Bilag A - Grammatik og programkode

Tony grammatikken

```
<Program>          : <Body>
<Function>         : <Head> <Body> <Tail>
<Head>             : func id ( <Par_decl_list> ) :  <Type>
<Tail>             : end id ;
<Type>             : id
                   | int
                   | bool
                   | array of <Type>
                   | record of { <Var_decl_list> }
<Par_decl_list>    : <Var_decl_list>
                   |  ε
<Var_decl_list>     : <Var_decl_list> , <Var_type>
                   | <Var_type>
<Var_type>         : id : <Type>
<Body>             : <Decl_list> <Statement_list>
<Decl_list>        : <Decl_list> <Declaration> ;
                   |  ε
<Declaration>      : type id  =  <Type>
                   | <Function>
                   | var <Var_decl_list>
<Statement_list>   : <Statement_list> <Statement>
                   | <Statement>
<Statement>        : return <Expression> ;
                   | write <Expression> ;
                   | new <Variable> of length <Expression> ;
                   | new <Variable> ;
                   | <Variable> = <Expression> ;
                   | <Variable> += <Expression> ;
                   | <Variable> -= <Expression> ;
                   | if <Expression> then <Statement> else <tStatement>
                   | if <Expression> then <Statement>
                   | while <Expression> do <Statement>
                   | for <Variable> = <Expression> to <Expression> do <Statement>
                   | for <Variable> = <Expression> downto <Expression> do <Statement>
                   | {  <Statement_list>  }
<Variable>         : id
                   | <Variable> [ <Expression> ]
                   | <Variable> . id
<Expression>       : <Term>
                   | <Expression> * <Expression>
                   | <Expression> / <Expression>
                   | <Expression> + <Expression>
                   | <Expression> - <Expression>
                   | <Expression> < <Expression>
                   | <Expression> > <Expression>
                   | <Expression> <= <Expression>
                   | <Expression> >= <Expression>
                   | <Expression> == <Expression>
                   | <Expression> != <Expression>
                   | <Expression> && <Expression>
                   | <Expression> || <Expression>
<Term>             : <Variable>
                   | - <Variable>
                   | id ( <Act_list> )
                   | ( <Expression> )
                   | ! <Term>
                   | | <Expression> |
                   | - num
                   | num
                   | true
                   | false
                   | null
<Act_list>         : <Exp_list>
                   |  ε
<Exp_list>         : <Expression>
                   | <Exp_list> , <Expression>
```

Scan fasen

Input til flex

tony.lex

```
%option noyywrap
%{
/* ------------------------------------------------------------
  Definition af TONY tokens til flex
  Programmør: Bjørk Busch
  Historik: 2005.03.26  Udgave til rapport 3 færdig
  Historik: 2005.04.26  advarsler ændret til fejl
  Historik: 2005.05.10  kosmetik i source
------------------------------------------------------------*/
#include "tony.tab.h"
#include "bjbu_std.h"
#include <string.h>

extern int linienr;
extern int id_linienr;

int commentLevel = 0;
void yyerror (char const *);

%}

%x COMMENT

%%
"\r\n"      {++linienr;} /* Windows */
"\n"        {++linienr;} /* linux */
[ \t]+      {}/* ignorer */

"(*"        {
              ++commentLevel;
              BEGIN(COMMENT);
            }
"*)"        {
              yyerror("comment slut uden start");
            }

"func"      return tFUNC;
"end"       return tEND;
"int"       return tINT;
"bool"      return tBOOL;
"of"        return tOF;
"array"     return tARRAY;
"record"    return tRECORD;
"type"      return tTYPE;
"var"       return tVAR;
"return"    return tRETURN;
"write"     return tWRITE;
"new"       return tNEW;
"if"        return tIF;
"then"      return tTHEN;
"while"     return tWHILE;
"do"        return tDO;
"length"    return tLENGTH;
"else"      return tELSE;
```

```
"for"       return tFOR;
"to"        return tTO;
"downto"    return tDOWNTO;

"="         return tASSIGN;
"*"         return tMUL;
"/"         return tDIV;
"+="        return tADDTO;
"+"         return tADD;
"-"         return tSUB;
"-="        return tSUBFROM;
"=="        return tEQ;
"!="        return tNEQ;
"<"         return tLT;
">"         return tGT;
"<="        return tLTEQ;
">="        return tGTEQ;
"&&"        return tAND;
"||"        return tOR;
"!"         return tNOT;
"|"         return tNUMERIC;
"null"      return tNULL;
"true"      return tTRUE;
"false"     return tFALSE;
"("         return tBEGPARANTES;
")"         return tENDPARANTES;
":"         return tCOLON;
";"         return tSEMICOLON;
","         return tKOMMA;
"."         return tPUNKTUM;
"["         return tBEGINDEX;
"]"         return tENDINDEX;
"{"         return tBEGTUBORG;
"}"         return tENDTUBORG;
0|([1-9][0-9]*)   {
              yylval.intconst = atoi(yytext);
              return tINTCONST;
            }

[a-zA-Z_][a-zA-Z0-9_]* {
              yylval.stringconst = NewStringCopy(yytext);
              id_linienr = linienr;
              return tIDENTIFIER;
            }
<INITIAL>.  {
              yyerror("Fejl ikke gyldig symboler");
            }

<COMMENT>"\r\n"   {++linienr;} /* Windows */
<COMMENT>"\n"     {++linienr;} /* linux */
<COMMENT>[ \t]+   {}/* ignorer */
<COMMENT>"(*"     {
                    ++commentLevel;
                  }
<COMMENT>"*)"     {
                    --commentLevel;
                    if (commentLevel==0)
                      BEGIN(INITIAL);
                  }
<COMMENT><<EOF>>  {
                    yyerror("Fejl comment ikke slutet");
                  }
<COMMENT>.        {}/* ignorer  andet inde i kommentar*/
```

%%

Parse fasen

Input til bison

tony.y

```
%{
/* ------------------------------------------------------
  Definition af TONY grammatik m.m. til bison
  Programmør: Bjørk Busch
  Historik: 2005.03.26  Udgave til rapport 3 færdig
            2005.05.09  Udgave til rapport 4 færdig
------------------------------------------------------*/

#include <stdio.h>
#include "bjbu_std.h"
#include "symbol.h"
#include "tonyAST.h"
#include "tonyAST_Make.h"

/* fællesdata */
sProgram *theProgram;
int linienr = 1;
int id_linienr = 1;
int alvorligFejl = 0;
int fejl = 0;
int advarsel = 0;


void yyerror (char const *);
int yylex(void);

%}

%union {
  int intconst;
  char *stringconst;
  struct sFunction *function;
  struct sHead *head;
  struct sTail *tail;
  struct sType *type;
  struct sPar_decl_list *par_decl_list;
  struct sVar_decl_list *var_decl_list;
  struct sVar_type *var_type;
  struct sBody *body;
  struct sDecl_list *decl_list;
  struct sDeclaration *declaration;
  struct sStatement_list *statement_list;
  struct sStatement *statement;
  struct sVariable *variable;
  struct sExpression *expression;
  struct sTerm *term;
  struct sAct_list *act_list;
  struct sExp_list *exp_list;
}


/* definition til bison */
%type <function> ntFunction;
%type <head> ntHead;
%type <tail> ntTail;
%type <type> ntType;
```

```
%type <par_decl_list> ntPar_decl_list;
%type <var_decl_list> ntVar_decl_list;
%type <var_type> ntVar_type;
%type <body> ntBody;
%type <decl_list> ntDecl_list;
%type <declaration> ntDeclaration;
%type <statement_list> ntStatement_list;
%type <statement> ntStatement;
%type <variable> ntVariable;
%type <expression> ntExpression;
%type <term> ntTerm;
%type <act_list> ntAct_list;
%type <exp_list> ntExp_list;


/* std. tokens */
%token tANDET;
%token <intconst> tINTCONST
%token <stringconst> tIDENTIFIER
%token  tASSIGN
%token  tMUL
%token  tDIV
%token  tADD
%token  tADDTO
%token  tSUB
%token  tSUBFROM
%token  tEQ
%token  tNEQ
%token  tLT
%token  tGT
%token  tLTEQ
%token  tGTEQ
%token  tAND
%token  tOR
%token  tNOT
%token  tNUMERIC
%token  tNULL
%token  tTRUE
%token  tFALSE
%token  tBEGPARANTES
%token  tENDPARANTES
%token  tCOLON
%token  tSEMICOLON
%token  tKOMMA
%token  tPUNKTUM
%token  tBEGINDEX
%token  tENDINDEX
%token  tBEGTUBORG
%token  tENDTUBORG

/* TONY tokens */
%token  tFUNC
%token  tEND
%token  tINT
%token  tBOOL
%token  tOF
%token  tARRAY
%token  tRECORD
%token  tTYPE
%token  tVAR
%token  tRETURN
%token  tWRITE
%token  tNEW
```

```
%token  tIF
%token  tTHEN
%token  tWHILE
%token  tDO
%token  tLENGTH
%token  tELSE
%token  tFOR
%token  tTO
%token  tDOWNTO


/* præcedence for operatorer - monadisk over dyadisk
  (1) !
  (2) * /
  (3) + -
  (4) < > <= =>
  (5) == !=
  (6) &&
  (7) ||
*/

%left tOR
%left tAND
%left tEQ tNEQ          /* ønskes indbyrdes venste-mest binding */
%left tLT tGT tLTEQ tGTEQ   /* ønskes indbyrdes venste-mest binding */
%left tADD tSUB         /* ønskes indbyrdes venste-mest binding */
%left tMUL tDIV         /* ønskes indbyrdes venste-mest binding */
/* %left tNOT unødvendig her da grammatik løser det */

%start ntProgram

%%
ntProgram : ntBody {
              theProgram = makeProgram ($1); }
          ;
ntFunction: ntHead ntBody ntTail {
              $$ = makeFunction($1,$2,$3); }
          ;
ntHead    : tFUNC tIDENTIFIER tBEGPARANTES ntPar_decl_list tENDPARANTES
              tCOLON ntType  {
              $$ = makeHead($2,$4,$7); }
          ;
ntTail    : tEND tIDENTIFIER {
              $$ = makeTail($2); }
          ;
ntType    : tIDENTIFIER {
              $$ = makeType_Id($1); }
          | tINT {
              $$ = makeType_Int(); }
          | tBOOL {
              $$ = makeType_Bool(); }
          | tARRAY tOF ntType {
              $$ = makeType_Array($3); }
          | tRECORD tOF tBEGTUBORG ntVar_decl_list tENDTUBORG {
              $$ = makeType_Record($4); }
          ;
ntPar_decl_list: ntVar_decl_list {
              $$ = makePar_decl_list_List($1); }
          | /* empty */ {
              $$ = makePar_decl_list_Empty(); }
          ;
ntVar_decl_list: ntVar_decl_list tKOMMA ntVar_type {
              $$ = makeVar_decl_list_List($1,$3); }
```

```
          | ntVar_type {
            $$ = makeVar_decl_list_Var($1); }
          ;
ntVar_type: tIDENTIFIER tCOLON ntType {
            $$ = makeVar_type($1,$3); }
          ;
ntBody    : ntDecl_list ntStatement_list {
            $$ = makeBody($1,$2); }
          ;
ntDecl_list: ntDecl_list ntDeclaration tSEMICOLON {
            $$ = makeDecl_list_List($1,$2); }
          | /* empty */ {
            $$ = makeDecl_list_Empty(); }
          ;
ntDeclaration: tTYPE tIDENTIFIER tASSIGN ntType {
            $$ = makeDeclaration_Type($2,$4); }
          | ntFunction {
            $$ = makeDeclaration_Function($1); }
          | tVAR ntVar_decl_list {
            $$ = makeDeclaration_Var($2); }
          ;
ntStatement_list: ntStatement_list ntStatement {
            $$ = makeStatement_list_List($1,$2); }
          | ntStatement {
            $$ = makeStatement_list_Statement($1); }
          ;
ntStatement: tRETURN ntExpression tSEMICOLON {
            $$ = makeStatement_Return($2); }
          | tWRITE ntExpression tSEMICOLON {
            $$ = makeStatement_Write($2); }
          | tNEW ntVariable tOF tLENGTH ntExpression tSEMICOLON {
            $$ = makeStatement_NewLength($2,$5); }
          | tNEW ntVariable tSEMICOLON {
            $$ = makeStatement_New($2); }
          | ntVariable tASSIGN ntExpression tSEMICOLON {
            $$ = makeStatement_Assign($1,$3); }
          | ntVariable tADDTO ntExpression tSEMICOLON {
            $$ = makeStatement_Assign($1,
                 makeExpression_Dyadisk(
                 makeExpression_Term(
                 makeTerm_Variable($1)),kExp_Add,$3)); }
          | ntVariable tSUBFROM ntExpression tSEMICOLON {
            $$ = makeStatement_Assign($1,
                 makeExpression_Dyadisk(
                 makeExpression_Term(
                 makeTerm_Variable($1)),kExp_Sub,$3)); }
          | tIF ntExpression tTHEN ntStatement tELSE ntStatement {
            $$ = makeStatement_IfElse($2,$4,$6); }
          | tIF ntExpression tTHEN ntStatement {
            $$ = makeStatement_If($2,$4); }
          | tWHILE ntExpression tDO ntStatement {
            $$ = makeStatement_While($2,$4); }
          | tBEGTUBORG ntStatement_list tENDTUBORG {
            $$ = makeStatement_Compound($2); }
          | tFOR ntVariable tASSIGN ntExpression
            tTO ntExpression tDO ntStatement {
            $$ = makeStatement_ForTo($2,$4,kStatement_ForTo,$6,$8); }
          | tFOR ntVariable tASSIGN ntExpression
            tDOWNTO ntExpression tDO ntStatement {
            $$ = makeStatement_ForTo($2,$4,kStatement_ForDownto,$6,$8); }
          ;
ntVariable: tIDENTIFIER {
            $$ = makeVariable_Id($1); }
```

```
          | ntVariable tBEGINDEX ntExpression tENDINDEX {
            $$ = makeVariable_Indexed($1,$3); }
          | ntVariable tPUNKTUM tIDENTIFIER {
            $$ = makeVariable_Struct($1,$3); }
          ;
ntExpression: ntTerm {
            $$ = makeExpression_Term($1); }
          | ntExpression tMUL ntExpression  {
            $$ = makeExpression_Dyadisk($1,kExp_Mul,$3); }
          | ntExpression tDIV ntExpression  {
            $$ = makeExpression_Dyadisk($1,kExp_Div,$3); }
          | ntExpression tADD ntExpression  {
            $$ = makeExpression_Dyadisk($1,kExp_Add,$3); }
          | ntExpression tSUB ntExpression  {
            $$ = makeExpression_Dyadisk($1,kExp_Sub,$3); }
          | ntExpression tLT ntExpression   {
            $$ = makeExpression_Dyadisk($1,kExp_Lt,$3); }
          | ntExpression tGT ntExpression   {
            $$ = makeExpression_Dyadisk($1,kExp_Gt,$3); }
          | ntExpression tLTEQ ntExpression {
            $$ = makeExpression_Dyadisk($1,kExp_Lteq,$3); }
          | ntExpression tGTEQ ntExpression {
            $$ = makeExpression_Dyadisk($1,kExp_Gteq,$3); }
          | ntExpression tEQ ntExpression   {
            $$ = makeExpression_Dyadisk($1,kExp_Eq,$3); }
          | ntExpression tNEQ ntExpression  {
            $$ = makeExpression_Dyadisk($1,kExp_Neq,$3); }
          | ntExpression tAND ntExpression  {
            $$ = makeExpression_Dyadisk($1,kExp_And,$3); }
          | ntExpression tOR ntExpression   {
            $$ = makeExpression_Dyadisk($1,kExp_Or,$3); }
ntTerm    : ntVariable {
            $$ = makeTerm_Variable($1); }
          | tSUB ntVariable {
            $$ = makeTerm_Parantes (
                  makeExpression_Dyadisk (
                  makeExpression_Term(makeTerm_Num(0)),
                  kExp_Sub,
                  makeExpression_Term(makeTerm_Variable($2))))
               }
          | tIDENTIFIER tBEGPARANTES ntAct_list tENDPARANTES {
            $$ = makeTerm_Call($1,$3); }
          | tBEGPARANTES ntExpression tENDPARANTES {
            $$ = makeTerm_Parantes($2); }
          | tNOT ntTerm {
            $$ = makeTerm_Not($2); }
          | tNUMERIC ntExpression tNUMERIC {
            $$ = makeTerm_Numeric($2); }
          | tSUB tINTCONST {
            $$ = makeTerm_Num(-$2); }
          | tINTCONST {
            $$ = makeTerm_Num($1); }
          | tTRUE {
            $$ = makeTerm_True(); }
          | tFALSE
            { $$ = makeTerm_False(); }
          | tNULL
            { $$ = makeTerm_Null(); }
          ;
ntAct_list: ntExp_list {
            $$ = makeAct_list_List($1); }
          | /* empty */ {
            $$ = makeAct_list_Empty(); }
```

```
          ;
ntExp_list: ntExpression {
            $$ = makeExp_list_Expression($1); }
          | ntExp_list tKOMMA ntExpression {
            $$ = makeExp_list_List($1,$3); }
          ;


%%
extern FILE *msgFile;
extern char *yytext;
void yyerror(char const *txt)
{
  if (txt == NULL)
    fprintf(msgFile, "%d: syntaksfejl ved %s\n", linienr, yytext);
  else
    fprintf(msgFile, "%d: syntaksfejl ved %s <%s>\n", linienr, yytext, txt);
  alvorligFejl++;
  exit(0);
}
```

