

- ☐ Gr. 1, Dr. H. Dobler  
☐ Gr. 2, Dr. G. Kronberger

Name \_\_\_\_\_ Aufwand in h \_\_\_\_\_

Punkte \_\_\_\_\_ Übungsleiter \_\_\_\_\_

Im Moodle-Kurs finden Sie unter Flex und Bison zwei ZIP-Dateien mit den GNU-Implementierungen von lex u. yacc, für Windows. Bei UNIX sind beide Werkzeuge üblicherweise bereits enthalten oder können leicht nachgeladen werden.

**1. MiniC: Scanner und Parser mit lex und yacc****(8 Punkte)**

MiniC ist eine kleine Teilmenge von C, angelehnt an MiniPascal. Unten links ist ein einfaches Programm zur Berechnung des Satzes von Pythagoras dargestellt, rechts die Grammatik von MiniC (die Sie auch im Moodle-Kurs in der Datei *MiniC.syn* finden):

```
void main() {
    int a, b, cs;
    scanf(a);
    scanf(b);
    cs = (a * a) + (b * b);
    printf(cs);
}
```

```
MC =      "void" "main" "(" ")" "{"
        [ VarDecl ]
        StatSeq
        "}" .
VarDecl = "int" ident { "," ident } ";" .
StatSeq = Stat { Stat } .
Stat =    [ ident "=" Expr
          | "scanf" "(" ident ")"
          | "printf" "(" Expr ")"
          ] ";" .
Expr =    Term { ( "+" | "-" ) Term } .
Term =    Fact { ( "*" | "/" ) Fact } .
Fact =    ident | number | "(" Expr ")" .
```

Erzeugen Sie mit lex einen lexikalischen Analysator (*scanner*) und mit yacc einen Syntaxanalysator (*parser*) für MiniC und bauen Sie daraus ein Programm für die Analyse von MiniC-Programmen.

**2. MiniCpp: Scanner und Parser mit lex und yacc****(10 + 6 Punkte)**

Wir werden uns intensiver mit der etwas größeren Sprache MiniCpp beschäftigen, mit der man auch etwas anspruchsvollere Programme schreiben kann, z. B. für das Sieb des Erathostenes:

```
void Sieve(int n); // declaration

void main() {
    int n;
    cout << "n > ";
    cin >> n;
    if (n > 2)
        Sieve(n);
} // main

void Sieve(int n) { // definition
    int col, i, j;
    bool *sieve = 0;
    sieve = new bool[n + 1];
    i = 2;
    while (i <= n) {
        sieve[i] = true;
        i++;
    } // while
```

```
    cout << 2 << " ";
    col = 1;
    i = 3;
    while (i <= n) {
        if (sieve[i]) {
            if (col == 10) {
                cout << endl;
                col = 0;
            } // if
            col++;
            cout << i << " ";
            j = i * i;
            while (j <= n) {
                sieve[j] = false;
                j = j + 2 * i;
            } // while
        } // if
        i = i + 2;
    } // while
    delete[] sieve;
} // Sieve
```

