1. MiniC: Scanner und Parser mit (f)lex und yacc/bison

MiniC.l

```
/*MiniC.l:
  ------
  Description of the lexical structure for MiniCw.
====================================================================*/

%{

  #include "MiniC.tab.h" /*generated by yacc/bison from MiniC.y
                           if option -d is used, defines NUMBER      */



%}

%%

[ \t\r\n]+   { ; }          /*ignore white space: blanks, tabs and new line
*/

[0-9]+   { return NUMBER; }

void                    { return VOID; }
main                    { return MAIN; }
int                     { return INT; }
scanf                   { return SCANF; }
printf                  { return PRINTF; }

[A-Za-z_][A-Za-z0-9\_]* { return IDENT; }

.                       { return yytext[0]; } /*return all other chars
                                 as tokens: '+', '-', ...           */
%%

int yywrap() {
  return 1; /*on end of input: no further files to scan             */
} /*yywrap*/

/* End of MiniC.l
====================================================================*/
```

MiniC.y

```
/*MiniC.y:
  ------
  Attributed grammar for MiniC.
```

```
====================================================================*/

%{
    #include <stdio.h>
%}

%token NUMBER
%token IDENT
%token VOID
%token MAIN
%token INT
%token SCANF
%token PRINTF

%%

MiniC: VOID MAIN '(' ')' '{'
                    OptVarDecl
                    StatSeq
                    '}'
  ;

OptVarDecl: /* eps */
  | VarDecl
  ;

VarDecl: INT IdList ';'
  ;

IdList: IDENT
  | IdList ',' IDENT
  ;

StatSeq: Stat
  | StatSeq Stat
  ;

Stat: ';'
  | IDENT '=' Expr ';'
  | SCANF '(' IDENT ')' ';'
  | PRINTF '(' Expr ')' ';'
  ;

Expr: Term
  | Expr '+' Term
  | Expr '-' Term
  ;

Term: Fact
  | Term '*' Fact
  | Term '/' Fact
  ;

Fact: IDENT
```

```
  | NUMBER
  | '(' Expr ')'
  ;

%%
extern int yylineno;

int yyerror(char *msg) {
  printf("error: %s in line %d\n", msg, yylineno);
  return 0;
} /*yyerror*/

int main(int argc, char *argv[]) {
  yyparse();
  return 0;
} /*main*/


/* End of Calc.y
=====================================================================*/
```

Commands

```
..\Flex-2.5.37\flex.exe --yylineno MiniC.l
..\Bison-2.7\bison.exe -g -d MiniC.y
gcc lex.yy.c MiniC.tab.c -o MiniC.exe
MiniC.exe < SVP.mc
```

SVP.mc

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

2. MiniCpp: Scanner und Parser mit (f)lex und yacc/bison UND ...

```
MiniCpp: MiniCppList
  ;
```

```
MiniCppList: /* eps */
  | MiniCppList ConstDef
  | MiniCppList VarDef
  | MiniCppList FuncDecl
  | MiniCppList FuncDef
  | MiniCppList ';'
  ;

ConstDef: CONST Type IDENT Init IdentList ';'
  ;

IdentList: /* eps */
  | IdentList ',' IDENT Init
  ;

Init: '=' FALSE
  | '=' TRUE
  | '=' NULLPTR
  | '=' '+' NUMBER
  | '=' '-' NUMBER
  ;

FuncDecl: FuncHead ';'
  ;

FuncDef: FuncHead Block
  ;

FuncHead: Type '*' IDENT '(' ')'
  | Type IDENT '(' ')'
  | Type IDENT '(' FormParList ')'
  | Type '*' IDENT '(' FormParList ')'
  ;

FormParList: VOID
  | TypeIdent TypeIdentList
  ;

TypeIdentList: /* eps */
  | TypeIdentList ',' TypeIdent
  ;

TypeIdent: Type '*' IDENT '[' ']'
  | Type '*' IDENT
  | Type IDENT '[' ']'
  ;

Type: VOID
  | BOOL
  | INT
  ;

Block: '{' BlockList '}'
  ;
```

```
BlockList: /* eps */
  | BlockList ConstDef
  | BlockList VarDef
  | BlockList Stat
  ;

Stat: EmptyStat
  | BlockStat
  | ExprStat
  | IfStat
  | WhileStat
  | BreakStat
  | InputStat
  | OutputStat
  | DeleteStat
  | ReturnStat
  ;

EmptyStat: ';'
  ;

BlockStat: Block
  ;

ExprStat: Expr
  ;

IfStat: IF '(' Expr ')' Stat StatList
  ;

StatList: /* eps */
  | StatList ELSE Stat
  ;

WhileStat: WHILE '(' Expr ')'
  ;

BreakStat: BREAK ';'
  ;

InputStat: CIN '>>' IDENT ';'
  ;

OutputStat: COUT CoutRight CoutRightList ';'
  ;

CoutRightList: /* eps */
  | CoutRightList CoutRight
  ;

CoutRight: '<<' Expr
  | '<<' STRING
  | '<<' ENDL
  ;

DeleteStat: DELETE '[' ']' IDENT ';'
  ;

ReturnStat: RETURN ';'
  | RETURN Expr ';'
  ;

Expr: OrExpr OrExprList
  ;

OrExprList: /* eps */
  | OrExprList '=' OrExpr
  | OrExprList '+=' OrExpr
  | OrExprList '-=' OrExpr
  | OrExprList '*=' OrExpr
  | OrExprList '/=' OrExpr
  | OrExprList '%=' OrExpr
  ;

OrExpr: AndExpr AndExprList
  ;

AndExprList: /* eps */
  | AndExprList '||' AndExpr
  ;

AndExpr: RelExpr RelExprList
  ;

RelExprList: /* eps */
  | RelExprList '&&' RelExpr
  ;

RelExpr: SimpleExpr SimpleExprList
  ;

SimpleExprList: /* eps */
  | SimpleExprList '==' SimpleExpr
  | SimpleExprList '!=' SimpleExpr
  | SimpleExprList '<' SimpleExpr
  | SimpleExprList '<=' SimpleExpr
  | SimpleExprList '>' SimpleExpr
  | SimpleExprList '>=' SimpleExpr
  ;

SimpleExpr: '+' Term TermList
  | '-' Term TermList
  | Term TermList
  ;

TermList: /* eps */
  | TermList '+' Term
  | TermList '-' Term
  ;

Term: NotFact NotFactList
  ;

NotFactList: /* eps */
  | NotFactList '*' NotFact
  | NotFactList '/' NotFact
  | NotFactList '%' NotFact
  ;

NotFact: Fact
  | '!' Fact
  ;

Fact: FALSE
  | TRUE
  | NULLPTR
  | NUMBER
  | DudeWtf
  | NEW Type '[' Expr ']'
  | '(' Expr ')'
  ;

DudeWtf: OptDecrOrIncr IDENT WeirdIdentShit OptDecrOrIncr
  ;

WeirdIdentShit: /* eps */
  | '[' Expr ']'
  | '(' ActParList ')'
  | '(' ')'
  ;

OptDecrOrIncr: /* eps */
  | '++'
  | '--'
  ;

ActParList: Expr ExprList
  ;

ExprList: /* eps */
  | ExprList ',' Expr
  ;
```