Strukturer til tony AST

tonyAST.h

```
/* ------------------------------------------------------------------------
  Definition af strukturer for TONY abstract syntax tree
  Programmør: Bjørk Busch
  Historik: 2005.03.26
  Historik: 2005.05.10  kosmetik i source
------------------------------------------------------------------------*/

typedef struct sProgram {
  int linienr;
  int maxScopeLevel;
  struct SymbolTable *symbolTable;
  union {
    struct { struct sBody *body; } eProgram;
  } val;
}sProgram;

typedef struct sFunction {
  int linienr;
  struct SymbolTable *symbolTable;
  union {
    struct { struct sHead *head; struct sBody *body;
             struct sTail *tail; } eFunction;
  } val;
}sFunction;

typedef struct sHead {
  int linienr;
  int labelnr;  /* for unique id */
  struct asElm *labelEntry;
  struct asElm *labelExit;
  union {
    struct { char *id; struct sPar_decl_list *par_decl_list;
         struct sType *type; int id_linienr; struct SYMBOL *symbol;} eHead;
    } val;
}sHead;

typedef struct sTail {
  int linienr;
```

```
  union {
    struct { char *id; } eTail;
  } val;
}sTail;

typedef struct sType {
  int linienr;
  struct sType *slutType;   /* sættes på i forb. med type-check */
  int dataLength;
  enum {kType_Id, kType_Int, kType_Bool, kType_Array,
        kType_Record, kType_Null} kind;
        /* sidste ikke i AST men en expression-type */
  union {
    struct { char *id; struct sType *type; int id_linienr; } eId;
    struct { struct sType *type; int elmLength; } eArray;
    struct { struct sVar_decl_list *var_decl_list;
             struct SymbolTable *symbolTable; int elmLength; } eRecord;
  } val;
}sType;

typedef struct sPar_decl_list {
  int linienr;
  int dataLength;
  enum {kPar_decl_list_List, kPar_decl_list_Empty} kind;
  union {
    struct { struct sVar_decl_list *var_decl_list; } eList;
  } val;
}sPar_decl_list;

typedef struct sVar_decl_list {
  int linienr;
  enum {kVar_decl_list_List, kVar_decl_list_Var} kind;
  union {
    struct { struct sVar_decl_list *var_decl_list;
             struct sVar_type *var_type; } eList;
    struct { struct sVar_type *var_type; } eVar;
  } val;
}sVar_decl_list;

typedef struct sVar_type {
  int linienr;
  int dataOffset;
  int scopeLevel;
  int labelnr;  /* for unique id */
  union {
    struct { char *id; struct sType *type; int id_linienr;
             struct SYMBOL *symbol;} eVar;
  } val;
}sVar_type;

typedef struct sBody {
  int linienr;
  int dataLength;
  union {
    struct { struct sDecl_list *decl_list;
             struct sStatement_list *statement_list; } eBody;
  } val;
}sBody;

typedef struct sDecl_list {
  int linienr;
  enum {kDecl_list_List, kDecl_list_Empty} kind;
  union {
```

```
    struct { struct sDecl_list *decl_list;
             struct sDeclaration *declaration; } eList;
  } val;
}sDecl_list;

typedef struct sDeclaration {
  int linienr;
  enum {kDeclaration_Type, kDeclaration_Function, kDeclaration_Var} kind;
  union {
    struct { char *id; struct sType *type; int id_linienr; bool aktiv;
             struct SYMBOL *symbol; } eType;
    struct { struct sFunction *function; } eFunction;
    struct { struct sVar_decl_list *var_decl_list; } eVar;
  } val;
}sDeclaration;

typedef struct sStatement_list {
  int linienr;
  enum {kStatement_list_List, kStatement_list_Statement} kind;
  union {
    struct { struct sStatement_list *statement_list;
             struct sStatement *statement; } eList;
    struct { struct sStatement *statement; } eStatement;
  } val;
}sStatement_list;

typedef struct sStatement {
  int linienr;
  bool aktiv;   /* kan sættes false i weed-fasen */
  enum {kStatement_Return, kStatement_Write, kStatement_NewLength,
        kStatement_New, kStatement_Assign, kStatement_IfElse,
        kStatement_If, kStatement_While, kStatement_Compound,
        kStatement_ForTo, kStatement_ForDownto} kind;
  union {
    struct { struct sExpression *expression; } eReturn;
    struct { struct sExpression *expression; } eWrite;
    struct { struct sVariable *variable;
             struct sExpression *expression; } eNewLength;
    struct { struct sVariable *variable; } eNew;
    struct { struct sVariable *variable;
             struct sExpression *expression; } eAssign;
    struct { struct sExpression *expression;
             struct sStatement *ifStatement;
             struct sStatement *elseStatement; } eIfElse;
    struct { struct sExpression *expression;
             struct sStatement *ifStatement; } eIf;
    struct { struct sExpression *expression;
             struct sStatement *statement; } eWhile;
    struct { struct sStatement_list *statement_list; } eCompound;
    struct { struct sVariable *variable;
             struct sExpression *expressionInit;
             struct sExpression *expressionTo;
             struct sStatement *statement; } eForTo;
  } val;
}sStatement;


typedef struct sVariable {
  int linienr;
  struct sType *type;       /* opsamlet ved typecheck */
  enum {kVariable_Id, kVariable_Indexed, kVariable_Struct} kind;
  union {
    struct { char *id; int id_linienr; struct sVar_type *var_type; } eId;
```

```
    struct { struct sVariable *variable;
             struct sExpression *expression; } eIndexed;
    struct { struct sVariable *variable; char *id; int id_linienr;
             struct sVar_type *var_type;} eStruct;
  } val;
}sVariable;

typedef struct sExpression {
  int linienr;
  struct sType *type; /* opsamlet ved typecheck */
  enum {kExp_Eq, kExp_Neq, kExp_Lt, kExp_Gt, kExp_Lteq, kExp_Gteq,
        kExp_Add, kExp_Sub, kExp_Mul, kExp_Div, kExp_Or, kExp_And,
        kExp_Term} kind;
  union {
    struct { struct sExpression *expressionLeft;
             struct sExpression *expressionRight; } eLeftRight;
             /* fælles for binaere - dyadiske */
    struct { struct sTerm *term; } eTerm;
  } val;
}sExpression;

typedef struct sTerm {
  int linienr;
  struct sType *type; /* opsamlet ved typecheck */
  enum {kTerm_Variable, kTerm_Call, kTerm_Parantes, kTerm_Not,
        kTerm_Numeric, kTerm_Num, kTerm_True, kTerm_False,
        kTerm_Null} kind;
  union {
    struct { struct sVariable *variable; } eVariable;
    struct { char *id; int id_linienr; struct sAct_list *act_list;
             struct sHead *head; } eCall;
    struct { struct sExpression *expression; } eParantes;
    struct { struct sTerm *term; } eNot;
    struct { struct sExpression *expression; } eNumeric;
    struct { int num; } eNum;
  } val;
}sTerm;

typedef struct sAct_list {
  int linienr;
  enum {kAct_list_List, kAct_list_Empty} kind;
  union {
    struct { struct sExp_list *exp_list; } eList;
  } val;
}sAct_list;

typedef struct sExp_list {
  int linienr;
  enum {kExp_list_Expression, kExp_list_List} kind;
  union {
    struct { struct sExpression *expression;
             struct sVar_type *var_type; } eExpression;
    struct { struct sExp_list *exp_list; struct sExpression *expression;
             struct sVar_type *var_type; } eList;
  } val;
}sExp_list;
```

Funktioner til konstruktion af tony AST

tonyAST_Make.h

```
/* ------------------------------------------------------------------------
  Definition af funktioner til opbygning af TONY abstract syntax tree
```

```
  Indeholder desuden hjælpefunktioner vedr. returnering af operator som
  tekst og funktioner omhandlende typer
  Programmør: Bjørk Busch
  Historik: 2005.03.26
  Historik: 2005.05.10  kosmetik i source
-----------------------------------------------------------------------*/

sProgram *makeProgram (sBody *body);
sFunction *makeFunction (sHead *head, sBody *body, sTail *tail);
sHead *makeHead (char *id, sPar_decl_list *par_decl_list, sType *type);
sTail *makeTail (char *id);
/*-----------------------------------------------------------------*/
sType *makeType_Id (char *id);
sType *makeType_Int ();
sType *makeType_Bool ();
sType *makeType_Array (sType *type);
sType *makeType_Record (sVar_decl_list *var_decl_list);
/* NB egentlig term-type -
   bruges for at undgå ekstra struktur til expression */
sType *makeType_Null ();
/*-----------------------------------------------------------------*/
sPar_decl_list *makePar_decl_list_List (sVar_decl_list *var_decl_list);
sPar_decl_list *makePar_decl_list_Empty ();
/*-----------------------------------------------------------------*/
sVar_decl_list *makeVar_decl_list_List (sVar_decl_list *var_decl_list,
                                        sVar_type *var_type);
sVar_decl_list *makeVar_decl_list_Var (sVar_type *var_type);
/*-----------------------------------------------------------------*/
sVar_type *makeVar_type (char *id, sType *type);
/*-----------------------------------------------------------------*/
sBody *makeBody (sDecl_list *decl_list, sStatement_list *statement_list);
/*-----------------------------------------------------------------*/
sDecl_list *makeDecl_list_List (sDecl_list *decl_list,
                                sDeclaration *declaration);
sDecl_list *makeDecl_list_Empty ();
/*-----------------------------------------------------------------*/
sDeclaration *makeDeclaration_Type (char *id, sType *type);
sDeclaration *makeDeclaration_Function (sFunction *function);
sDeclaration *makeDeclaration_Var (sVar_decl_list *var_decl_list);
/*-----------------------------------------------------------------*/
sStatement_list *makeStatement_list_List (sStatement_list *statement_list,
                                          sStatement *statement);
sStatement_list *makeStatement_list_Statement (sStatement *statement);
/*-----------------------------------------------------------------*/
sStatement *makeStatement_Return (sExpression *expression);
sStatement *makeStatement_Write (sExpression *expression);
sStatement *makeStatement_NewLength (sVariable *variable,
                                     sExpression *expression);
sStatement *makeStatement_New (sVariable *variable);
sStatement *makeStatement_Assign (sVariable *variable,
                                  sExpression *expression);
sStatement *makeStatement_IfElse (sExpression *expression,
                                  sStatement *ifStatement,
                                  sStatement *elseStatement);
sStatement *makeStatement_If (sExpression *expression,
                              sStatement *ifStatement);
sStatement *makeStatement_While (sExpression *expression,
                                 sStatement *statement);
sStatement *makeStatement_Compound (sStatement_list *statement_list);
sStatement *makeStatement_ForTo (sVariable *variable,
                                 sExpression *expressionInit,
                                 int kind,
                                 sExpression *expressionTo,
```

```
                                    sStatement *statement);
/*------------------------------------------------------------------*/
sVariable *makeVariable_Id (char *id);
sVariable *makeVariable_Indexed (sVariable *variable,
                                    sExpression *expression);
sVariable *makeVariable_Struct (sVariable *variable, char *id);
/*------------------------------------------------------------------*/
sExpression *makeExpression_Dyadisk (sExpression *expressionLeft,
                                     int kExp, sExpression *expressionRight);
sExpression *makeExpression_Term (sTerm *term);
/*------------------------------------------------------------------*/
sTerm *makeTerm_Variable (sVariable *variable);
sTerm *makeTerm_Call (char *id, sAct_list *act_list);
sTerm *makeTerm_Parantes (sExpression *expression);
sTerm *makeTerm_Not (sTerm *term);
sTerm *makeTerm_Numeric (sExpression *expression);
sTerm *makeTerm_Num (int num);
sTerm *makeTerm_True ();
sTerm *makeTerm_False ();
sTerm *makeTerm_Null ();
/*------------------------------------------------------------------*/
sAct_list *makeAct_list_List (sExp_list *exp_list);
sAct_list *makeAct_list_Empty ();
/*------------------------------------------------------------------*/
sExp_list *makeExp_list_Expression (sExpression *expression);
sExp_list *makeExp_list_List (sExp_list *exp_list, sExpression *expression);

/*------------------------------------------------------------------
Funktion der returner expression operation som tekst
------------------------------------------------------------------*/
char *expression_Kind_Txt(sExpression *s);
/*------------------------------------------------------------------
Funktioner der vedører typer
------------------------------------------------------------------*/
/*------------------------------------------------------------------
Funktion der returnerer type som tekst
------------------------------------------------------------------*/
char *type_Kind_Txt(sType *s);
/*------------------------------------------------------------------
Funktion finder grund-typen
------------------------------------------------------------------*/
sType *getSlutType(sType *type);
/*------------------------------------------------------------------
Funktion der bestemmer om 2 typer er ens
------------------------------------------------------------------*/
bool equalTypes(sType *t1, sType *t2);
```

