Week 6

Exercise 42-3

../42–3/parser/grammar

```

%filenames parser
%scanner ../scanner/scanner.h

%token WRITE
%token IDENT
%token NUMBER

%%
input:
      //empty
|
    input line
;

line:
    '\n'
|
    function '\n'
    {
      std::cout << "\t" << $1 << std::endl;
    }
;


function:
      WRITE
      '('
      variable_list
      ')'
;


variable_list:
    variable_list
    ','
    var
|
    var
;

var:
    IDENT
|
    NUMBER
;
```

../43/parser/grammar

```
//%default-actions quiet
%filenames parser
%scanner ../scanner/scanner.h


%baseclass-preinclude cmath

%token NR
%stype double
%left '-' '+'
%left '*' '/'
%right NEG //unary minus
%right '$' //sqrt

%%

input:
      //empty
|
    input line
;

line:
    '\n'
|
    expr '\n'
    {
      std::cout << "\t" << $1 << std::endl;
    }
;


expr:
    NR
|
    '-' expr %prec NEG  //unary minus
    {
      $$ = -$2;
    }
|
    expr '+' expr
    {
      $$ = $1 + $3;
    }
|
    expr '-' expr
    {
      $$ = $1 - $3;
    }
|
    expr '*' expr
    {
      $$ = $1 * $3;
    }
|
    expr '/' expr
    {
      $$ = $1 / $3;
    }
|
    //sqrt
    '$' expr
    {
      $$ = sqrt($2);
    }
|
    '(' expr ')'
    {
      $$ = $2;
    }
;
```

../45/parser/grammar

```
//%default-actions quiet
%filenames parser
%scanner ../scanner/scanner.h

//%baseclass-preinclude      x.h or <x.h>


//    Semantic values used by the parser.
//    Two often used types are predefined, extend or alter as seems fit.
//    When %union is not used, use:
//%stype   struct-name/class-name
//%union
//{
//      //  define union fields here. The fields shown are for demo-use only
//    int          i;
//    unsigned     u;
//    std::string *s;
//};
//  Typed nonterminals indicate the union-value that's returned:
//%type<i>
//    rule1 or TOKEN
//    rule2

// lowest precedence
//%token
//%nonassoc
//%left
//%right
// highest precedence

%baseclass-preinclude cmath

%token NR
%stype std::size_t
%left '!'
%left '='
%left '+'
%left '*'
%left '^'
%right '-'

%%

input:
      //empty
|
    input line
;

line:
    '\n'
|
    expr '\n'
    {
      std::cout << "\t" << $1 << std::endl;
    }
;

expr:
    NR
|
    '-' expr
    {
      $$ = -$2;
```

```
        }
|
        expr '+' expr
        {
          $$ = $1 + $3;
        }
|
        expr '*' expr
        {
          $$ = $1 * $3;
        }
|
        expr '!' '=' expr
        {
          $$ = ($1 != $4);
        }
|
        expr '=' '=' expr
        {
          $$ = ($1 == $4);
        }
|
        expr '^' expr //wordt dit wel op binary exponent manier gedaan?
        {
          $$ = pow($1, $3);
        }

;
```

../46/parser/grammar

```
//%default-actions quiet
%filenames parser
%scanner ../scanner/scanner.h

%baseclass-preinclude cmath

%token VAR
%token NR
%left '+'
%left '*'
%right '-'

%%

input:
      //empty
|
    input line
;

line:
    '\n'
|
    expr '\n'
    {
      std::cout << "\t" << $1 << std::endl;
    }
;


expr:
    VAR
|
    NR
|
    math
|
    array
;

math:
    '-' expr
    {
      $$ = - $2;
    }
|
    expr '+' expr
    {
      $$ = $1 + $3;
    }
|
    expr '*' expr
    {
      $$ = $1 * $3;
    }
;

array:  //var[idx].. is allowed whereas NR[idx] is not
    array '[' expr ']'
    |
    VAR '[' expr ']'
;
```

../47/parser/grammar

```
//%default-actions quiet
%filenames parser
%scanner ../scanner/scanner.h


%token WORD
%token INT
%token FLOAT

%%



list:
    //empty
|
    entry
|
    comma_list
|
    norm_list
;

comma_list:
    comma_list ',' entry
    |
    entry ',' entry
;

norm_list:
    norm_list ' ' entry
    |
    entry ' ' entry
;

entry:
  ' '
|
  WORD
|
  INT
|
  FLOAT
;
```