc\_full.y – Πρόλογος & δηλώσεις bison

```
%{  
    #include <math.h>  
    #include <stdio.h>  
    #include <stdlib.h>  
    #define YYSTYPE double  
%}  
  
%token NUMBER  
%token PLUS MINUS TIMES DIVIDE POWER  
%token LEFT RIGHT  
%token END  
  
%left PLUS MINUS  
%left TIMES DIVIDE  
%left NEG  
%right POWER  
  
%start Input
```

# Προτεραιότητες στα token (1)

## ▶ Προτεραιότητες

### ◦ %left, %right

- Ορίζουν την προτεραιότητα στα token που ακολουθούν στη γραμμή με αυξανόμενη προτεραιότητα από πάνω προς τα κάτω
- Τα tokens που εμφανίζονται στην ίδια γραμμή έχουν την ίδια προτεραιότητα

## ▶ %left PLUS, MINUS

- Το left σημαίνει ότι έχουμε αριστερή προσημαιστικότητα

- $4-3+1 \quad \Rightarrow \quad (4-3)+1$

# Προτεραιότητες στα token (2)

- ▶ %left TIMES, DIVIDE
- ▶ %left NEG
- ▶ %right POWER
  - Το right σημαίνει ότι έχουμε δεξιά προσηλωτικότητα
    - $2^2^3 \quad \Rightarrow \quad 2^{(2^3)}$

# calc\_full.y – Γραμματικοί κανόνες

Input:

| Input Line

;

Line:

END

| Expression END { printf("Result: %f\n", \$1); }

;

Expression:

NUMBER { \$\$=\$1; }

| Expression PLUS Expression { \$\$=\$1+\$3; }

| Expression MINUS Expression { \$\$=\$1-\$3; }

| Expression TIMES Expression { \$\$=\$1\*\$3; }

| Expression DIVIDE Expression { \$\$=\$1/\$3; }

| MINUS Expression **%prec NEG** { \$\$=-\$2; }

| Expression POWER Expression { \$\$=pow(\$1,\$3); }

| LEFT Expression RIGHT { \$\$=\$2; }

;

# Προτεραιότητες στα token (3)

- ▶ %prec NEG
  - Το %prec αλλάζει μια ήδη δηλωμένη προτεραιότητα μέσα σε έναν κανόνα
  - MINUS Expression %prec NEG { \$\$=-\$2; }

# calc\_full.y – Επίλογος

- ▶ Ο κώδικας των συναρτήσεων yyerror & main

```
int yyerror(char *s) {  
    printf("%s\n", s);  
}  
  
int main() {  
    if (yyparse()==0)  
        fprintf(stderr, "Successful parsing.\n");  
    else  
        fprintf(stderr, "error found.\n");  
}
```

# calc\_full – Οδηγίες εκτέλεσης

- ▶ Για τη δημιουργία του κώδικα της ΣΑ:
  - `bison -d calc_full.y`
    - `calc_full.tab.h`
    - `calc_full.tab.c`
- ▶ Για τη δημιουργία του κώδικα της ΛΑ:
  - `flex calc_full.l`
    - `lex.yy.c`
- ▶ Για τη συνένωση των κωδικων ΣΑ & ΛΑ και τη δημιουργία του εκτελέσιμου:
  - `gcc -o calc_full lex.yy.c calc_full.tab.c -lfl -lm`
    - `calc_full`



# 3<sup>ο</sup> παραδείγμα

- ▶ 3-ΕΡΓΑΣΤΗΡΙΟ » 3-BISON » Παραδείγματα »  
1-Απλοί συντακτικοί αναλυτές  
(συνεργασία με ΛΑ)
  - 3-simple-parser
    - Απλός ΣΑ με αρχεία εισόδου-εξόδου και σημειώσεις (δήλωση μεταβλητών, ανάθεση τιμών/μεταβλητών)
    - unzip simple-parser2.zip
      - simple-parser.l
      - simple-parser.y
      - Makefile
      - simple-parser-notes.txt
      - input-fail.txt
      - input-success.txt

# simple-parser.l – Ορισμοί

```
%{  
    #include <stdio.h>  
    #include <string.h>  
    #include <stdlib.h>  
    #include "simple-parser.tab.h"  
  
    extern int flag;  
    extern int line;  
  
    void prn(char *s);  
%}  
  
DELIMITERS      [ \t]+  
ID               [A-Za-z][A-Za-z0-9]*(_[A-Za-z0-9]+)*  
DIGIT_H          [1-9]  
DIGIT_L          [0-9]
```