tonyAST_Make.c

```
/* ------------------------------------------------------------------------
  Realisering af funktioner til opbygning af TONY abstract syntax tree
  Programmør: Bjørk Busch
  Historik: 2005.03.26
  Historik: 2005.05.10  kosmetik i source
------------------------------------------------------------------------*/
#include <string.h>
#include "bjbu_std.h"
#include "symbol.h"
#include "tonyAbstractAsm.h"
#include "tonyAST.h"
#include "tonyAST_Make.h"

/*------------------------------------------------------------------
Fælles data på tværs af moduler
```

```
------------------------------------------------------------------*/
extern int linienr;
extern int id_linienr;
extern bool restriktivCheck;

/*-----------------------------------------------------------------
Strukturer der skal sikre singleton på primitive typer
------------------------------------------------------------------*/
sType *primitiv_Type_Int = NULL;  /* for type def på expression */
sType *primitiv_Type_Bool = NULL; /* for type def på expression */
sType *primitiv_Type_Null = NULL; /* for type def på expression */

/*-----------------------------------------------------------------
Funktioner for opbygning af AST-Strukturer
------------------------------------------------------------------*/
sProgram *makeProgram (sBody *body)
{
  sProgram *s;
  s = NEW(sProgram);
  s->linienr = linienr;
  s->symbolTable = NULL;
  s->val.eProgram.body = body;
  return s;
}
sFunction *makeFunction (sHead *head, sBody *body, sTail *tail)
{
  sFunction *s;
  s = NEW(sFunction);
  s->linienr = linienr;
  s->symbolTable = NULL;
  s->val.eFunction.head = head;
  s->val.eFunction.body = body;
  s->val.eFunction.tail = tail;
  return s;
}
sHead *makeHead (char *id, sPar_decl_list *par_decl_list, sType *type)
{
  sHead *s;
  s = NEW(sHead);
  s->linienr = linienr;
  s->val.eHead.id = id;
  s->val.eHead.id_linienr = id_linienr;
  s->val.eHead.par_decl_list = par_decl_list;
  s->val.eHead.type = type;
  return s;
}
sTail *makeTail (char *id)
{
  sTail *s;
  s = NEW(sTail);
  s->linienr = linienr;
  s->val.eTail.id = id;
  return s;
}
/*-----------------------------------------------------------------*/
sType *makeType_Id ( char *id)
{
  sType *s;
  s = NEW(sType);
  s->linienr = linienr;
  s->kind = kType_Id;
  s->val.eId.id = id;
  s->val.eId.id_linienr = id_linienr;
```

```
  s->slutType = NULL;
  /* ovrskrives/påsættes ved i forbindelse med type-check */
  return s;
}
sType *makeType_Int ()
{
  sType *s;
  if (primitiv_Type_Int == NULL) {
    s = NEW(sType);
    s->linienr = 0;
    s->kind = kType_Int;
    s->slutType = s;  /* er sluttype */
    primitiv_Type_Int = s;
  }
  else
    s = primitiv_Type_Int;
  return s;
}
sType *makeType_Bool ()
{
  sType *s;
  if (primitiv_Type_Bool == NULL) {
    s = NEW(sType);
    s->linienr = 0;
    s->kind = kType_Bool;
    s->slutType = s;  /* er sluttype */
    primitiv_Type_Bool = s;
  }
  else
    s = primitiv_Type_Bool;
  return s;
}
sType *makeType_Array ( sType *type)
{
  sType *s;
  s = NEW(sType);
  s->linienr = linienr;
  s->kind = kType_Array;
  s->val.eArray.type  = type;
  s->slutType = s;  /* er sluttype */
  return s;
}
sType *makeType_Record ( sVar_decl_list *var_decl_list)
{
  sType *s;
  s = NEW(sType);
  s->linienr = linienr;
  s->kind = kType_Record;
  s->val.eRecord.var_decl_list  = var_decl_list;
  s->val.eRecord.symbolTable = NULL;
  s->slutType = s;  /* er sluttype */
  return s;
}
sType *makeType_Null ()       /* er ikke en non-terminal, men egentlig en term-
type  */
{                 /* medtages for at udgå ekstra type-type struktur */
  sType *s;
  if (primitiv_Type_Null == NULL) {
    s = NEW(sType);
    s->linienr = 0;
    s->kind = kType_Null;
    s->slutType = s;  /* er sluttype */
    primitiv_Type_Null = s;
```

```
  }
  else
    s = primitiv_Type_Null;
  return s;
}
/*------------------------------------------------------------------*/
sPar_decl_list *makePar_decl_list_List (sVar_decl_list *var_decl_list)
{
  sPar_decl_list *s;
  s = NEW(sPar_decl_list);
  s->linienr = linienr;
  s->kind = kPar_decl_list_List;
  s->val.eList.var_decl_list  = var_decl_list;
  return s;
}
sPar_decl_list *makePar_decl_list_Empty ()
{
  sPar_decl_list *s;
  s = NEW(sPar_decl_list);
  s->linienr = linienr;
  s->kind = kPar_decl_list_Empty;
  return s;
}
/*------------------------------------------------------------------*/
sVar_decl_list *makeVar_decl_list_List (sVar_decl_list *var_decl_list,
                                        sVar_type *var_type)
{
  sVar_decl_list *s;
  s = NEW(sVar_decl_list);
  s->linienr = linienr;
  s->kind = kVar_decl_list_List;
  s->val.eList.var_decl_list = var_decl_list;
  s->val.eList.var_type = var_type;
  return s;
}
sVar_decl_list *makeVar_decl_list_Var (sVar_type *var_type)
{
  sVar_decl_list *s;
  s = NEW(sVar_decl_list);
  s->linienr = linienr;
  s->kind = kVar_decl_list_Var;
  s->val.eVar.var_type = var_type;
  return s;
}
/*------------------------------------------------------------------*/
sVar_type *makeVar_type (char *id, sType *type)
{
  sVar_type *s;
  s = NEW(sVar_type);
  s->linienr = linienr;
  s->val.eVar.id = id;
  s->val.eVar.id_linienr = id_linienr;
  s->val.eVar.type = type;
  return s;
}
/*------------------------------------------------------------------*/
sBody *makeBody (sDecl_list *decl_list, sStatement_list *statement_list)
{
  sBody *s;
  s = NEW(sBody);
  s->linienr = linienr;
  s->val.eBody.decl_list = decl_list;
  s->val.eBody.statement_list = statement_list;
```

```
  return s;
}
/*------------------------------------------------------------------*/
sDecl_list *makeDecl_list_List (sDecl_list *decl_list,
                                sDeclaration *declaration)
{
  sDecl_list *s;
  s = NEW(sDecl_list);
  s->linienr = linienr;
  s->kind = kDecl_list_List;
  s->val.eList.decl_list = decl_list;
  s->val.eList.declaration = declaration;
  return s;
}
sDecl_list *makeDecl_list_Empty ()
{
  sDecl_list *s;
  s = NEW(sDecl_list);
  s->linienr = linienr;
  s->kind = kDecl_list_Empty;
  return s;
}
/*------------------------------------------------------------------*/
sDeclaration *makeDeclaration_Type (char *id, sType *type)
{
  sDeclaration *s;
  s = NEW(sDeclaration);
  s->linienr = linienr;
  s->kind = kDeclaration_Type;
  s->val.eType.id = id;
  s->val.eType.id_linienr = id_linienr;
  s->val.eType.type = type;
  s->val.eType.aktiv = true;    /* kan deaktiveres i weed-fasen */
  return s;
}
sDeclaration *makeDeclaration_Function (sFunction *function)
{
  sDeclaration *s;
  s = NEW(sDeclaration);
  s->linienr = linienr;
  s->kind = kDeclaration_Function;
  s->val.eFunction.function = function;
  return s;
}
sDeclaration *makeDeclaration_Var (sVar_decl_list *var_decl_list)
{
  sDeclaration *s;
  s = NEW(sDeclaration);
  s->linienr = linienr;
  s->kind = kDeclaration_Var;
  s->val.eVar.var_decl_list = var_decl_list;
  return s;
}
/*------------------------------------------------------------------*/
sStatement_list *makeStatement_list_List (sStatement_list *statement_list,
                                          sStatement *statement)
{
  sStatement_list *s;
  s = NEW(sStatement_list);
  s->linienr = linienr;
  s->kind = kStatement_list_List;
  s->val.eList.statement_list = statement_list;
  s->val.eList.statement = statement;
```

```
  return s;
}
sStatement_list *makeStatement_list_Statement (sStatement *statement)
{
  sStatement_list *s;
  s = NEW(sStatement_list);
  s->linienr = linienr;
  s->kind = kStatement_list_Statement;
  s->val.eList.statement = statement;
  return s;
}
/*---------------------------------------------------------------------*/
sStatement *makeStatement_Return (sExpression *expression)
{
  sStatement *s;
  s = NEW(sStatement);
  s->linienr = linienr;
  s->aktiv = true;
  s->kind = kStatement_Return;
  s->val.eReturn.expression = expression;
  return s;
}
sStatement *makeStatement_Write (sExpression *expression)
{
  sStatement *s;
  s = NEW(sStatement);
  s->linienr = linienr;
  s->aktiv = true;
  s->kind = kStatement_Write;
  s->val.eWrite.expression = expression;
  return s;
}
sStatement *makeStatement_NewLength (sVariable *variable,
                                     sExpression *expression)
{
  sStatement *s;
  s = NEW(sStatement);
  s->linienr = linienr;
  s->aktiv = true;
  s->kind = kStatement_NewLength;
  s->val.eNewLength.variable = variable;
  s->val.eNewLength.expression = expression;
  return s;
}
sStatement *makeStatement_New (sVariable *variable)
{
  sStatement *s;
  s = NEW(sStatement);
  s->linienr = linienr;
  s->aktiv = true;
  s->kind = kStatement_New;
  s->val.eNew.variable = variable;
  return s;
}
sStatement *makeStatement_Assign (sVariable *variable, sExpression *expression)
{
  sStatement *s;
  s = NEW(sStatement);
  s->linienr = linienr;
  s->aktiv = true;
  s->kind = kStatement_Assign;
  s->val.eAssign.variable = variable;
  s->val.eAssign.expression = expression;
```

```
  return s;
}
sStatement *makeStatement_IfElse (sExpression *expression,
                                  sStatement *ifStatement,
                                  sStatement *elseStatement)
{
  sStatement *s;
  s = NEW(sStatement);
  s->linienr = linienr;
  s->aktiv = true;
  s->kind = kStatement_IfElse;
  s->val.eIfElse.expression = expression;
  s->val.eIfElse.ifStatement = ifStatement;
  s->val.eIfElse.elseStatement = elseStatement;
  return s;
}
sStatement *makeStatement_If (sExpression *expression,
                              sStatement *ifStatement)
{
  sStatement *s;
  s = NEW(sStatement);
  s->linienr = linienr;
  s->aktiv = true;
  s->kind = kStatement_If;
  s->val.eIf.expression = expression;
  s->val.eIf.ifStatement = ifStatement;
  return s;
}
sStatement *makeStatement_While (sExpression *expression,
                                 sStatement *statement)
{
  sStatement *s;
  s = NEW(sStatement);
  s->linienr = linienr;
  s->aktiv = true;
  s->kind = kStatement_While;
  s->val.eWhile.expression = expression;
  s->val.eWhile.statement = statement;
  return s;
}
sStatement *makeStatement_Compound (sStatement_list *statement_list)
{
  sStatement *s;
  s = NEW(sStatement);
  s->linienr = linienr;
  s->aktiv = true;
  s->kind = kStatement_Compound;
  s->val.eCompound.statement_list = statement_list;
  return s;
}
sStatement *makeStatement_ForTo (sVariable *variable,
                                 sExpression *expressionInit,
                                 int kind,
                                 sExpression *expressionTo,
                                 sStatement *statement)
{
  sStatement *s;
  s = NEW(sStatement);
  s->linienr = linienr;
  s->aktiv = true;
  s->kind = kind;
  s->val.eForTo.variable = variable;
  s->val.eForTo.expressionInit = expressionInit;
```

```
  s->val.eForTo.expressionTo = expressionTo;
  s->val.eForTo.statement = statement;
  return s;
}
/*-------------------------------------------------------------------*/
sVariable *makeVariable_Id (char *id)
{
  sVariable *s;
  s = NEW(sVariable);
  s->linienr = linienr;
  s->type = NULL;   /* påsættes ved symbol-check */
  s->kind = kVariable_Id;
  s->val.eId.id = id;
  s->val.eId.id_linienr = id_linienr;
  s->val.eId.var_type = NULL; /* påsættels i symbol-fasen */
  return s;
}
sVariable *makeVariable_Indexed (sVariable *variable,
                                 sExpression *expression)
{
  sVariable *s;
  s = NEW(sVariable);
  s->linienr = linienr;
  s->type = NULL;   /* påsættes ved symbol-check */
  s->kind = kVariable_Indexed;
  s->val.eIndexed.variable = variable;
  s->val.eIndexed.expression = expression;
  return s;
}
sVariable *makeVariable_Struct (sVariable *variable, char *id)
{
  sVariable *s;
  s = NEW(sVariable);
  s->linienr = linienr;
  s->type = NULL;   /* påsættes ved symbol-check */
  s->kind = kVariable_Struct;
  s->val.eStruct.variable = variable;
  s->val.eStruct.id = id;
  s->val.eStruct.id_linienr = id_linienr;
  s->val.eStruct.var_type = NULL; /* påsættels i symbol-fasen */
  return s;
}
/*-------------------------------------------------------------------*/
sExpression *makeExpression_Dyadisk (sExpression *expressionLeft,
                                     int kExp,
                                     sExpression *expressionRight)
{
  sExpression *s;
  s = NEW(sExpression);
  s->linienr = linienr;
  s->type = NULL;   /* påsættes ved symbol-check */
  s->kind = kExp;
  s->val.eLeftRight.expressionLeft = expressionLeft;
  s->val.eLeftRight.expressionRight = expressionRight;
  return s;
}
sExpression *makeExpression_Term (sTerm *term)
{
  sExpression *s;
  s = NEW(sExpression);
  s->linienr = linienr;
  s->type = NULL;   /* påsættes ved symbol-check */
  s->kind = kExp_Term;
```

```
  s->val.eTerm.term = term;
  return s;
}
/*------------------------------------------------------------------*/
sTerm *makeTerm_Variable (sVariable *variable)
{
  sTerm *s;
  s = NEW(sTerm);
  s->linienr = linienr;
  s->type = NULL;   /* påsættes ved symbol-check */
  s->kind = kTerm_Variable;
  s->val.eVariable.variable = variable;
  return s;
}
sTerm *makeTerm_Call (char *id, sAct_list *act_list)
{
  sTerm *s;
  s = NEW(sTerm);
  s->linienr = linienr;
  s->type = NULL;   /* påsættes ved symbol-check */
  s->kind = kTerm_Call;
  s->val.eCall.id = id;
  s->val.eCall.id_linienr = id_linienr;
  s->val.eCall.act_list = act_list;
  s->val.eCall.head = NULL;
  return s;
}
sTerm *makeTerm_Parantes (sExpression *expression)
{
  sTerm *s;
  s = NEW(sTerm);
  s->linienr = linienr;
  s->type = NULL;   /* påsættes ved symbol-check */
  s->kind = kTerm_Parantes;
  s->val.eParantes.expression = expression;
  return s;
}
sTerm *makeTerm_Not (sTerm *term)
{
  sTerm *s;
  s = NEW(sTerm);
  s->linienr = linienr;
  s->type = NULL;   /* påsættes ved symbol-check */
  s->kind = kTerm_Not;
  s->val.eNot.term = term;
  return s;
}
sTerm *makeTerm_Numeric (sExpression *expression)
{
  sTerm *s;
  s = NEW(sTerm);
  s->linienr = linienr;
  s->type = makeType_Int(); /* fast basis type*/
  s->kind = kTerm_Numeric;
  s->val.eNumeric.expression = expression;
  return s;
}
sTerm *makeTerm_Num (int num)
{
  sTerm *s;
  s = NEW(sTerm);
  s->linienr = linienr;
  s->type = makeType_Int(); /* fast basis type */
```

```
  s->kind = kTerm_Num;
  s->val.eNum.num = num;
  return s;
}
sTerm *makeTerm_NegativNum (int num)
{
  sTerm *s;
  s = NEW(sTerm);
  s->linienr = linienr;
  s->type = makeType_Int(); /* fast basis type */
  s->kind = kTerm_Num;
  s->val.eNum.num = -num;
  return s;
}
sTerm *makeTerm_True ()
{
  sTerm *s;
  s = NEW(sTerm);
  s->linienr = linienr;
  s->type = makeType_Bool();  /* fast basis type */
  s->kind = kTerm_True;
  return s;
}
sTerm *makeTerm_False ()
{
  sTerm *s;
  s = NEW(sTerm);
  s->linienr = linienr;
  s->type = makeType_Int(); /* fast basis type */
  s->kind = kTerm_False;
  return s;
}
sTerm *makeTerm_Null ()
{
  sTerm *s;
  s = NEW(sTerm);
  s->linienr = linienr;
  s->type = makeType_Null();  /* fast basis type */
  s->kind = kTerm_Null;
  return s;
}
/*------------------------------------------------------------------*/
sAct_list *makeAct_list_List (sExp_list *exp_list)
{
  sAct_list *s;
  s = NEW(sAct_list);
  s->linienr = linienr;
  s->kind = kAct_list_List;
  s->val.eList.exp_list = exp_list;
  return s;
}
sAct_list *makeAct_list_Empty ()
{
  sAct_list *s;
  s = NEW(sAct_list);
  s->linienr = linienr;
  s->kind = kAct_list_Empty;
  return s;
}
/*------------------------------------------------------------------*/
sExp_list *makeExp_list_List (sExp_list *exp_list, sExpression *expression)
{
  sExp_list *s;
```

```
  s = NEW(sExp_list);
  s->linienr = linienr;
  s->kind = kExp_list_List;
  s->val.eList.exp_list = exp_list;
  s->val.eList.expression = expression;
  return s;
}
sExp_list *makeExp_list_Expression (sExpression *expression)
{
  sExp_list *s;
  s = NEW(sExp_list);
  s->linienr = linienr;
  s->kind = kExp_list_Expression;
  s->val.eExpression.expression = expression;
  return s;
}
/*-------------------------------------------------------------
Funktion der returner expression operation som tekst
-------------------------------------------------------------*/
char *expression_Kind_Txt(sExpression *s)
{
  if (s == NULL)
    return "MANGLER EXP";
  switch(s->kind)
  {
    case kExp_Eq:     return "==";
    case kExp_Neq:    return "!=";
    case kExp_Lt:     return "<";
    case kExp_Gt:     return ">";
    case kExp_Lteq:   return "<=";
    case kExp_Gteq:   return ">=";
    case kExp_Add:    return "+";
    case kExp_Sub:    return "-";
    case kExp_Mul:    return "*";
    case kExp_Div:    return "/";
    case kExp_Or:     return "||";
    case kExp_And:    return "&&";
    case kExp_Term:   return "Term";
    default:          return "exp???";
  }
}
/*-------------------------------------------------------------
Funktioner der vedører typer
-------------------------------------------------------------*/
/*-------------------------------------------------------------
Funktion der returnerer type som tekst
-------------------------------------------------------------*/
char *type_Kind_Txt(sType *s)
{
  if (s == NULL)
    return "???";
  switch(s->kind)
  {
    case kType_Id:    return s->val.eId.id;
    case kType_Int:   return "int";
    case kType_Bool:  return "bool";
    case kType_Array: return "array";
    case kType_Record:return "record";
    case kType_Null:  return "null";
    default:          return "????";
  }
}
/*-------------------------------------------------------------
```

```
Funktion finder grund-typen
------------------------------------------------------------*/
sType *getSlutType(sType *type)
{
  if (type == NULL) return NULL;
  return type->slutType;
}
/*------------------------------------------------------------
Funktion der bestemmer om 2 typer er ens
------------------------------------------------------------*/
bool equalTypes(sType *t1, sType *t2) {
  sType *slutType;
  sType *slutType2;
  if (t1 == NULL || t2 == NULL)
    return false;
  if (t1 == t2)
    return true;
  if (t1->kind == kType_Id && t2->kind == kType_Id)
    return strcmp(t1->val.eId.id,t2->val.eId.id) == 0;
  if (t1->kind == kType_Null) {
    slutType = getSlutType(t2);
    if (slutType == NULL)
      return false;
    if (slutType->kind == kType_Record)
      return true;
    if (slutType->kind == kType_Array)
      return true;
    return false;
  }
  if (t2->kind == kType_Null) {
    slutType = getSlutType(t1);
    if (slutType == NULL)
      return false;
    if (slutType->kind == kType_Record)
      return true;
    if (slutType->kind == kType_Array)
      return true;
    return false;
  }
  if (restriktivCheck)  return false;
  slutType = getSlutType(t1);
  slutType2 = getSlutType(t1);
  if (slutType == NULL || slutType2 == NULL)
    return false;
  if (slutType == slutType2)
    return true;
  return false;
}
```

