ÖREBRO UNIVERSITY

COMPILERS AND INTERPRETERS

Assignment 5

Authors:

Linus Baumgärtner lbaumgaertner@culba.de

Marko Reichhart marko.reichhart@web.de

Part A

In listing 1 the grammar for a list of statements can be seen. The previous grammar was extended by the non-terminals stmt and $stmt_list$. The functions that were suggested in the lab were used to create the syntax tree. To connect the statements a semicolon node is introduced.

```
Listing 1: Grammar for list of statements
   start:
       list DONE
2
                            { print_syntax_tree($1); }
3
   list:
4
         stmt list
                            { $$ = mknode(';', $1, $2); }
5
                            { $$ = mknode(' '); }
6
   stmt:
                                                         { $$ = mknode(';'); }
       | expr ';'
                                                         { $$ = $1; }
10
       | ID '=' expr ';'
                                                         { $$ = mknode('=', mkleaf(ID, $1), $3); }
       | PRINT '(' ID ')' ';'
                                                         { $$ = mknode(PRINT, mkleaf(ID, $3)); }
       | READ '(' ID ')' ';'
                                                         { $$ = mknode(READ, mkleaf(ID, $3)); }
13
       | WHILE '(' expr ')' stmt
                                                         { $$ = mknode(WHILE, $3, $5); }
14
       | IF '(' expr ')' stmt %prec IFX
                                                         { $$ = mknode(IF, $3, $5); }
15
       | IF '(' expr ')' stmt ELSE stmt
                                                         { $$ = mknode(IF, $3, $5, $7); }
16
       | '{' stmt_list '}'
| '{' '}'
                                                         { $$ = $2; }
17
                                                         { $$ = mknode(' ');}
18
19
   stmt_list:
20
                            { $$ = $1; }
21
                            { $$ = mknode(';', $1, $2); }
        | stmt_list stmt
22
23
24
   expr:
         NUM
                                  { $$ = mkleaf(NUM, $1); }
25
                                  { $$ = mkleaf(ID, $1); }
       | ID
26
                                  { $$ = mknode('+', $1, $3); }
{ $$ = mknode('-', $1, $3); }
       | expr '+' expr
27
       | expr '-' expr
28
                                  { $$ = mknode('*', $1, $3); }
       | expr '*' expr
29
                                 { $$ = mknode('/', $1, $3); }
       | expr '/' expr
30
                                  { $$ = mknode('%', $1, $3); }
       | expr '%' expr
31
       | expr '^' expr
                                 { $$ = mknode('^', $1, $3); }
                                 { $$ = mknode('&', $1, $3); }
        | expr '&' expr
        | expr '|' expr
                                 { $$ = mknode('|', $1, $3); }
        | expr '<' expr
                                 { $$ = mknode('<', $1, $3); }
35
        | expr '>' expr
                                 { $$ = mknode('>', $1, $3); }
36
        | '(' expr ')'
                                  { $$ = $2; }
37
        | expr '?' expr ':' expr { $$ = mknode('?', $1, $3, $5); }
38
39
40
  %%
41
```

Part B

To print the tree, the recursive printout function from the lab was used and extended by all occurring cases and can be seen in listing 2. The function is called when the program is aborted by ctrl+d.

```
Listing 2: Syntax tree printout function

void print_syntax_tree(TreeNode* p) {
 printf("\nSyntax tree:\n");
 printtree1(p, 0);
}
```

```
5
        void printtree1(TreeNode* p, int level) {
  6
 7
                   if (p==0)
                             return;
 8
                   printf("%*s", level, "");
 9
                   switch (p->type) {
10
                             case ' ':
11
                                       printf("{}\n");
12
                                       break:
13
                             case ID:
14
                                       printf("%s\n", symtable[p->leaf_value].lexeme);
15
16
                             case NUM:
17
                                       printf("%d\n", p->leaf_value);
18
                                       break;
19
                             case '=':
20
                                       printf("=\n");
21
                                       printtree1(p->args[0], level + 4);
22
                                       printtree1(p->args[1], level + 4);
23
                                       break:
24
                             case IF:
25
                                       printf("if\n");
26
                                       printtree1(p->args[0], level + 4);
27
28
                                       printtree1(p->args[1], level + 4);
                                       printtree1(p->args[2], level + 4);
29
30
                                       break;
                             case WHILE:
31
                                       printf("while\n");
32
                                       printtree1(p->args[0], level + 4);
33
                                       printtree1(p->args[1], level + 4);
34
                                       break;
35
                             case PRINT:
36
                                       printf("print\n");
37
                                       printtree1(p->args[0], level + 4);
38
                                       break;
39
                             case READ:
40
                                       printf("read\n");
41
42
                                       printtree1(p->args[0], level + 4);
                                       break;
43
                             case '+': case '-': case '*': case '/': case '%': case '\alpha': c
44
                                       printf("%c\n", p->type);
45
                                       printtree1(p->args[0], level + 4);
46
                                       printtree1(p->args[1], level + 4);
47
                                       break;
48
                             case '?':
49
                                       printf("%c\n", p->type);
50
                                       printtree1(p->args[0], level + 4);
51
                                       printtree1(p->args[1], level + 4);
52
                                       printtree1(p->args[2], level + 4);
53
                                       break:
54
                             case ';':
55
                                       printf(";\n");
56
                                       printtree1(p->args[0], level + 4);
57
                                       if (p->args[1] != 0) {
58
                                                 printtree1(p->args[1], level + 4);
59
60
                                       break;
61
62
                   }
63 }
```

Part C

The implementation of the statements if, while, print and read can be seen in listing 1 and listing 2. They are inserted into the grammar as an option for the non-terminal stmt.