Hier die Grammatik für MiniCpp, die Sie im Moodle-Kurs auch in der Datei *MiniCpp.syn* finden:

```

MiniCpp =      { ConstDecl | VarDef | FuncDecl | FuncDef } .
ConstDecl =    'const' Type ident Init ';' .
Init =         '=' ( false | true | number ) .
VarDef =       Type [ '*' ] ident [ Init ]
               { ',' [ '*' ] ident [ Init ] } ';' .
FuncDecl =     FuncHead ';' .
FuncDef =      FuncHead Block .
FuncHead =     Type [ '*' ] ident '(' [ FormParList ] ')' .
FormParList =  ( 'void' |
               Type [ '*' ] ident [ '[' ']' ]
               { ',' Type [ '*' ] ident [ '[' ']' ] } ) .
Type =         'void' | 'bool' | 'int' .
Block =        '{' { ConstDecl | VarDef | Stat } '}' .
Stat =         ( IncStat | DecStat | AssignStat
               | CallStat | IfStat
               | WhileStat | BreakStat
               | InputStat | OutputStat | DeleteStat | ReturnStat
               | Block
               | ';'
               ) .
IncStat =      ident '++' ';' .
DecStat =      ident '--' ';' .
AssignStat =   ident [ '[' Expr ']' ] '=' Expr ';' .
CallStat =     ident '(' [ ActParList ] ')' ';' .
ActParList =   Expr { ',' Expr } .
IfStat =       'if' '(' Expr ')' Stat [ 'else' Stat ] .
WhileStat =    'while' '(' Expr ')' Stat .
BreakStat =    'break' ';' .
InputStat =    'cin' '>>' ident ';' .
OutputStat =   'cout' '<<' ( Expr | string | 'endl' )
               { '<<' ( Expr | string | 'endl' ) } ';' .
DeleteStat =   'delete' '[' ']' ident ';' .
ReturnStat =   'return' [ Expr ] ';' .
Expr =         OrExpr .
OrExpr =       AndExpr { '||' AndExpr } .
AndExpr =      RelExpr { '&&' RelExpr } .
RelExpr =      SimpleExpr
               [ ( '==' | '!=' | '<' | '<=' | '>' | '>=' )
                 SimpleExpr ] .
SimpleExpr =   [ '+' | '-' ]
               Term { ( '+' | '-' ) Term } .
Term =         NotFact { ( '*' | '/' | '%' ) NotFact } .
NotFact =      [ '!' ] Fact .
Fact =         'false' | 'true'
               | number
               | ident [ ( '[' Expr ']' )
                       | ( '(' [ ActParList ] ')' )
                       ]
               | 'new' Type '[' Expr ']'
               | '(' Expr ')' .

```

- Erzeugen Sie mit lex/flex einen lexikalischen Analysator (*scanner*) und mit yacc/bison einen Syntaxanalysator (*parser*) für MiniCpp und bauen Sie daraus ein Programm für die lexikalische und syntaktische Analyse von MiniCpp-Programmen.
- Erweitern Sie Ihre Grammatik aus a) zu einer ATG, sodass der (statische) Funktionsaufrufgraph des analysierten Programms erstellt wird. Gehen Sie so vor, dass von der ATG eine Textdatei (.gv) für *GraphViz* ([www.graphviz.org](http://www.graphviz.org)) erzeugt wird, die mit *GVEdit* (oder mit *dot.exe* direkt oder über [www.webgraphviz.com](http://www.webgraphviz.com)) in eine Abbildung umgesetzt werden kann.

# 1 MiniC: Scanner und Parser mit lex und yacc

## Lösungsidee

Als Erstes wird der Scanner mit Hilfe von *lex* erstellt, wobei hier alle Token abgebildet werden müssen.

Im zweiten Schritt wird dann der Parser mit Hilfe der gegebenen Grammatik erstellt.

## Sourcecode

```
..... ../Source/MiniC.l .....
1  /*MiniC.l:
2     -----
3     Lex/Flex description for MiniC.
4     =====*/
5
6  %{
7
8     #include "MiniC.tab.h" /*generated with "bison -d MiniC.y"      */
9     extern int yylval;     /*lexical attribute for current token   */
10    extern int yylineno;    /*current line number, initalized with 1 */
11  %}
12
13  %%
14
15  [ \t]+ { ; }             /*ignore white space: blanks and tabs */
16
17  "/*" {                   /*skip C comments */
18      int prevCh = 0, ch = input();
19      for(;;) {
20          if (ch == EOF)
21              break; /*error: end of file within comment */
22          if (ch == '\n')
23              yylineno++;
24          else if (prevCh == '*' && ch == '/')
25              break; /*ok: correct end of comment */
26          prevCh = ch;
27          ch = input();
28      } /*for*/
29  }
30
31  int { return INT; }
32  main { return MAIN; }
33  printf { return PRINTF; }
34  scanf { return SCANF; }
35  void { return VOID; }
36
37  [a-zA-Z][a-zA-Z0-9]* { return IDENT; }
38
39  [0-9]+ { yylval = atoi(yytext); return NUMBER; }
```

```
40
41 \n          { yylineno++; }
42
43 .          { return yytext[0]; } /*return all other chars
44                                     as tokens to parser: '+', ... */
45
46 %%
47
48 int yywrap() {
49     return 1; /*on end of input: no further files to scan */
50 } /*yywrap*/
51
52 /* End of MiniC.l
53 =====*/
```