Weed fasen

Funktioner til weed fasen

tonyWeed.c

```
/* ------------------------------------------------------------------
  Realisering af funktioner til test af funktions id og returns
  Funktioner returnere true, hvis der er garanti for et
  return statement
  Statements der ikke kan nås p.g.a. return forud deaktiveres, så de
  kan udelades i de efterfølgende faser
  Programmør: Bjørk Busch
  Historik: 2005.03.26
  Historik: 2005.05.10  kosmetik i source
------------------------------------------------------------------*/
#include <stdio.h>
#include "bjbu_std.h"
#include "symbol.h"
#include "tonyAbstractAsm.h"
#include "tonyAST.h"

/*-----------------------------------------------------------------
Fælles data på tværs af moduler
-----------------------------------------------------------------*/
extern int alvorligFejl;
extern int fejl;
extern int advarsel;
extern FILE *msgFile;


/*-----------------------------------------------------------------*/
void weedProgram (sProgram *s);
void weedFunction (sFunction *s);
/*-----------------------------------------------------------------*/
bool weedBody (sBody *s);
/*-----------------------------------------------------------------*/
void weedDecl_list (sDecl_list *s);
void weedDecl_list_List (sDecl_list *s);
void weedDecl_list_Empty (sDecl_list *s);
/*-----------------------------------------------------------------*/
void weedDeclaration (sDeclaration *s);
void weedDeclaration_Type (sDeclaration *s);
void weedDeclaration_Function (sDeclaration *s);
/*-----------------------------------------------------------------*/
bool weedStatement_list (sStatement_list *s, bool returnBefore);
bool weedStatement_list_List (sStatement_list *s, bool returnBefore);
bool weedStatement_list_Statement (sStatement_list *s, bool returnBefore);
/*-----------------------------------------------------------------*/
bool weedStatement (sStatement *s, bool returnBefore);
bool weedStatement_Return (sStatement *s, bool returnBefore);
bool weedStatement_Write (sStatement *s, bool returnBefore);
bool weedStatement_NewLength (sStatement *s, bool returnBefore);
bool weedStatement_New (sStatement *s, bool returnBefore);
bool weedStatement_Assign (sStatement *s, bool returnBefore);
bool weedStatement_IfElse (sStatement *s, bool returnBefore);
bool weedStatement_If (sStatement *s, bool returnBefore);
bool weedStatement_While (sStatement *s, bool returnBefore);
bool weedStatement_Compound (sStatement *s, bool returnBefore);
bool weedStatement_ForTo (sStatement *s, bool returnBefore);
/*-----------------------------------------------------------------*/
```

```
void weedProgram (sProgram *s)
{
  fprintf(msgFile, "TONY weeding fase start\n");
  weedBody(s->val.eProgram.body);
  fprintf(msgFile, "TONY weeding fase slut\n");
}
void weedFunction (sFunction *s)
{
  if (!weedBody(s->val.eFunction.body))
  {
    fprintf(msgFile,
      "alvorligFejl: funktion <%s> linie:<%d> mangler return statement\n",
      s->val.eFunction.head->val.eHead.id, s->val.eFunction.head->linienr);
    ++alvorligFejl;
  }
  if ( strcmp(s->val.eFunction.head->val.eHead.id,
        s->val.eFunction.tail->val.eTail.id) != 0 )
  {
    fprintf(msgFile,
      "%d: Advarsel funktion afslutter ikke med samme navn <%s> som den er \
defineret med <%s> i linie:<%d>\n",
      s->val.eFunction.tail->linienr,s->val.eFunction.tail->val.eTail.id,
      s->val.eFunction.head->val.eHead.id, s->val.eFunction.head->linienr);
    ++fejl;
  }
}
/*------------------------------------------------------------------*/
bool weedBody (sBody *s)
{
  weedDecl_list(s->val.eBody.decl_list);
  return weedStatement_list(s->val.eBody.statement_list, false);
}
/*------------------------------------------------------------------*/
void weedDecl_list (sDecl_list *s)
{
  switch(s->kind)
  {
    case kDecl_list_List:  weedDecl_list_List  (s);break;
    case kDecl_list_Empty: weedDecl_list_Empty (s);break;
  }
}
void weedDecl_list_List (sDecl_list *s)
{
  weedDecl_list(s->val.eList.decl_list);
  weedDeclaration(s->val.eList.declaration);
}
void weedDecl_list_Empty (sDecl_list *s)
{
}
/*------------------------------------------------------------------*/
void weedDeclaration (sDeclaration *s)
{
  switch(s->kind)
  {
    case kDeclaration_Type:     weedDeclaration_Type    (s);break;
    case kDeclaration_Function: weedDeclaration_Function(s);break;
    case kDeclaration_Var:      break;
  }
}

void weedDeclaration_Type (sDeclaration *s)
{
```

```
  if (s->val.eType.type->kind == kType_Id) {
    if (strcmp(s->val.eType.id, s->val.eType.type->val.eId.id ) == 0 ) {
      s->val.eType.aktiv = false;
      fprintf(msgFile, "%d: Fejl navn<%s> og type er identisk\n",
        s->val.eType.id_linienr,s->val.eType.id);
      ++fejl;
    }
  }
}

void weedDeclaration_Function (sDeclaration *s)
{
  weedFunction(s->val.eFunction.function);
}
/*-------------------------------------------------------------------*/
bool weedStatement_list (sStatement_list *s, bool returnBefore)
{
  switch(s->kind)
  {
    case kStatement_list_List:
        return weedStatement_list_List      (s,returnBefore);break;
    case kStatement_list_Statement:
        return weedStatement_list_Statement (s,returnBefore);break;
  }
}
bool weedStatement_list_List (sStatement_list *s, bool returnBefore)
{
  /* ok hvis bare listen eller statement garanterer return */
  bool retListe;
  bool retStm;
  retListe = weedStatement_list(s->val.eList.statement_list, returnBefore);
  retStm = weedStatement(s->val.eList.statement, retListe);
  return retListe || retStm;
}
bool weedStatement_list_Statement (sStatement_list *s, bool returnBefore)
{
  return weedStatement(s->val.eList.statement, returnBefore);
}
/*-------------------------------------------------------------------*/
bool weedStatement (sStatement *s, bool returnBefore)
{
  bool ret;
  if (returnBefore){
    fprintf(msgFile,
"%d: Advarsel statement vil aldrig blive udfoert p.g.a. tidligere return\n",
      s->linienr);
    ++advarsel;
  }
  switch(s->kind)
  {
    case kStatement_Return:
        ret = weedStatement_Return    (s,returnBefore);break;
    case kStatement_Write:
        ret = weedStatement_Write     (s,returnBefore);break;
    case kStatement_NewLength:
        ret = weedStatement_NewLength (s,returnBefore);break;
    case kStatement_New:
        ret = weedStatement_New       (s,returnBefore);break;
    case kStatement_Assign:
        ret = weedStatement_Assign    (s,returnBefore);break;
    case kStatement_IfElse:
        ret = weedStatement_IfElse    (s,returnBefore);break;
    case kStatement_If:
```

```
        ret = weedStatement_If        (s,returnBefore);break;
    case kStatement_While:
        ret = weedStatement_While     (s,returnBefore);break;
    case kStatement_Compound:
        ret = weedStatement_Compound  (s,returnBefore);break;
    case kStatement_ForTo:
        ret = weedStatement_ForTo     (s,returnBefore);break;
    case kStatement_ForDownto:
        ret = weedStatement_ForTo     (s,returnBefore);break;
  }
  if (returnBefore)
    s->aktiv = false;
  return ret;
}
bool weedStatement_Return (sStatement *s, bool returnBefore)
{
  return true;
}
bool weedStatement_Write (sStatement *s, bool returnBefore)
{
  return returnBefore;
}
bool weedStatement_NewLength (sStatement *s, bool returnBefore)
{
  return returnBefore;
}
bool weedStatement_New (sStatement *s, bool returnBefore)
{
  return returnBefore;
}
bool weedStatement_Assign (sStatement *s, bool returnBefore)
{
  return returnBefore;
}
bool weedStatement_IfElse (sStatement *s, bool returnBefore)
{
  /* ok hvis både if og else statement garanterer return
   eller der er garanteret et return tidligere */
  return returnBefore ||
     (weedStatement(s->val.eIfElse.ifStatement, returnBefore)
      && weedStatement(s->val.eIfElse.elseStatement, returnBefore));
}
bool weedStatement_If (sStatement *s, bool returnBefore)
{
  weedStatement(s->val.eIf.ifStatement, returnBefore);
  return returnBefore;
}
bool weedStatement_While (sStatement *s, bool returnBefore)
{
  return returnBefore;
}
bool weedStatement_Compound (sStatement *s, bool returnBefore)
{
  return weedStatement_list(s->val.eCompound.statement_list, returnBefore);
}
bool weedStatement_ForTo (sStatement *s, bool returnBefore)
{
  return returnBefore;
}
/*-------------------------------------------------------------------*/
```