# simple-parser.l – Κανόνες & κώδικας χρήστη

## ► Κανόνες:

"int"	{prn("INT"); return SINT;}
"."	{prn("SEMI"); return SEMI;}
"="	{prn("ASSIGNOP"); return ASSIGNOP;}
{DELIMITERS}	{ }
\n	{line++; }
0 -?({DIGIT_H}+{DIGIT_L}*)+	{prn("INTCONST"); return INTCONST;}
{ID}	{prn("IDENTIFIER"); return IDENTIFIER;}
.	{printf("Token error\n");}

## ► Συνάρτηση prn:

```
void prn(char *s)
{
    printf("\n\t%s: %s ", s, yytext);
    return;
}
```

# simple-parser.y – Πρόλογος & δηλώσεις bison

```
%{  
    #include <stdio.h>  
    #include <string.h>  
    int line=1;  
    int errflag=0;  
    extern char *yytext;  
    #define YYSTYPE char *  
%}  
  
%token SINT SEMI ASSIGNOP IDENTIFIER INTCONST  
  
%start program
```

# simple-parser.y – Γραμματικοί κανόνες

```
program : program decl
        | program assign
        |

decl    : type aid SEMI { printf("\n\t### Line:%d Declaration\n", line); }
type    : SINT          { $$ = strdup(yytext); }
aid     : IDENTIFIER     { $$ = strdup(yytext); }
tim     : INTCONST       { $$ = "SINT"; }
        | IDENTIFIER     { $$ = strdup(yytext); }
assign  : aid ASSIGNOP tim SEMI
        {
            if (!strcmp($3, "SINT"))
            {
                printf("\n\t### Line:%d Value assignment\n", line);
            }
            else
            {
                printf("\n\t### Line:%d Variable assignment\n", line);
            }
        }
```

# simple-parser.y – Επίλογος

```
int main(int argc, char **argv)
{
    int i;
    if(argc == 2)
        yyin=fopen(argv[1], "r");
    else
        yyin=stdin;

    int parse = yyparse();

    if (errflag==0 && parse==0)
        printf("\nINPUT FILE: PARSING SUCCEEDED.\n", parse);
    else
        printf("\nINPUT FILE: PARSING FAILED.\n", parse);

    return 0;
}
```

# simple-parser – Οδηγίες εκτέλεσης

## ▶ make

- `bison -d simple-parser.y`
- `flex simple-parser.l`
- `gcc simple-parser.tab.c lex.yy.c -o simple-parser`

## ▶ Ανάγνωση των αρχείων εισόδου [Linux]

- `./simple-parser input-success.txt`
  - `./simple-parser < input-success.txt` ← Ανακατεύθυνση
- `./simple-parser input-fail.txt`
  - `./simple-parser < input-fail.txt` ← Ανακατεύθυνση

Υπενθύμιση: Στα Windows η εντολή  
θα είναι `simple-parser` χωρίς `./`

# Εργασία B-2

## ▶ Εργασία Μεταγλωττιστών

### ◦ Μέρος B-2

- Κατάλληλη προσαρμογή κώδικα Flex (μέρος A-3) με κώδικα Bison (μέρος B-1 / simple-bison-code.zip)
- Η λεκτική ανάλυση θα (εξακολουθήσει να) γίνεται από το Flex
- Η συντακτική ανάλυση θα γίνεται από το Bison
  - Αναγνώριση όλων των εκφράσεων της γλώσσας Uni-C

## ▶ Προσοχή στα ζητούμενα!

- Διαβάστε προσεκτικά την εκφώνηση της εργασίας



# Εργασία B-2

- ▶ Υπενθυμίζονται τα παρακάτω:
  - Επαρκή σχόλια στον κώδικα
  - Αναλυτικό έγγραφο τεκμηρίωσης με:
    - Εξαντλητικές δοκιμές για τον συντακτικό αναλυτή
    - Σχολιασμό αποτελεσμάτων
    - Αναφορά στο διαμοιρασμό καθηκόντων / αρμοδιοτήτων
    - Δηλώστε οπωσδήποτε εάν ο κώδικάς σας κάνει compile ή όχι και εάν παράγει σωστά/αποδεκτά αποτελέσματα ή όχι
      - Καθώς και: Warnings, ελλείψεις, προβλήματα κάθε είδους!
  - **Υποβολή ZIP αρχείου στο eClass μαζί με το Μέρος B-3**
    - Κώδικας flex (.l) και κώδικας bison (.y) [+έγγραφο τεκμηρίωσης!]
    - Να υπάρχουν όλα τα απαραίτητα συνοδευτικά αρχεία (Makefile, input.txt, output.txt, κλπ)
    - Το όνομα του ZIP να ξεκινάει με τον αριθμό της ομάδας
    - Στοιχεία όλων των μελών στα σχόλια της υποβολής (eClass)