---

```
..../Source/MiniC.y
1 %{ #include <stdio.h>
2     #include <stdlib.h>
3
4     extern int yylineno;
5     extern int yylval;
6 %}
7
8 %token INT
9 %token MAIN
10 %token PRINTF
11 %token SCANF
12 %token VOID
13
14 %token IDENT
15 %token NUMBER
16 %%
17 MC:
18     VOID MAIN '(' ')' Block
19 ;
20 Block:
21     '{' OptVarDecl StatSeq '}'
22 ;
23 OptVarDecl:
24     /* EPS */
25     | INT Idlist ';'
26 ;
27
28 Idlist:
29     IDENT
30     | Idlist ',' IDENT
31 ;
32
33 StatSeq:
```

```
34  /*EPS*/
35  | Stat
36  | StatSeq Stat
37  ;
38
39  Stat:
40  | ';'
41  | IDENT '=' Expr ';'
42  | SCANF '(' IDENT ')' ';'
43  | PRINTF '(' IDENT ')' ';'
44  ;
45
46  Expr:
47  | Term { $$ = $1; /*default*/ }
48  | Expr '+' Term { $$ = $1 + $3; }
49  | Expr '-' Term { $$ = $1 - $3; }
50  ;
51
52  Term:
53  | Fact { $$ = $1; /*default*/ }
54  | Term '*' Fact { $$ = $1 * $3; }
55  | Term '/' Fact { $$ = $1 / $3; }
56  ;
57
58  Fact:
59  | IDENT
60  | NUMBER { $$ = $1; /*default*/ }
61  | '(' Expr ')' { $$ = $2; }
62  ;
63  %%
64  extern FILE * yyin;
65  extern int yylineno;
66
67  int yyerror(char *message) {
68  | printf("ERROR %s in line %d\n",message,yylineno);
69  | return 0;
70  }
71  int main(int argc, char ** argv) {
72  | if(argc > 1) {
73  | | FILE *fin = fopen(argv[1],"r");
74  | | if(fin != NULL){
75  | | | yyin = fin;
76  | | } else {
77  | | | printf("ERROR: file not found");
78  | | | exit(-1);
79  | | }
80  | }
81  | yyparse();
```

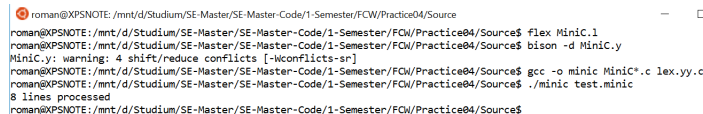
```
82 printf("%d lines processed\n",yylineno);
83 return 0;
84 }
```

---

```
..... ../Source/test.minic .....
1 void main() {
2     int a, b, cs;
3     scanf(a);
4     scanf(b);
5     cs = (a * a) + (b * b);
6     printf(cs);
7 }
```

---

### Test-Screenshots



```
roman@XPSNOTE:/mnt/d/Studium/SE-Master/SE-Master-Code/1-Semester/FCW/Practice04/Source$ flex Minic.l
roman@XPSNOTE:/mnt/d/Studium/SE-Master/SE-Master-Code/1-Semester/FCW/Practice04/Source$ bison -d Minic.y
Minic.y: warning: 4 shift/reduce conflicts [-Wconflicts-sr]
roman@XPSNOTE:/mnt/d/Studium/SE-Master/SE-Master-Code/1-Semester/FCW/Practice04/Source$ gcc -o minic Minic*.c lex.yy.c
roman@XPSNOTE:/mnt/d/Studium/SE-Master/SE-Master-Code/1-Semester/FCW/Practice04/Source$ ./minic test.minic
8 lines processed
roman@XPSNOTE:/mnt/d/Studium/SE-Master/SE-Master-Code/1-Semester/FCW/Practice04/Source$
```

## 2 MiniCpp: Scanner und Parser mit lex und yacc

### a) Lexikalischen Analysator und Syntaxanalysator

#### Lösungsidee

Analog zu Aufgabe-1, wobei hier die erweiterte Grammatik verwendet wird.