Symbol fasen

Strukturer og funktioner til manipulation af symboltabel

symbol.h

symbol.h
```
#define HashSize 317
#define SYMBOL_DUPLIKAT 0    /* 1: tilader duplikat symboler i samme tabel */

/*-------------------------------------------------------------------
  Definition af strukturer
  Historik: 2005.05.10  kosmetik i source
-------------------------------------------------------------------*/
typedef struct XREF  /* til reference af hvor symbol anvendes */
{
  int linienr;
  struct XREF *next;
}XREF;

typedef struct SYMBOL {
  int linienr;
  char *name;
  enum {symType, symHead, symVar} kind;  /* reference til */
  void *astNode;              /* stype, sHead, sVar_Type */
  int symbolBuildDeclare_bloknr;      /* kan bruges til at se om def. før eller
efter */
  int antalXref;
  struct XREF *xref;
  struct SYMBOL *next;
} SYMBOL;

typedef struct SymbolTable {
  int level;
  int symboltabelnr;
  SYMBOL *table[HashSize];
  struct SymbolTable *next;
} SymbolTable;

/*-------------------------------------------------------------------
Definition af funktioner
-------------------------------------------------------------------*/

int Hash(char *str);
/* Ekstra funktioner  til beregning af et gyldigt index til hash-tabel */
int HashIndex(char *str);

SymbolTable *initSymbolTable(SymbolTable *nextSymbolTable);
SymbolTable *scopeSymbolTable(SymbolTable *t);
SYMBOL *putSymbol(SymbolTable *t, char *name, int kind, void *astNode, int sym-
bolBuildDeclare_bloknr, int linienr);
SYMBOL *getSymbol(SymbolTable *t, char *name, int linienr);
/*void makeSymbolXref (SYMBOL *s, int linienr);*/
```

symbol.c

```
/* -----------------------------------------------------------------
  Definitioner til symboltabel
  Programmør: Bjørk Busch
  Historik: 2005.05.10  kosmetik i source
-----------------------------------------------------------------*/
#include <stdio.h>
#include "bjbu_std.h"
```

```
#include "symbol.h"
/*------------------------------------------------------------
Fælles data på tværs af moduler
------------------------------------------------------------*/
extern int alvorligFejl;
extern FILE *msgFile;

/*------------------------------------------------------------
Fælles data for dette modul
------------------------------------------------------------*/
int symboltabelnr = 0;


/*------------------------------------------------------------
Implementering af Hash funktion
------------------------------------------------------------*/
int Hash(char *str){
  int i = 0;
  int hashCode = 0;

  for (i = 0; str[i] != 0; i++) {
    hashCode = (hashCode<<1) + str[i];
  }
  return hashCode;
}
int HashIndex(char *str) {
  return (Hash(str)%HashSize);
}

/*------------------------------------------------------------------------
Initerer en ny symboltabel med reference til næste niveau opad
------------------------------------------------------------------------*/
SymbolTable *initSymbolTable(SymbolTable *nextSymbolTable)
{
  int i;
  SymbolTable *symbTab = NEW(SymbolTable);
  ++symboltabelnr;
  symbTab->symboltabelnr = symboltabelnr;
  symbTab->next = nextSymbolTable;
  if (nextSymbolTable != NULL)
    symbTab->level = nextSymbolTable->level + 1;
  else
    symbTab->level = 0;
  for (i=0; i < HashSize; i++)
    symbTab->table[i] = NULL;
  return symbTab;
}

/*------------------------------------------------------------------------
Returner næste "niveau" af Symboltabel.
------------------------------------------------------------------------*/
SymbolTable *scopeSymbolTable(SymbolTable *t) {
  if (t != NULL) t = t->next;
  return t;
}

/*------------------------------------------------------------------------
Nedenstående finder SYMBOL  med name i den angivne SymbolTable
Bruges både i putSymbol og i getSymbol funktionerne
------------------------------------------------------------------------*/
SYMBOL *getSymbolInSymbolTable(SymbolTable *t, char *name) {
  SYMBOL *s = NULL;
  if (t == NULL)
```

```
    return NULL;
  s = t->table[HashIndex(name)];
  while (s != NULL) {
    if (strcmp(s->name, name) == 0)
      return s;
    s = s->next;
  }
  return NULL;
}

/*---------------------------------------------------------------------
Nedenstående implementerer  voud makeSymbolXref(Symbol *s, int linienr)
Funktionen bruges til at opsætte en reference til hvor symbol anvendes
---------------------------------------------------------------------*/
void makeNewSymbolXref (SYMBOL *s, int linienr) {
  XREF *x = NEW(XREF);
  x->linienr = linienr;
  x->next = s->xref;
  s->xref = x;
  s->antalXref = s->antalXref + 1;
}

/*---------------------------------------------------------------------
Nedenstående implementerer  SYMBOL *getSymbol
---------------------------------------------------------------------*/
SYMBOL *getSymbol(SymbolTable *t, char *name, int linienr) {
  SYMBOL *s;
  int index;
  while (t != NULL) {
    s = getSymbolInSymbolTable(t, name);
    if (s != NULL) {
      makeNewSymbolXref (s, linienr);
      return s;
    }
    t = t->next;
  }
  return NULL;
}

/*---------------------------------------------------------------------
Nedenstående implementerer  SYMBOL *putSymbol
---------------------------------------------------------------------*/
SYMBOL *putSymbol(SymbolTable *t, char *name, int kind, void *astNode, int sym-
bolBuildDeclare_bloknr, int linienr)
{
  SYMBOL *newSymbol;
  int index;
#if SYMBOL_DUPLIKAT
  newSymbol = NULL;
#else
  newSymbol = getSymbolInSymbolTable(t, name);
#endif
  if ( newSymbol != NULL)
  {
    fprintf(msgFile, "alvorligFejl - symbol allerede defineret i scope name<%s>
linie<%d>\n",name,newSymbol->linienr);
    ++alvorligFejl;
    newSymbol = NULL;
  }
  else {
    index = HashIndex(name);
    newSymbol = NEW(SYMBOL);
    newSymbol->name = name;    /* det fremgår ikke om der skal kopi til */
```

```
    newSymbol->kind = kind;
    newSymbol->astNode = astNode;
    newSymbol->symbolBuildDeclare_bloknr = symbolBuildDeclare_bloknr;
    newSymbol->linienr = linienr;
    newSymbol->antalXref = 0;
    newSymbol->xref = NULL;
    newSymbol->next = t->table[index];
    t->table[index] = newSymbol;
  }
  return newSymbol;
}
```

Opbygning af symboltabel fra AST

tonySymbols_Build.c

```
/* ------------------------------------------------------------------
  Realisering af funktioner til opbygning af symboltabeller
  Programmør: Bjørk Busch
  Historik: 2005.03.26
  Historik: 2005.05.10  kosmetik i source
------------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include "bjbu_std.h"
#include "symbol.h"
#include "tonyAbstractAsm.h"
#include "tonyAST.h"

/*-----------------------------------------------------------------
Fælles data på tværs af moduler
-----------------------------------------------------------------*/
extern int alvorligFejl;
extern int fejl;
extern FILE *msgFile;

/*-----------------------------------------------------------------
Fælles data for dette moduler
-----------------------------------------------------------------*/
/* variabel der bruges til kontrol af om decleration er forud  for brug */
int symbolBuildDeclare_bloknr = 0;
struct SymbolTable *buildSymbolTable;
/*-----------------------------------------------------------------
Funktioner for opbygning af symboltabel
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------*/
void symbolsBuildProgram (sProgram *s);
void symbolsBuildFunction (sFunction *s);
void symbolsBuildHead (sHead *s);
/*-----------------------------------------------------------------*/
void symbolsBuildType (sType *s);
void symbolsBuildType_Id (sType *s);
void symbolsBuildType_Int (sType *s);
void symbolsBuildType_Bool (sType *s);
void symbolsBuildType_Array (sType *s);
void symbolsBuildType_Record (sType *s);
/*-----------------------------------------------------------------*/
void symbolsBuildPar_decl_list (sPar_decl_list *s);
void symbolsBuildPar_decl_list_List (sPar_decl_list *s);
void symbolsBuildPar_decl_list_Empty (sPar_decl_list *s);
/*-----------------------------------------------------------------*/
void symbolsBuildVar_decl_list (sVar_decl_list *s);
```

```
void symbolsBuildVar_decl_list_List (sVar_decl_list *s);
void symbolsBuildVar_decl_list_Var (sVar_decl_list *s);
/*-------------------------------------------------------------------*/
void symbolsBuildVar_type (sVar_type *s);
/*-------------------------------------------------------------------*/
void symbolsBuildBody (sBody *s);
/*-------------------------------------------------------------------*/
void symbolsBuildDecl_list (sDecl_list *s);
void symbolsBuildDecl_list_List (sDecl_list *s);
void symbolsBuildDecl_list_Empty (sDecl_list *s);
/*-------------------------------------------------------------------*/
void symbolsBuildDeclaration (sDeclaration *s);
void symbolsBuildDeclaration_Type (sDeclaration *s);
void symbolsBuildDeclaration_Function (sDeclaration *s);
void symbolsBuildDeclaration_Var (sDeclaration *s);
/*-------------------------------------------------------------------*/

void symbolsBuildProgram (sProgram *s)
{
  fprintf(msgFile, "TONY opbygning af symboltabel start\n");
  buildSymbolTable = initSymbolTable(NULL);   /* opret ny symboltabel */
  s->symbolTable = buildSymbolTable;          /* gem symboltabel i AST */
  symbolsBuildBody(s->val.eProgram.body);
  fprintf(msgFile, "TONY opbygning af symboltabel slut\n");
}
void symbolsBuildFunction (sFunction *s)
{
  /* gem adresse på aktuel symboltabel */
  SymbolTable *saveSymbolTable = buildSymbolTable;
  /* opret ny symboltabel - nyt level */
  buildSymbolTable = initSymbolTable(buildSymbolTable);
  /* gem symboltabel i AST */
  s->symbolTable = buildSymbolTable;
  symbolsBuildHead(s->val.eFunction.head);
    symbolsBuildBody(s->val.eFunction.body);
  /* reetabler adresse på aktuel symboltabel */
  buildSymbolTable = saveSymbolTable;
}
void symbolsBuildHead (sHead *s)
{
  symbolsBuildPar_decl_list(s->val.eHead.par_decl_list);
}
/*-------------------------------------------------------------------*/
void symbolsBuildType (sType *s)
{
  switch(s->kind)
  {
    case kType_Id:     symbolsBuildType_Id    (s);break;
    case kType_Int:    symbolsBuildType_Int   (s);break;
    case kType_Bool:   symbolsBuildType_Bool  (s);break;
    case kType_Array:  symbolsBuildType_Array (s);break;
    case kType_Record: symbolsBuildType_Record(s);break;
  }
}

void symbolsBuildType_Id (sType *s)
{
}
void symbolsBuildType_Int (sType *s)
{
}
void symbolsBuildType_Bool (sType *s)
{
```

```
}
void symbolsBuildType_Array (sType *s)
{
    symbolsBuildType(s->val.eArray.type);
}
void symbolsBuildType_Record (sType *s)
{
  /* gem adresse på aktuel symboltabel */
  SymbolTable *saveSymbolTable = buildSymbolTable;
  /* opret ny symboltabel - nyt level uden ref. op */
  buildSymbolTable = initSymbolTable(NULL);
  /* gem symboltabel i AST */
  s->val.eRecord.symbolTable = buildSymbolTable;
  symbolsBuildVar_decl_list(s->val.eRecord.var_decl_list);
  /* reetabler adresse på aktuel symboltabel */
  buildSymbolTable = saveSymbolTable;
}
/*-----------------------------------------------------------------*/
void symbolsBuildPar_decl_list (sPar_decl_list *s)
{
  ++symbolBuildDeclare_bloknr;
  switch(s->kind)
  {
    case kPar_decl_list_List: symbolsBuildPar_decl_list_List  (s);break;
    case kPar_decl_list_Empty:symbolsBuildPar_decl_list_Empty (s);break;
  }
}
void symbolsBuildPar_decl_list_List (sPar_decl_list *s)
{
  symbolsBuildVar_decl_list(s->val.eList.var_decl_list);
}
void symbolsBuildPar_decl_list_Empty (sPar_decl_list *s)
{
}
/*-----------------------------------------------------------------*/
void symbolsBuildVar_decl_list (sVar_decl_list *s)
{
  switch(s->kind)
  {
    case kVar_decl_list_List: symbolsBuildVar_decl_list_List(s);break;
    case kVar_decl_list_Var:  symbolsBuildVar_decl_list_Var (s);break;
  }
}
void symbolsBuildVar_decl_list_List (sVar_decl_list *s)
{
  symbolsBuildVar_decl_list(s->val.eList.var_decl_list);
  symbolsBuildVar_type(s->val.eList.var_type);
}
void symbolsBuildVar_decl_list_Var (sVar_decl_list *s)
{
  symbolsBuildVar_type(s->val.eVar.var_type);
}
/*-----------------------------------------------------------------*/
void symbolsBuildVar_type (sVar_type *s)
{
  s->val.eVar.symbol = putSymbol(buildSymbolTable, s->val.eVar.id,
                          symVar, s, symbolBuildDeclare_bloknr,
                          s->val.eVar.id_linienr);
  symbolsBuildType(s->val.eVar.type);
}
/*-----------------------------------------------------------------*/
void symbolsBuildBody (sBody *s)
{
```

```
  symbolsBuildDecl_list(s->val.eBody.decl_list);
}
/*------------------------------------------------------------------*/
void symbolsBuildDecl_list (sDecl_list *s)
{
  switch(s->kind)
  {
    case kDecl_list_List:  symbolsBuildDecl_list_List  (s);break;
    case kDecl_list_Empty: symbolsBuildDecl_list_Empty (s);break;
  }
}
void symbolsBuildDecl_list_List (sDecl_list *s)
{
  symbolsBuildDecl_list(s->val.eList.decl_list);
  symbolsBuildDeclaration(s->val.eList.declaration);
}
void symbolsBuildDecl_list_Empty (sDecl_list *s)
{
}
/*------------------------------------------------------------------*/
void symbolsBuildDeclaration (sDeclaration *s)
{
  ++symbolBuildDeclare_bloknr;
  switch(s->kind)
  {
    case kDeclaration_Type:     symbolsBuildDeclaration_Type    (s);break;
    case kDeclaration_Function: symbolsBuildDeclaration_Function(s);break;
    case kDeclaration_Var:      symbolsBuildDeclaration_Var     (s);break;
  }
}
void symbolsBuildDeclaration_Type (sDeclaration *s)
{
  if (!s->val.eType.aktiv) return;  /* deaktiveret i weed-fase */
  s->val.eType.symbol = putSymbol(buildSymbolTable, s->val.eType.id,
                            symType, s, symbolBuildDeclare_bloknr,
                            s->val.eType.id_linienr);
  symbolsBuildType(s->val.eType.type);
}
void symbolsBuildDeclaration_Function (sDeclaration *s)
{
  s->val.eFunction.function->val.eFunction.head->val.eHead.symbol =
    putSymbol(buildSymbolTable,
      s->val.eFunction.function->val.eFunction.head->val.eHead.id,
      symHead,
      s->val.eFunction.function->val.eFunction.head,
      symbolBuildDeclare_bloknr,
      s->val.eFunction.function->val.eFunction.head->val.eHead.id_linienr);
  symbolsBuildFunction(s->val.eFunction.function);
}
void symbolsBuildDeclaration_Var (sDeclaration *s)
{
  symbolsBuildVar_decl_list(s->val.eVar.var_decl_list);
}
/*------------------------------------------------------------------*/
```

