Week 6

Exercise 42-3

../42–3/parser/grammar

```
1
2  %filenames parser
3  %scanner ../scanner/scanner.h
4
5  %token WRITE
6  %token IDENT
7  %token NUMBER
8
9  %%
10 input:
11       //empty
12 |
13     input line
14 ;
15
16 line:
17     '\n'
18 |
19     function '\n'
20     {
21       std::cout << "\t" << $1 << std::endl;
22     }
23 ;
24
25
26 function:
27       WRITE
28       '('
29       variable_list
30       ')'
31 ;
32
33
34 variable_list:
35     variable_list
36     ','
37     var
38 |
39     var
40 ;
41
42 var:
43     IDENT
44 |
45     NUMBER
46 ;
```

../43/parser/grammar

```
1  //%default-actions quiet
2  %filenames parser
3  %scanner ../scanner/scanner.h
4
5
6  %baseclass-preinclude cmath
7
8  %token NR
9  %stype double
10 %left '-' '+'
11 %left '*' '/'
12 %right NEG //unary minus
13 %right '$' //sqrt
14
15 %%
16
17 input:
18       //empty
19 |
20     input line
21 ;
22
23 line:
24     '\n'
25 |
26     expr '\n'
27     {
28       std::cout << "\t" << $1 << std::endl;
29     }
30 ;
31
32
33 expr:
34     NR
35 |
36     '-' expr %prec NEG  //unary minus
37     {
38       $$ = -$2;
39     }
40 |
41     expr '+' expr
42     {
43       $$ = $1 + $3;
44     }
45 |
46     expr '-' expr
47     {
48       $$ = $1 - $3;
49     }
50 |
51     expr '*' expr
52     {
53       $$ = $1 * $3;
54     }
55 |
56     expr '/' expr
57     {
58       $$ = $1 / $3;
59     }
60 |
61     //sqrt
62     '$' expr
63     {
64       $$ = sqrt($2);
```

```
65      }
66  |
67      '(' expr ')'
68      {
69        $$ = $2;
70      }
71  ;
```

../45/parser/grammar

```
1  //%default-actions quiet
2  %filenames parser
3  %scanner ../scanner/scanner.h
4
5  //%baseclass-preinclude      x.h or <x.h>
6
7
8  //    Semantic values used by the parser.
9  //    Two often used types are predefined, extend or alter as seems fit.
10 //    When %union is not used, use:
11 //%stype   struct-name/class-name
12 //%union
13 //{
14 //      //  define union fields here. The fields shown are for demo-use only
15 //    int           i;
16 //    unsigned      u;
17 //    std::string *s;
18 //};
19 //  Typed nonterminals indicate the union-value that's returned:
20 //%type<i>
21 //    rule1 or TOKEN
22 //    rule2
23
24 // lowest precedence
25 //%token
26 //%nonassoc
27 //%left
28 //%right
29 // highest precedence
30
31 %baseclass-preinclude cmath
32
33 %token NR
34 %stype std::size_t
35 %left '!'
36 %left '='
37 %left '+'
38 %left '*'
39 %left '^'
40 %right '-'
41
42 %%
43
44 input:
45       //empty
46 |
47     input line
48 ;
49
50 line:
51     '\n'
52 |
53     expr '\n'
54     {
55       std::cout << "\t" << $1 << std::endl;
56     }
57 ;
58
59 expr:
60     NR
61 |
62     '-' expr
63     {
64       $$ = -$2;
```

```
65      }
66  |
67      expr '+' expr
68      {
69        $$ = $1 + $3;
70      }
71  |
72      expr '*' expr
73      {
74        $$ = $1 * $3;
75      }
76  |
77      expr '!' '=' expr
78      {
79        $$ = ($1 != $4);
80      }
81  |
82      expr '=' '=' expr
83      {
84        $$ = ($1 == $4);
85      }
86  |
87      expr '^' expr //wordt dit wel op binary exponent manier gedaan?
88      {
89        $$ = pow($1, $3);
90      }
91
92  ;
```

../46/parser/grammar

```
1  //%default-actions quiet
2  %filenames parser
3  %scanner ../scanner/scanner.h
4
5  %baseclass-preinclude cmath
6
7  %token VAR
8  %token NR
9  %left '+'
10 %left '*'
11 %right '-'
12
13 %%
14
15 input:
16       //empty
17 |
18     input line
19 ;
20
21 line:
22     '\n'
23 |
24     expr '\n'
25     {
26       std::cout << "\t" << $1 << std::endl;
27     }
28 ;
29
30
31 expr:
32     VAR
33 |
34     NR
35 |
36     math
37 |
38     array
39 ;
40
41 math:
42     '-' expr
43     {
44       $$ = - $2;
45     }
46 |
47     expr '+' expr
48     {
49       $$ = $1 + $3;
50     }
51 |
52     expr '*' expr
53     {
54       $$ = $1 * $3;
55     }
56 ;
57
58 array:  //var[idx].. is allowed whereas NR[idx] is not
59     array '[' expr ']'
60     |
61     VAR '[' expr ']'
62 ;
```

../47/parser/grammar

```
1  //%default-actions quiet
2  %filenames parser
3  %scanner ../scanner/scanner.h
4
5
6  %token WORD
7  %token INT
8  %token FLOAT
9
10 %%
11
12
13
14 list:
15     //empty
16 |
17     entry
18 |
19     comma_list
20 |
21     norm_list
22 ;
23
24 comma_list:
25     comma_list ',' entry
26     |
27     entry ',' entry
28 ;
29
30 norm_list:
31     norm_list ' ' entry
32     |
33     entry ' ' entry
34 ;
35
36 entry:
37   ' '
38 |
39   WORD
40 |
41   INT
42 |
43   FLOAT
44 ;
```