#### Sourcecode

```
..... ../Source/minicpp/MiniCpp.l .....
1 /*MiniCpp.l:
2     -----
3     Lex/Flex description for MiniCpp.
4     =====*/
5
6 %{
7
8     #ifndef YYSTYPE
9         # define YYSTYPE char*
10    #endif
11
12
13    #include "MiniCpp.tab.h" /*generated with "bison -d MiniCpp.y" */
14    extern int yylineno; /*current line number, initialized with 1 */
15    extern char *yylval;
16
```

```
17 %}
18
19 %%
20
21 [ \t]+ { ; }          /*ignore white space: blanks and tabs    */
22
23 "/*"      {           /*skip C comments                        */
24     int prevCh = 0, ch = input();
25     for(;;) {
26         if (ch == EOF)
27             break; /*error: end of file within comment    */
28         if (ch == '\n')
29             yylineno++;
30         else if (prevCh == '*' && ch == '/')
31             break; /*ok: correct end of comment            */
32         prevCh = ch;
33         ch = input();
34     } /*for*/
35 }
36
37 "//"      {           /*skip C comments                        */
38     int ch = input();
39     for(;;) {
40         if (ch == EOF)
41             break; /*error: end of file within comment    */
42
43         if (ch == '\n') {
44             yylineno++;
45             break;
46         }
47         ch = input();
48     } /*for*/
49 }
50
51 void      { return VOID;  }
52 int       { return INT;   }
53 bool      { return BOOL;  }
54 string    { return STRING; }
55
56 cout      { return COUT;  }
57 cin       { return CIN;   }
58 endl      { return ENDL;  }
59
60 ident     { return IDENT; }
61 number    { return NUMBER; }
62 true      { return TRUE;  }
63 false     { return FALSE; }
64
65 const     { return CONST; }
```

```
66
67 if      { return IF;      }
68 else    { return ELSE;    }
69 while   { return WHILE;   }
70 break   { return BREAK;   }
71 return  { return RETURN;  }
72
73 new      { return NEW;     }
74 delete   { return DELETE;  }
75
76
77 "||"     { return OR;      }
78 "&&"     { return AND;      }
79 ">>"     { return SHIFTR;   }
80 "<<"     { return SHIFTL;   }
81 "!="     { return NOTEQUAL; }
82 "=="     { return EQUAL;    }
83 "<="     { return SMALLEREQ; }
84 ">="     { return BIGGEREQ; }
85 "++"     { return PLUSPLUS; }
86 "--"     { return MINUSMINUS; }
87
88 [a-zA-Z][a-zA-Z0-9]* {yylval = strdup(yytext); return IDENT; }
89
90 "["[^"]*" { ; return STRING; }
91
92 [0-9]+     { ; return NUMBER; }
93
94 "\n"       { yylineno++; }
95
96 .         { return yytext[0]; } /*return all other chars
97                                     as tokens to parser: '+', ... */
98
99 %%
100
101 int yywrap() {
102     return 1; /*on end of input: no further files to scan */
103 } /*yywrap*/
104
105 /* End of MiniC.l
106 =====*/

```

---

```

1 % { #include <stdio.h>
2     #include <stdlib.h>
3
4     //change output type to char *
5     #ifndef YYSTYPE
6         # define YYSTYPE char*