Type check fasen

Funktioner til typecheck

tonySymbols_Check.c

```
/* ------------------------------------------------------------------
  Realisering af funktioner til opbygning af symboltabeller
  Programmør: Bjørk Busch
  Historik: 2005.03.26
  Historik: 2005.05.10  kosmetik i source
------------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include "bjbu_std.h"
#include "symbol.h"
#include "tonyAbstractAsm.h"
#include "tonyAST.h"

/*------------------------------------------------------------------
Fælles data på tværs af moduler
------------------------------------------------------------------*/
extern int alvorligFejl;
extern int fejl;
extern int advarsel;
extern FILE *msgFile;
extern bool restriktivCheck;
extern bool recordArray;
/*------------------------------------------------------------------
Funktioner for typer
Er implementeret i tonyAST_Make.c
------------------------------------------------------------------*/
extern sType *makeType_Int ();
extern sType *makeType_Bool ();
/* NB egentlig term-type -
  bruges for at undgå ekstra struktur til expression */
extern sType *makeType_Null ();
extern char *expression_Kind_Txt(sExpression *s);
extern char *type_Kind_Txt(sType *s);
extern sType *getSlutType(sType *type);
extern bool equalTypes(sType *t1, sType *t2);

/*------------------------------------------------------------------
Fælles data for dette moduler
------------------------------------------------------------------*/
/* variabel der bruges til kontrol af om declaration er forud  for brug */
int symbolCheckDeclare_bloknr = 0;
struct SymbolTable *checkSymbolTable;

/*------------------------------------------------------------------
Funktioner for check af typer
------------------------------------------------------------------*/

void symbolsCheckProgram (sProgram *s);
void symbolsCheckFunction (sFunction *s);
void symbolsCheckHead (sHead *s);
/*------------------------------------------------------------------*/
void symbolsCheckType (sType *s);
void symbolsCheckType_Id (sType *s);
void symbolsCheckType_Int (sType *s);
void symbolsCheckType_Bool (sType *s);
void symbolsCheckType_Array (sType *s);
```

```
void symbolsCheckType_Record (sType *s);
/*-------------------------------------------------------------*/
void symbolsCheckPar_decl_list (sPar_decl_list *s);
void symbolsCheckPar_decl_list_List (sPar_decl_list *s);
void symbolsCheckPar_decl_list_Empty (sPar_decl_list *s);
/*-------------------------------------------------------------*/
void symbolsCheckVar_decl_list (sVar_decl_list *s);
void symbolsCheckVar_decl_list_List (sVar_decl_list *s);
void symbolsCheckVar_decl_list_Var (sVar_decl_list *s);
/*-------------------------------------------------------------*/
void symbolsCheckVar_type (sVar_type *s);
/*-------------------------------------------------------------*/
void symbolsCheckBody (sBody *s);
/*-------------------------------------------------------------*/
void symbolsCheckDecl_list (sDecl_list *s);
void symbolsCheckDecl_list_List (sDecl_list *s);
void symbolsCheckDecl_list_Empty (sDecl_list *s);
/*-------------------------------------------------------------*/
void symbolsCheckDeclaration (sDeclaration *s);
void symbolsCheckDeclaration_Type (sDeclaration *s);
void symbolsCheckDeclaration_Function (sDeclaration *s);
void symbolsCheckDeclaration_Var (sDeclaration *s);
/*-------------------------------------------------------------*/

/*-------------------------------------------------------------*/
void symbolsCheckStatement_list (sStatement_list *s);
void symbolsCheckStatement_list_List (sStatement_list *s);
void symbolsCheckStatement_list_Statement (sStatement_list *s);
/*-------------------------------------------------------------*/
void symbolsCheckStatement (sStatement *s);
void symbolsCheckStatement_Return (sStatement *s);
void symbolsCheckStatement_Write (sStatement *s);
void symbolsCheckStatement_NewLength (sStatement *s);
void symbolsCheckStatement_New (sStatement *s);
void symbolsCheckStatement_Assign (sStatement *s);
void symbolsCheckStatement_IfElse (sStatement *s);
void symbolsCheckStatement_If (sStatement *s);
void symbolsCheckStatement_While (sStatement *s);
void symbolsCheckStatement_Compound (sStatement *s);
void symbolsCheckStatement_ForTo (sStatement *s);
/*-------------------------------------------------------------*/
void symbolsCheckVariable (sVariable *s);
void symbolsCheckVariable_Id (sVariable *s);
void symbolsCheckVariable_Indexed (sVariable *s);
void symbolsCheckVariable_Struct (sVariable *s);
/*-------------------------------------------------------------*/
void symbolsCheckExpression (sExpression *s);
void symbolsCheckExpression_Dyadisk (sExpression *s);
void symbolsCheckExpression_Term (sExpression *s);
/*-------------------------------------------------------------*/
void symbolsCheckTerm (sTerm *s);
void symbolsCheckTerm_Variable (sTerm *s);
void symbolsCheckTerm_Call (sTerm *s);
void symbolsCheckTerm_Parantes (sTerm *s);
void symbolsCheckTerm_Not (sTerm *s);
void symbolsCheckTerm_Numeric (sTerm *s);
void symbolsCheckTerm_Num (sTerm *s);
void symbolsCheckTerm_True (sTerm *s);
void symbolsCheckTerm_False (sTerm *s);
void symbolsCheckTerm_Null (sTerm *s);
/*-------------------------------------------------------------*/
void symbolsCheckAct_list (sAct_list *s, sHead *head);
/*-------------------------------------------------------------*/
```

```
void symbolsCheckExp_list (sExp_list *exp_list, sVar_decl_list *var_list);
void symbolsCheckExp_list_List (sExp_list *exp_list,
                                sVar_decl_list *var_list);
void symbolsCheckExp_list_Expression (sExp_list *exp_list,
                                      sVar_decl_list *var_list);
bool symbolsCheckExp_Var_typer (sExpression *exp, sVar_type *var);


sType *checkReturnType;   /* sættes ved ny head for chect at return */


void symbolsCheckProgram (sProgram *s)
{
  fprintf(msgFile, "TONY type-check start\r\n");
  checkReturnType = makeType_Int();     /* tillad return int i main !! */
  checkSymbolTable = s->symbolTable;      /* opsæt adresse til symboltabel */
  symbolsCheckBody(s->val.eProgram.body);
  fprintf(msgFile, "TONY type-check slut\r\n");
}
void symbolsCheckFunction (sFunction *s)
{
  /* gem adresse på aktuel symboltabel */
  SymbolTable *saveSymbolTable = checkSymbolTable;
  /* opsæt adresse til symboltabel */
  checkSymbolTable = s->symbolTable;
  symbolsCheckHead(s->val.eFunction.head);
    symbolsCheckBody(s->val.eFunction.body);
  /* reetabler adresse på aktuel symboltabel */
  checkSymbolTable = saveSymbolTable;
}
void symbolsCheckHead (sHead *s)
{
  checkReturnType = s->val.eHead.type;
  symbolsCheckPar_decl_list(s->val.eHead.par_decl_list);
    symbolsCheckType(s->val.eHead.type);
}
/*-----------------------------------------------------------------*/
void symbolsCheckType (sType *s)
{
    switch(s->kind)
  {
    case kType_Id:    symbolsCheckType_Id   (s);break;
    case kType_Int:   symbolsCheckType_Int  (s);break;
    case kType_Bool:  symbolsCheckType_Bool (s);break;
    case kType_Array: symbolsCheckType_Array  (s);break;
    case kType_Record:  symbolsCheckType_Record (s);break;
  }
}

void symbolsCheckType_Id (sType *s)
{
  bool walvorligFejl = false;
  sDeclaration *decl;
  SYMBOL *symbol = getSymbol(checkSymbolTable, s->val.eId.id, s->linienr);
  s->val.eId.type = NULL;
  if (symbol == NULL) {
    fprintf(msgFile, "%d: Fejl: type %s er ikke defineret\n",
        s->val.eId.id_linienr,s->val.eId.id);
    ++alvorligFejl;
    walvorligFejl = true;
  }
  else if (symbol->kind != symType) {
    if (symbol->kind == symVar) {
```

```
      fprintf(msgFile,
        "%d: Fejl: <%s> er en variabel og kan ikke bruges som type\n",
        s->val.eId.id_linienr,s->val.eId.id);
      ++alvorligFejl;
      walvorligFejl = true;
    }
    else if (symbol->kind == symHead) {
      fprintf(msgFile,
        "%d: Fejl: <%s> er en funktion og kan ikke bruges som type\n",
        s->val.eId.id_linienr,s->val.eId.id);
      ++alvorligFejl;
      walvorligFejl = true;
    }
    else {
      fprintf(msgFile,
        "%d: Fejl: <%s> ikke en type!!!!!\n",
        s->val.eId.id_linienr,s->val.eId.id);
      ++alvorligFejl;
      walvorligFejl = true;
    }
  }
  else if (symbolCheckDeclare_bloknr < symbol->symbolBuildDeclare_bloknr) {
    decl = symbol->astNode;
    if (restriktivCheck)
    {
      fprintf(msgFile,
        "%d: Fejl: typen <%s> er ikke tilgaengelig - erklaeret senere<%d>\n",
        s->val.eId.id_linienr,s->val.eId.id, decl->linienr);
      ++alvorligFejl;
      walvorligFejl = true;
    }
    else {
/*
      if (symbolCheckDeclare_bloknr == symbol->symbolBuildDeclare_bloknr)
        fprintf(msgFile,
  "%d: Advarsel: typen <%s> er erklaeret i samme sætning som den bruges<%d>\n",
          s->val.eId.id_linienr,s->val.eId.id, decl->linienr);
      else
*/
        fprintf(msgFile,
"%d: Advarsel: reference til typen <%s> som foerst erklaeres senere<%d>\n",
          s->val.eId.id_linienr,s->val.eId.id, decl->linienr);
      ++advarsel;
    }
  }
  if (walvorligFejl==false)
  {
    decl = symbol->astNode;
    s->val.eId.type = decl->val.eType.type;/*registrer ref. til sType i AST*/
    s->slutType = s->val.eId.type->slutType;
    if (s->slutType == NULL) {    /* fanger cykliske */
      fprintf(msgFile,
        "%d: Fejl: type <%s> reference er ikke gyldig\n",
        s->val.eId.id_linienr,s->val.eId.id);
      ++alvorligFejl;
    }
  }
}

void symbolsCheckType_Int (sType *s)
{
}
void symbolsCheckType_Bool (sType *s)
```

```
{
}
void symbolsCheckType_Array (sType *s)
{
  symbolsCheckType(s->val.eArray.type);
}
void symbolsCheckType_Record (sType *s)
{
  symbolsCheckVar_decl_list(s->val.eRecord.var_decl_list);
}
/*-------------------------------------------------------------*/
void symbolsCheckPar_decl_list (sPar_decl_list *s)
{
  ++symbolCheckDeclare_bloknr;
  switch(s->kind)
  {
    case kPar_decl_list_List:  symbolsCheckPar_decl_list_List  (s);break;
    case kPar_decl_list_Empty: symbolsCheckPar_decl_list_Empty (s);break;
  }
}
void symbolsCheckPar_decl_list_List (sPar_decl_list *s)
{
  symbolsCheckVar_decl_list(s->val.eList.var_decl_list);
}
void symbolsCheckPar_decl_list_Empty (sPar_decl_list *s)
{
}
/*-------------------------------------------------------------*/
void symbolsCheckVar_decl_list (sVar_decl_list *s)
{
  switch(s->kind)
  {
    case kVar_decl_list_List: symbolsCheckVar_decl_list_List(s);break;
    case kVar_decl_list_Var:  symbolsCheckVar_decl_list_Var (s);break;
  }
}
void symbolsCheckVar_decl_list_List (sVar_decl_list *s)
{
  symbolsCheckVar_decl_list(s->val.eList.var_decl_list);
  symbolsCheckVar_type(s->val.eList.var_type);
}
void symbolsCheckVar_decl_list_Var (sVar_decl_list *s)
{
  symbolsCheckVar_type(s->val.eVar.var_type);
}
/*-------------------------------------------------------------*/
void symbolsCheckVar_type (sVar_type *s)
{
  symbolsCheckType(s->val.eVar.type);
}
/*-------------------------------------------------------------*/
void symbolsCheckBody (sBody *s)
{
  symbolsCheckDecl_list(s->val.eBody.decl_list);
  symbolsCheckStatement_list(s->val.eBody.statement_list);
}
/*-------------------------------------------------------------*/
void symbolsCheckDecl_list (sDecl_list *s)
{
  switch(s->kind)
  {
    case kDecl_list_List:  symbolsCheckDecl_list_List  (s);break;
    case kDecl_list_Empty: symbolsCheckDecl_list_Empty (s);break;
```

```
  }
}
void symbolsCheckDecl_list_List (sDecl_list *s)
{
  symbolsCheckDecl_list(s->val.eList.decl_list);
  symbolsCheckDeclaration(s->val.eList.declaration);
}
void symbolsCheckDecl_list_Empty (sDecl_list *s)
{
}
/*------------------------------------------------------------------*/
void symbolsCheckDeclaration (sDeclaration *s)
{
  ++symbolCheckDeclare_bloknr;
    switch(s->kind)
  {
    case kDeclaration_Type:     symbolsCheckDeclaration_Type    (s);break;
    case kDeclaration_Function: symbolsCheckDeclaration_Function(s);break;
    case kDeclaration_Var:      symbolsCheckDeclaration_Var     (s);break;
  }
}
void symbolsCheckDeclaration_Type (sDeclaration *s)
{
  if (!s->val.eType.aktiv) return;  /* deaktiveret i weed-fase */