```



```
7      #endif
8
9      extern int yylineno;
10     extern char* yylval;
11
12     //variable definitions
13     //=====
14     FILE *outputFile = NULL;
15     char *methods[255]; //array of
16                        // method calls per method
17                        //NOTE: should be a dynamic array
18     int cur = 0; //current index in array
19     int i=0; //variable used for loops
20 %}
21
22 %token VOID
23 %token INT
24 %token BOOL
25 %token STRING
26
27 %token COUT
28 %token CIN
29 %token ENDL
30
31 %token IDENT
32 %token NUMBER
33 %token TRUE
34 %token FALSE
35
36 %token CONST
37
38 %token IF
39 %token ELSE
40 %token WHILE
41 %token BREAK
42 %token RETURN
43
44 %token NEW
45 %token DELETE
46
47 %token OR
48 %token AND
49 %token SHIFTR
50 %token SHIFTL
51 %token NOTEQUAL
52 %token EQUAL
53 %token SMALLERREQ
54 %token BIGGEREQ
55 %token PLUSPLUS
```

```
56 %token MINUSMINUS
57
58
59 %%
60 MiniCpp:
61     /*EPS*/
62     | MiniCpp ConstDecl
63     | MiniCpp VarDef
64     | MiniCpp FuncDecl
65     | MiniCpp FuncDef
66     ;
67
68 ConstDecl:
69     CONST Type IDENT Init ';'
70     ;
71
72 Init:
73     '=' TypeValue
74     ;
75
76 TypeValue:
77     FALSE
78     | TRUE
79     | NUMBER
80     ;
81
82 VarDef:
83     Type OptStar IDENT OptInit VarDefList ';'
84     ;
85
86 OptStar:
87     /*EPS*/
88     | '*'
89     ;
90
91 OptInit:
92     /*EPS*/
93     | Init
94     ;
95
96 VarDefList:
97     /*EPS*/
98     | VarDefList ',' OptStar IDENT OptInit
99     ;
100
101 FuncDecl:
102     FuncHead ';'
103     ;
104
```

```
105 FuncDef:
106     FuncHead Block {
107         //loop through all saved methods
108         for(i=0;i<cur;i++){
109             //write to ouput file using dot syntax
110             //eg. main -> test
111             fprintf(outputFile,"%s -> %s\n",$1,methods[i]);
112             free(methods[i]); //free string
113                             //which was produced by strdup
114         }
115         //reset method index
116         cur=0;
117     }
118 ;
119
120 FuncHead:
121     Type OptStar IDENT '(' FormParList ')' {$$ = $3;}
122 ;
123
124 FormParList:
125     /*EPS*/
126     | VOID
127     | Type OptStar IDENT OptBrackets FormParListList
128 ;
129
130 OptBrackets:
131     /*EPS*/
132     | '[' ']'
133 ;
134
135 FormParListList:
136     /*EPS*/
137     | FormParListList ',' Type OptStar IDENT OptBrackets
138 ;
139
140 Type:
141     VOID
142     | BOOL
143     | INT
144 ;
145
146 Block:
147     '{' BlockList '}'
148 ;
149
150 BlockList:
151     /*EPS*/
152     | BlockList ConstDecl
153     | BlockList VarDef
```

```
154 | BlockList Stat
155 ;
156
157 Stat:
158     IncStat
159 | DecStat
160 | AssignStat
161 | CallStat
162 | IfStat
163 | WhileStat
164 | BreakStat
165 | InputStat
166 | OutputStat
167 | DeleteStat
168 | ReturnStat
169 | Block
170 | ';'
171 ;
172
173 IncStat:
174     IDENT PLUSPLUS ';'
175 ;
176
177 DecStat:
178     IDENT MINUSMINUS ';'
179 ;
180
181 AssignStat:
182     IDENT OptExpr '=' Expr ';'
183 ;
184
185 OptExpr:
186     /*EPS*/
187     | '[' Expr ']'
188
189 CallStat:
190     IDENT '(' ' ' ')' ';'
191 | IDENT '(' ActParList ' ' ')' ';'
192 ;
193
194 ActParList:
195     Expr ActParListList
196 ;
197
198 ActParListList:
199     /*EPS*/
200     | ActParListList ',' Expr
201 ;
202
```

```
{ methods[cur]=$1;cur++; /*save method name*/ }
{ methods[cur]=$1;cur++; /*save method name*/ }
```

```
203 IfStat:
204     IF '(' Expr ')' Stat
205 | IF '(' Expr ')' Stat ELSE Stat
206 ;
207
208 WhileStat:
209     WHILE '(' Expr ')' Stat
210 ;
211
212 BreakStat:
213     BREAK ';'
214 ;
215
216 InputStat:
217     CIN SHIFTR IDENT ';'
218 ;
219
220 OutputStat:
221     COUT SHIFTL Expr OutputStatList ';'
222 | COUT SHIFTL STRING OutputStatList ';'
223 | COUT SHIFTL ENDL OutputStatList ';'
224 ;
225
226 OutputStatList:
227     /*EPS*/
228 | OutputStatList SHIFTL Expr
229 | OutputStatList SHIFTL STRING
230 | OutputStatList SHIFTL ENDL
231 ;
232
233 DeleteStat:
234     DELETE '[' ']' IDENT ';'
235 ;
236
237 ReturnStat:
238     RETURN ';'
239 | RETURN Expr ';'
240
241 Expr:
242     OrExpr
243 ;
244
245 OrExpr:
246     AndExpr OrExprList
247 ;
248
249 OrExprList:
250     /*EPS*/
251 | OrExprList OR AndExpr
```

```
252 ;
253
254 AndExpr:
255     RelExpr AndExprList
256 ;
257
258 AndExprList:
259     /*EPS*/
260     | AndExprList AND RelExpr
261 ;
262
263 RelExpr:
264     SimpleExpr
265     | SimpleExpr ExprOp SimpleExpr
266 ;
267
268 ExprOp:
269     EQUAL
270     | NOTEQUAL
271     | '<'
272     | '>'
273     | SMALLEREQ
274     | BIGGEREQ
275
276 SimpleExpr:
277     OptSign Term
278     | OptSign Term SimpleExprList
279 ;
280
281 OptSign:
282     /*EPS*/
283     | '+'
284     | '-'
285
286 SimpleExprList:
287     OptSign Term
288 ;
289
290 Term:
291     NotFact TermList
292 ;
293
294 TermList:
295     /*EPS*/
296     | '*' NotFact
297     | '/' NotFact
298     | '%' NotFact
299 ;
300
```

```
301 NotFact:
302     Fact
303     | '!' Fact
304     ;
305
306 Fact:
307     FALSE
308     | TRUE
309     | NUMBER
310     | IDENT
311     | IDENT '[' Expr ']'
312     | IDENT '(' ')'
313     | IDENT '(' ActParList ')'
314     | NEW Type '[' Expr ']'
315     | '(' Expr ')'
316     ;
317
318 %%
319 extern FILE * yyin;
320 extern int yylineno;
321
322 int yyerror(char *message) {
323     printf("ERROR %s in line %d\n",message,yylineno);
324     return 0;
325 }
326 int main(int argc, char ** argv) {
327
328     if(argc > 1) {
329         FILE *fin = fopen(argv[1],"r");
330         if(fin != NULL){
331             yyin = fin;
332         } else {
333             printf("ERROR: file not found");
334             exit(-1);
335         }
336     }
337
338     //open graphviz output file
339     outputFile = fopen("output.gv","w");
340     if(outputFile == NULL){
341         printf("ERROR: could not open output file output.gv");
342         exit(-1);
343     }
344
345     //graphviz header
346     fprintf(outputFile,"digraph G {");
347     yyparse();
348     //graphviz end
349     fprintf(outputFile,"}");
```

```
350     fclose(outputFile);
351     printf("%d lines processed\n",yylineno);
352     return 0;
353 }
```

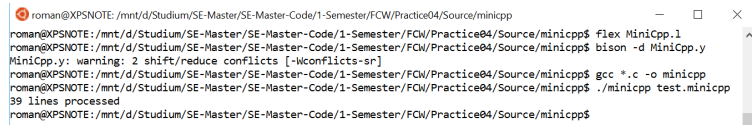
---

```
..... ../Source/minicpp/test.minicpp .....
1 void Sieve(int n); // declaration
2 void main(void) {
3     int n;
4     cout << "n > ";
5     cin >> n;
6     if(n == 0)
7         Sieve();
8
9 }
10 void Sieve(int n) { // definition
11     int col, i, j;
12     bool *sieve = 0;
13     sieve = new bool[n + 1];
14     i = 2;
15     while (i <= n) {
16         sieve[i] = true;
17         i++;
18     } // while
19     cout << 2 << " ";
20     col = 1;
21     i = 3;
22     while (i <= n) {
23         if (sieve[i]) {
24             if (col == 10) {
25                 cout << endl;
26                 col = 0;
27             } // if
28             col++;
29             cout << i << " ";
30             j = i * i;
31             while (j <= n) {
32                 sieve[j] = false;
33                 j = j + 2 * i;
34             } // while
35         } // if
36         i = i + 2;
37     } // while
38     delete[] sieve;
39 } // Sieve
```