  symbolsCheckType(s->val.eType.type);
}
void symbolsCheckDeclaration_Function (sDeclaration *s)
{
  symbolsCheckFunction(s->val.eFunction.function);
}
void symbolsCheckDeclaration_Var (sDeclaration *s)
{
  symbolsCheckVar_decl_list(s->val.eVar.var_decl_list);
}
/*------------------------------------------------------------------*/




/*------------------------------------------------------------------*/
void symbolsCheckStatement_list (sStatement_list *s)
{
    switch(s->kind)
  {
    case kStatement_list_List:      symbolsCheckStatement_list_List     (s);
                                    break;
    case kStatement_list_Statement: symbolsCheckStatement_list_Statement(s);
                                    break;
  }
}
void symbolsCheckStatement_list_List (sStatement_list *s)
{
  symbolsCheckStatement_list(s->val.eList.statement_list);
  symbolsCheckStatement(s->val.eList.statement);
}
void symbolsCheckStatement_list_Statement (sStatement_list *s)
{
  symbolsCheckStatement(s->val.eList.statement);
}
/*------------------------------------------------------------------*/
void symbolsCheckStatement (sStatement *s)
{
  switch(s->kind)
```

```
  {
    case kStatement_Return:   symbolsCheckStatement_Return  (s);break;
    case kStatement_Write:    symbolsCheckStatement_Write   (s);break;
    case kStatement_NewLength:symbolsCheckStatement_NewLength(s);break;
    case kStatement_New:      symbolsCheckStatement_New     (s);break;
    case kStatement_Assign:   symbolsCheckStatement_Assign  (s);break;
    case kStatement_IfElse:   symbolsCheckStatement_IfElse  (s);break;
    case kStatement_If:       symbolsCheckStatement_If      (s);break;
    case kStatement_While:    symbolsCheckStatement_While   (s);break;
    case kStatement_Compound: symbolsCheckStatement_Compound(s);break;
    case kStatement_ForTo:    symbolsCheckStatement_ForTo   (s);break;
    case kStatement_ForDownto:symbolsCheckStatement_ForTo   (s);break;
  }
}
void symbolsCheckStatement_Return (sStatement *s)
{
  symbolsCheckExpression(s->val.eReturn.expression);
  if (getSlutType(s->val.eReturn.expression->type)
      != getSlutType(checkReturnType)) {
    fprintf(msgFile,
"%d: Fejl: returtype <%s -> %s> svarer ikke til funktionshoved<%s -> %s>\n",
    s->linienr,
    type_Kind_Txt(s->val.eReturn.expression->type),
    type_Kind_Txt(getSlutType(s->val.eReturn.expression->type)),
    type_Kind_Txt(checkReturnType),
    type_Kind_Txt(getSlutType(checkReturnType)));
    ++alvorligFejl;
  }
}
void symbolsCheckStatement_Write (sStatement *s)
{
  sType *slutType;
  symbolsCheckExpression(s->val.eWrite.expression);
  slutType = getSlutType(s->val.eWrite.expression->type);
  if (slutType == NULL) {
    fprintf(msgFile,
      "%d: Fejl: expression i write er ikke en identificerbar type\n",
      s->linienr);
    ++alvorligFejl;
  }
  else if (restriktivCheck && slutType->kind != kType_Int) {
    fprintf(msgFile,
"%d: Fejl: expression i write er af typen<%s> men kun <int> er accepteret\n",
      s->linienr,
      type_Kind_Txt(slutType));
    ++alvorligFejl;
  }
}
void symbolsCheckStatement_NewLength (sStatement *s)
{
  sType *slutType;
  symbolsCheckVariable(s->val.eNewLength.variable);

  slutType = getSlutType(s->val.eNewLength.variable->type);
  if (slutType == NULL) {
    fprintf(msgFile,
"%d: Fejl: varabel i new %s length of .. har ikke en identificerbar type\n",
      s->linienr, s->val.eNewLength.variable->val.eId.id);
    ++alvorligFejl;
  }
  else if (slutType->kind != kType_Array) {
    fprintf(msgFile,
"%d: Fejl: variabel i new %s length of .. har af typen <%s> men \
```

```
kun <array> er accepteret\n",
      s->linienr, s->val.eNewLength.variable->val.eId.id, ty-
pe_Kind_Txt(slutType));
    ++alvorligFejl;
  }
  symbolsCheckExpression(s->val.eNewLength.expression);
  slutType = getSlutType(s->val.eNewLength.expression->type);
  if (slutType == NULL) {
    fprintf(msgFile, "%d: Fejl: expression i new %s length of .. har \
ikke en identificerbar type\n",s->linienr,
      s->val.eNewLength.variable->val.eId.id);
    ++alvorligFejl;
  }
  else if (slutType->kind != kType_Int) {
    fprintf(msgFile, "%d: Fejl: expression i new %s length of .. er \
af typen <%s> men kun <int> er accepteret\n",s->linienr,
      s->val.eNewLength.variable->val.eId.id, type_Kind_Txt(slutType));
    ++alvorligFejl;
  }
}
void symbolsCheckStatement_New (sStatement *s)
{
  sType *slutType;
  symbolsCheckVariable(s->val.eNew.variable);

  slutType = getSlutType(s->val.eNew.variable->type);
  if (slutType == NULL) {
    fprintf(msgFile,
      "%d: Fejl: varabel i new %s har ikke en identificerbar type\n",
      s->linienr, s->val.eNew.variable->val.eId.id);
    ++alvorligFejl;
  }
  else if (slutType->kind != kType_Record) {
    fprintf(msgFile,
"%d: Fejl: variabel i new %s af typen <%s> men kun <record> er accepteret\n",
      s->linienr, s->val.eNew.variable->val.eId.id, type_Kind_Txt(slutType));
    ++alvorligFejl;
  }
  else if (recordArray) {
    if (s->val.eNew.variable->kind == kVariable_Indexed) {
      fprintf(msgFile,
      "%d: Fejl: new er ikke tilladt paa array af record\n",
      s->linienr);
      ++alvorligFejl;
    }
  }
}
void symbolsCheckStatement_Assign (sStatement *s)
{
  bool walvorligFejl = false;
  sType *slutType;
  symbolsCheckVariable(s->val.eAssign.variable);
  symbolsCheckExpression(s->val.eAssign.expression);

  if (s->val.eAssign.variable->type == NULL) {
    fprintf(msgFile,
      "%d: Fejl: assign ikke mulig da variabelens type ikke er def\n",
      s->linienr);
    ++alvorligFejl;
    walvorligFejl = true;
  }
  else if (s->val.eAssign.variable->type->slutType == NULL) {
    fprintf(msgFile,
```

```
      "%d: Fejl: assign ikke mulig da variabelens sluttype ikke er def\n",
      s->linienr);
    ++alvorligFejl;
    walvorligFejl = true;
  }
  if (s->val.eAssign.expression->type == NULL) {
    fprintf(msgFile,
      "%d: Fejl: assign ikke mulig da expression type ikke def\n",
      s->linienr);
    ++alvorligFejl;
    walvorligFejl = true;
  }
  else if (s->val.eAssign.expression->type->slutType == NULL) {
    fprintf(msgFile,
      "%d: Fejl: assign ikke mulig da expression sluttype ikke def\n",
      s->linienr);
    ++alvorligFejl;
    walvorligFejl = true;
  }
  if (!walvorligFejl)
    if (!equalTypes(s->val.eAssign.variable->type,
          s->val.eAssign.expression->type)) {
    fprintf(msgFile,
    "%d: Fejl: type i assignment <%s> = <%s> : der tilades kun ens typer\n",
      s->linienr,
      type_Kind_Txt(s->val.eAssign.variable->type),
      type_Kind_Txt(s->val.eAssign.expression->type));
    ++alvorligFejl;
  }
}
void symbolsCheckStatement_IfElse (sStatement *s)
{
  sType *slutType;
  symbolsCheckExpression(s->val.eIfElse.expression);
  slutType = getSlutType(s->val.eIfElse.expression->type);
  if (slutType == NULL) {
    fprintf(msgFile,
      "%d: Fejl: expression i if-else er ikke en identificerbar type\n",
      s->val.eIfElse.expression->linienr);
    ++alvorligFejl;
  }
  else if (slutType->kind != kType_Bool) {
    fprintf(msgFile,
      "%d: Fejl: expression i if-else er af typen <%s> men \
kun <bool> er accepteret\n",
      s->val.eIfElse.expression->linienr, type_Kind_Txt(slutType));
    ++alvorligFejl;
  }

  symbolsCheckStatement(s->val.eIfElse.ifStatement);

  symbolsCheckStatement(s->val.eIfElse.elseStatement);
}
void symbolsCheckStatement_If (sStatement *s)
{
  sType *slutType;
  symbolsCheckExpression(s->val.eIf.expression);

  slutType = getSlutType(s->val.eIf.expression->type);
  if (slutType == NULL) {
    fprintf(msgFile,
      "%d: Fejl: expression i if er ikke en identificerbar type\n",
      s->val.eIf.expression->linienr);
```

```
    ++alvorligFejl;
  }
  else if (slutType->kind != kType_Bool) {
    fprintf(msgFile,
"%d: Fejl: expression i if er af typen <%s> men kun <bool> er accepteret\n",
      s->val.eIf.expression->linienr, type_Kind_Txt(slutType));
    ++alvorligFejl;
  }

  symbolsCheckStatement(s->val.eIf.ifStatement);
}
void symbolsCheckStatement_While (sStatement *s)
{
  sType *slutType;
  symbolsCheckExpression(s->val.eWhile.expression);

  slutType = getSlutType(s->val.eWhile.expression->type);
  if (slutType == NULL) {
    fprintf(msgFile,
      "%d: Fejl: expression i while er ikke en identificerbar type\n",
      s->linienr);
    ++alvorligFejl;
  }
  else if (slutType->kind != kType_Bool) {
    fprintf(msgFile,
      "%d: Fejl: expression i while er af typen <%s> men kun <bool> \
er accepteret\n",s->linienr, type_Kind_Txt(slutType));
    ++alvorligFejl;
  }

  symbolsCheckStatement(s->val.eWhile.statement);
}
void symbolsCheckStatement_Compound (sStatement *s)
{
  symbolsCheckStatement_list(s->val.eCompound.statement_list);
}
void symbolsCheckStatement_ForTo (sStatement *s)
{
  bool walvorligFejl = false;
  sType *slutType;
  symbolsCheckVariable(s->val.eForTo.variable);
  symbolsCheckExpression(s->val.eForTo.expressionInit);

  if (s->val.eForTo.variable->type == NULL) {
    fprintf(msgFile,
      "%d: Fejl: variabel i for-to skal vaere at typen int\n",s->linienr);
    ++alvorligFejl;
    walvorligFejl = true;
  }
  else if (s->val.eForTo.variable->type->kind != kType_Int) {
    fprintf(msgFile,
"%d: Fejl: variabel i for-to er af typen <%s> men kun <int> er accepteret\n",
      s->linienr, type_Kind_Txt(s->val.eForTo.variable->type));
    ++alvorligFejl;
    walvorligFejl = true;
  }
  if (s->val.eForTo.expressionInit->type == NULL) {
    fprintf(msgFile,
      "%d: Fejl: assign ikke mulig da expression type ikke def\n",
      s->linienr);
    ++alvorligFejl;
    walvorligFejl = true;
  }
```

```
  else if (s->val.eForTo.expressionInit->type->kind != kType_Int) {
    fprintf(msgFile,
      "%d: Fejl: assign-expression i for-to er af typen <%s> men \
kun <int> er accepteret\n",s->linienr,
      type_Kind_Txt(s->val.eForTo.expressionInit->type));
    ++alvorligFejl;
    walvorligFejl = true;
  }

  symbolsCheckExpression(s->val.eForTo.expressionTo);
  slutType = getSlutType(s->val.eForTo.expressionTo->type);
  if (slutType == NULL) {
    fprintf(msgFile,
      "%d: Fejl: to expression i for-to er ikke en identificerbar type\n",
      s->linienr);
    ++alvorligFejl;
  }
  else if (slutType->kind != kType_Int) {
    fprintf(msgFile,
      "%d: Fejl: to expression i for-to er af typen <%s> men kun <int> \
er accepteret\n",s->linienr, type_Kind_Txt(slutType));
    ++alvorligFejl;
  }

  symbolsCheckStatement(s->val.eForTo.statement);
}
/*-------------------------------------------------------------*/
void symbolsCheckVariable (sVariable *s)
{
  switch(s->kind)
  {
    case kVariable_Id:      symbolsCheckVariable_Id     (s);break;
    case kVariable_Indexed: symbolsCheckVariable_Indexed(s);break;
    case kVariable_Struct:  symbolsCheckVariable_Struct (s);break;
  }
}
void symbolsCheckVariable_Id (sVariable *s)
{
  bool walvorligFejl = false;
  sVar_type *var_type;
  SYMBOL *symbol;
  symbol = getSymbol(checkSymbolTable, s->val.eId.id, s->linienr);

  s->type = NULL;
  if (symbol == NULL) {
    fprintf(msgFile, "%d: Fejl: variabel <%s> er ikke defineret\n",
      s->val.eId.id_linienr,s->val.eId.id);
    ++alvorligFejl;
    walvorligFejl = true;
  }
  else if (symbol->kind != symVar) {
    if (symbol->kind == symType) {
      fprintf(msgFile,
        "%d: Fejl: <%s> en type og kan ikke bruges som variabel\n",
        s->val.eId.id_linienr,s->val.eId.id);
      ++alvorligFejl;
      walvorligFejl = true;
    }
    else if (symbol->kind == symHead) {
      fprintf(msgFile,
        "%d: Fejl: <%s> en funktion og kan ikke bruges som type\n",
        s->val.eId.id_linienr,s->val.eId.id);
      ++alvorligFejl;
```

```
      walvorligFejl = true;
    }
    else {
      fprintf(msgFile, "%d: Fejl: <%s> ikke en variabel!!!!!\n",
        s->val.eId.id_linienr,s->val.eId.id);
      ++alvorligFejl;
      walvorligFejl = true;
    }
  }
  else if (symbolCheckDeclare_bloknr < symbol->symbolBuildDeclare_bloknr) {
    var_type = symbol->astNode;
    if (restriktivCheck) {
      fprintf(msgFile,
"%d: Fejl: variablen <%s> er ikke tilgaengelig - erklaret senere<%d>\n",
        s->val.eId.id_linienr,s->val.eId.id,
        var_type->linienr);
      ++alvorligFejl;