---



### Test-Screenshots



```
roman@XPSNOTE: /mnt/d/Studium/SE-Master/SE-Master-Code/1-Semester/FCW/Practice04/Source/minicpp
roman@XPSNOTE: /mnt/d/Studium/SE-Master/SE-Master-Code/1-Semester/FCW/Practice04/Source/minicpp$ flex MiniCpp.l
roman@XPSNOTE: /mnt/d/Studium/SE-Master/SE-Master-Code/1-Semester/FCW/Practice04/Source/minicpp$ bison -d MiniCpp.y
MiniCpp.y: warning: 2 shift/reduce conflicts [-Wconflicts-sr]
roman@XPSNOTE: /mnt/d/Studium/SE-Master/SE-Master-Code/1-Semester/FCW/Practice04/Source/minicpp$ gcc *.c -o minicpp
roman@XPSNOTE: /mnt/d/Studium/SE-Master/SE-Master-Code/1-Semester/FCW/Practice04/Source/minicpp$ ./minicpp test.minicpp
39 lines processed
roman@XPSNOTE: /mnt/d/Studium/SE-Master/SE-Master-Code/1-Semester/FCW/Practice04/Source/minicpp$
```

## b) ATG - Erstellung des Funktionsaufrufgraph

### Lösungsidee

Um den Funktionsaufrufgraph muss die Grammatik um Aktionen erweitert werden. Da der erzeugte Analysator bottom-up Syntaxanalyse betreibt, müssen alle Methodenaufrufe, die sich innerhalb einer Methode befinden temporär gespeichert werden. Nachdem der Analysator dann bei **FuncDef** angekommen ist, steht die umgebende Methode fest und die gespeicherten Methoden können in die Ausgabedatei geschrieben werden.

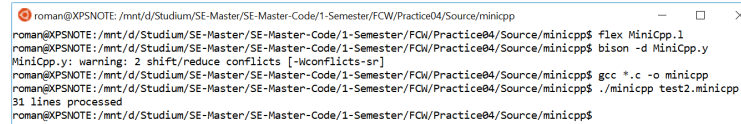
### Sourcecode

Die Änderungen am Quellcode dieser Aufgabe ist bereits in Aufgabe 2a enthalten.

```
..... ../Source/minicpp/test2.minicpp .....
1 void testA();
2 void testB();
3 void testC();
4 void testD();
5 void testE(int i);
6 void test();
7 void testA(){
8     testB();
9     testD();
10 }
11
12 void testB() {
13     testC();
14 }
15
16 void test() {
17     testA();
18     testD();
19 }
20
21 void main() {
22     test();
23     testE(5);
24 }
25
26 //recursion and after main method
27 void testE(int i) {
```

```
28  if(i == 0)
29      return;
30  testE(i-1);
31 }
```

### Test-Screenshots



```
roman@XPSNOTE: /mnt/d/Studium/SE-Master/SE-Master-Code/1-Semester/FCW/Practice04/Source/minicpp
roman@XPSNOTE: /mnt/d/Studium/SE-Master/SE-Master-Code/1-Semester/FCW/Practice04/Source/minicpp$ flex MiniCpp.l
roman@XPSNOTE: /mnt/d/Studium/SE-Master/SE-Master-Code/1-Semester/FCW/Practice04/Source/minicpp$ bison -d MiniCpp.y
MiniCpp.y: warning: 2 shift/reduce conflicts [-Wconflicts-sr]
roman@XPSNOTE: /mnt/d/Studium/SE-Master/SE-Master-Code/1-Semester/FCW/Practice04/Source/minicpp$ gcc *.c -o minicpp
roman@XPSNOTE: /mnt/d/Studium/SE-Master/SE-Master-Code/1-Semester/FCW/Practice04/Source/minicpp$ ./minicpp test2.minicpp
31 lines processed
roman@XPSNOTE: /mnt/d/Studium/SE-Master/SE-Master-Code/1-Semester/FCW/Practice04/Source/minicpp$
```

```
1 digraph G {testA -> testB
2 testA -> testD
3 testB -> testC
4 test -> testA
5 test -> testD
6 main -> test
7 main -> testE
8 testE -> testE
9 }
```