      walvorligFejl = true;
    }
    else {
      fprintf(msgFile,
        "%d: advarsel: variablen <%s> er erklaret senere end brug<%d>\n",
        s->val.eId.id_linienr,s->val.eId.id,
        var_type->linienr);
      ++advarsel;
    }
  }
  if (!walvorligFejl) {
    /* registrer ref. til sVar_Type i AST */
    s->val.eId.var_type = symbol->astNode;
    s->type = s->val.eId.var_type->val.eVar.type;
    if (s->type->slutType == NULL) {
      fprintf(msgFile,
        "%d: Fejl: variabel <%s> kan ikke foeres tilbage til gyldig type\n",
        s->val.eId.id_linienr,s->val.eId.id);
      ++alvorligFejl;
    }
  }
}

void symbolsCheckVariable_Indexed (sVariable *s)
{
  symbolsCheckVariable(s->val.eIndexed.variable);
  s->type = NULL;
  if (s->val.eIndexed.variable->type == NULL) {
    fprintf(msgFile, "%d: Fejl: indexering ikke mulig her\n",s->linienr);
    ++alvorligFejl;
  }
  else if (s->val.eIndexed.variable->type->slutType == NULL) {
    fprintf(msgFile, "%d: Fejl: indexering ikke mulig her\n",s->linienr);
    ++alvorligFejl;
  }
  else if (s->val.eIndexed.variable->type->slutType->kind != kType_Array) {
    fprintf(msgFile, "%d: Fejl: indexering ikke mulig paa typen <%s>\n",
      s->linienr, type_Kind_Txt(s->val.eIndexed.variable->type->slutType));
    ++alvorligFejl;
  }
  else {
    s->type = s->val.eIndexed.variable->type->slutType->val.eArray.type;
  }

  symbolsCheckExpression(s->val.eIndexed.expression);
```

```
  if (s->val.eIndexed.expression->type == NULL) {
    fprintf(msgFile,
      "%d: Fejl: indexering ikke mulig - indextype mangler\n", s->linienr);
    ++alvorligFejl;
  }
  else if (s->val.eIndexed.expression->type->kind != kType_Int) {
    fprintf(msgFile,
"%d: Fejl: indexering ikke mulig med typen <%s> men kun med typen <int>\n",
      s->linienr, type_Kind_Txt(s->val.eIndexed.expression->type));
    ++alvorligFejl;
  }
}
void symbolsCheckVariable_Struct (sVariable *s)
{
  bool walvorligFejl = false;
  SYMBOL *symbol;

  symbolsCheckVariable(s->val.eStruct.variable);

  s->type = NULL;
  if (s->val.eStruct.variable->type == NULL) {
    fprintf(msgFile,
      "%d: Fejl: struktur-niveau ikke mulig her\n",s->linienr);
    ++alvorligFejl;
    walvorligFejl = true;
  }
  else if (s->val.eStruct.variable->type->slutType == NULL) {
    fprintf(msgFile,
      "%d: Fejl: struktur-niveau ikke mulig her\n",s->linienr);
    ++alvorligFejl;
    walvorligFejl = true;
  }
  else if (s->val.eStruct.variable->type->slutType->kind != kType_Record) {
    fprintf(msgFile,
      "%d: Fejl: strutur-niveau ikke mulig paa typen <%s> men kun \
på typen <record>\n",
      s->linienr, type_Kind_Txt(s->val.eStruct.variable->type->slutType));
    ++alvorligFejl;
    walvorligFejl = true;
  }
  else {
    symbol = getSymbol(
        s->val.eStruct.variable->type->slutType->val.eRecord.symbolTable,
        s->val.eStruct.id, s->linienr);
    if (symbol == NULL) {
      fprintf(msgFile,
        "%d: Fejl: variabel <%s> er ikke defineret\n",
        s->val.eStruct.id_linienr,s->val.eStruct.id);
      ++alvorligFejl;
      walvorligFejl = true;
    }
    else if (symbol->kind != symVar) {
      if (symbol->kind == symType) {
        fprintf(msgFile,
          "%d: Fejl: <%s> en type og kan ikke bruges som variabel\n",
          s->val.eStruct.id_linienr,s->val.eStruct.id);
        ++alvorligFejl;
        walvorligFejl = true;
      }
      else if (symbol->kind == symHead) {
        fprintf(msgFile,
          "%d: Fejl: <%s> en funktion og kan ikke bruges som type\n",
          s->val.eStruct.id_linienr,s->val.eStruct.id);
```

```
        ++alvorligFejl;
        walvorligFejl = true;
      }
      else {
        fprintf(msgFile,
          "%d: Fejl: <%s> ikke en variabel!!!!!\n",
          s->val.eStruct.id_linienr,s->val.eStruct.id);
        ++alvorligFejl;
        walvorligFejl = true;
      }
    }
    else if (symbolCheckDeclare_bloknr < symbol->symbolBuildDeclare_bloknr){
      if (restriktivCheck) {
        fprintf(msgFile,
        "%d: Fejl: variablen <%s> er ikke tilgaengelig - erklaret senere\n",
          s->val.eStruct.id_linienr,s->val.eStruct.id);
        ++alvorligFejl;
        walvorligFejl = true;
      }
      else {
        fprintf(msgFile,
          "%d: Fejl: variablen <%s> er erklaret senere en brug\n",
          s->val.eStruct.id_linienr,s->val.eStruct.id);
        ++advarsel;
      }
    }
  }
  if (!walvorligFejl) {
    /* registrer ref. til sVar_Type i AST */
    s->val.eId.var_type = symbol->astNode;
    s->type = s->val.eId.var_type->val.eVar.type;
    if (s->type->slutType == NULL) {
      fprintf(msgFile,
        "%d: Fejl: type <%s> kan ikke foeres tilbage til gyldig type\n",
        s->val.eStruct.id_linienr,s->val.eStruct.id);
      ++alvorligFejl;
      walvorligFejl = true;
    }
  }
}
/*-----------------------------------------------------------------*/
void symbolsCheckExpression (sExpression *s)
{
  switch(s->kind)
  {
    case kExp_Eq:   symbolsCheckExpression_Dyadisk  (s);break;
    case kExp_Neq:  symbolsCheckExpression_Dyadisk  (s);break;
    case kExp_Lt:   symbolsCheckExpression_Dyadisk  (s);break;
    case kExp_Gt:   symbolsCheckExpression_Dyadisk  (s);break;
    case kExp_Lteq: symbolsCheckExpression_Dyadisk  (s);break;
    case kExp_Gteq: symbolsCheckExpression_Dyadisk  (s);break;
    case kExp_Add:  symbolsCheckExpression_Dyadisk  (s);break;
    case kExp_Sub:  symbolsCheckExpression_Dyadisk  (s);break;
    case kExp_Mul:  symbolsCheckExpression_Dyadisk  (s);break;
    case kExp_Div:  symbolsCheckExpression_Dyadisk  (s);break;
    case kExp_Or:   symbolsCheckExpression_Dyadisk  (s);break;
    case kExp_And:  symbolsCheckExpression_Dyadisk  (s);break;
    case kExp_Term: symbolsCheckExpression_Term     (s);break;
  }
}
void symbolsCheckExpression_Dyadisk (sExpression *s)
{
  symbolsCheckExpression(s->val.eLeftRight.expressionLeft);
```

```
  symbolsCheckExpression(s->val.eLeftRight.expressionRight);
  if (s->val.eLeftRight.expressionLeft->type == NULL ||
      s->val.eLeftRight.expressionRight->type == NULL){
    if (s->val.eLeftRight.expressionLeft->type == NULL) {
      fprintf(msgFile,
        "%d: Fejl: venste del af expression mangler type\n",s->linienr);
      ++alvorligFejl;
    }
    if (s->val.eLeftRight.expressionRight->type == NULL){
      fprintf(msgFile, "%d: Fejl: hoejre del af expression\n",s->linienr);
      ++alvorligFejl;
    }
    return;
  }
  switch(s->kind)
  {
    case kExp_Eq:
    case kExp_Neq:  s->type = makeType_Bool();
      if (! equalTypes(s->val.eLeftRight.expressionLeft->type,
            s->val.eLeftRight.expressionRight->type)) {
        fprintf(msgFile,
"%d: Fejl: type i expression <%s> %s <%s> der tilades kun ens typer\n",
          s->linienr, type_Kind_Txt(s->val.eLeftRight.expressionLeft->type),
          expression_Kind_Txt(s),
          type_Kind_Txt(s->val.eLeftRight.expressionRight->type));
        ++alvorligFejl;
      }
      break;
    case kExp_Lt:
    case kExp_Gt:
    case kExp_Lteq:
    case kExp_Gteq: s->type = makeType_Bool();
      if (s->val.eLeftRight.expressionLeft->type->kind != kType_Int ||
        s->val.eLeftRight.expressionRight->type->kind != kType_Int) {
        fprintf(msgFile,
"%d: Fejl: type i expression <%s> %s <%s> der tilades kun type int\n",
            s->linienr,
            type_Kind_Txt(s->val.eLeftRight.expressionLeft->type),
            expression_Kind_Txt(s),
            type_Kind_Txt(s->val.eLeftRight.expressionRight->type));
        ++alvorligFejl;
      }
      break;
    case kExp_Add:
    case kExp_Sub:
    case kExp_Mul:
    case kExp_Div:  s->type = makeType_Int();
      if (s->val.eLeftRight.expressionLeft->type->kind != kType_Int ||
        s->val.eLeftRight.expressionRight->type->kind != kType_Int) {
        fprintf(msgFile,
"%d: Fejl: type i expression <%s> %s <%s> der tilades kun type int\n",
            s->linienr,
            type_Kind_Txt(s->val.eLeftRight.expressionLeft->type),
            expression_Kind_Txt(s),
            type_Kind_Txt(s->val.eLeftRight.expressionRight->type));
        ++alvorligFejl;
      }
      break;
    case kExp_Or:
    case kExp_And:  s->type = makeType_Bool();
      if (s->val.eLeftRight.expressionLeft->type->kind != kType_Bool ||
        s->val.eLeftRight.expressionRight->type->kind != kType_Bool) {
        fprintf(msgFile,
```

```
"%d: Fejl: type i expression <%s> %s <%s> der tilades kun type bool\n",
            s->linienr,
            type_Kind_Txt(s->val.eLeftRight.expressionLeft->type),
            expression_Kind_Txt(s),
            type_Kind_Txt(s->val.eLeftRight.expressionRight->type));
        ++alvorligFejl;
      }
      break;
  }

}
void symbolsCheckExpression_Term (sExpression *s)
{
  symbolsCheckTerm(s->val.eTerm.term);
  s->type = s->val.eTerm.term->type;
}
/*-------------------------------------------------------------------*/
void symbolsCheckTerm (sTerm *s)
{
    switch(s->kind)
  {
    case kTerm_Variable:  symbolsCheckTerm_Variable(s);break;
    case kTerm_Call:      symbolsCheckTerm_Call    (s);break;
    case kTerm_Parantes:  symbolsCheckTerm_Parantes(s);break;
    case kTerm_Not:       symbolsCheckTerm_Not     (s);break;
    case kTerm_Numeric:   symbolsCheckTerm_Numeric (s);break;
    case kTerm_Num:       symbolsCheckTerm_Num     (s);break;
    case kTerm_True:      symbolsCheckTerm_True    (s);break;
    case kTerm_False:     symbolsCheckTerm_False   (s);break;
    case kTerm_Null:      symbolsCheckTerm_Null    (s);break;
  }
}
void symbolsCheckTerm_Variable (sTerm *s)
{
  /* check variabel */
  symbolsCheckVariable(s->val.eVariable.variable);
  s->type = s->val.eVariable.variable->type;
}
void symbolsCheckTerm_Call (sTerm *s)
{
  SYMBOL *symbol = getSymbol(checkSymbolTable, s->val.eCall.id, s->linienr);
  s->type = NULL;
  if (symbol == NULL) {
    fprintf(msgFile,
      "%d: Fejl: funktion %s er ikke defineret/tilgaengelig\n",
      s->val.eCall.id_linienr,s->val.eCall.id);
    ++alvorligFejl;
  }
  else if (symbol->kind != symHead) {
    if (symbol->kind == symType) {
      fprintf(msgFile,
        "%d: Fejl: %s en type og kan ikke bruges som funktion\n",
        s->val.eCall.id_linienr,s->val.eCall.id);
      ++alvorligFejl;
    }
    else if (symbol->kind == symVar) {
      fprintf(msgFile,
        "%d: Fejl: %s en variabel og kan ikke bruges som funktion\n",
        s->val.eCall.id_linienr,s->val.eCall.id);
      ++alvorligFejl;
    }
    else {
      fprintf(msgFile,
```

```
        "%d: Fejl: %s ikke en funktion!!!!!\n",
        s->val.eCall.id_linienr,s->val.eCall.id);
      ++alvorligFejl;
    }
  }
  else {
    s->val.eCall.head = symbol->astNode;/* registrer ref. til sHead i AST */
    s->type = s->val.eCall.head->val.eHead.type;
    if (s->type->slutType == NULL) {
      fprintf(msgFile,
        "%d: Fejl: funktion %s kan ikke foeres tilbage til gyldig type\n",
        s->val.eCall.id_linienr,s->val.eCall.id);
      ++alvorligFejl;
    }
    /* check forudsætter funktion fundet */
    symbolsCheckAct_list(s->val.eCall.act_list, s->val.eCall.head );
  }

}
void symbolsCheckTerm_Parantes (sTerm *s)
{
  symbolsCheckExpression(s->val.eParantes.expression);
  s->type = s->val.eParantes.expression->type;
}
void symbolsCheckTerm_Not (sTerm *s)
{
  s->type = makeType_Bool();
  symbolsCheckTerm(s->val.eNot.term);
  if (s->val.eNot.term->type == NULL) {
    fprintf(msgFile,
      "%d: Fejl: negation ikke mulig - type på udtryk mangler\n",s->linienr);
    ++alvorligFejl;
  }
  else if (s->val.eNot.term->type->kind != kType_Bool) {
    fprintf(msgFile,
"%d: Fejl: negation ikke mulig med typen <%s> men kun med typen <bool>\n",
        s->linienr, type_Kind_Txt(s->val.eNot.term->type));
    ++alvorligFejl;
  }
}
void symbolsCheckTerm_Numeric (sTerm *s)
{
  s->type = makeType_Int();
  symbolsCheckExpression(s->val.eNumeric.expression);
  if (s->val.eNumeric.expression->type == NULL) {
    fprintf(msgFile,
      "%d: Fejl: numeric operator ikke mulig - type på udtryk mangler\n",
      s->linienr);
    ++alvorligFejl;
    return;
  }
  if (s->val.eNumeric.expression->type->kind == kType_Int)
    return;   /* ok */
  if (!restriktivCheck && s->val.eNumeric.expression->type->slutType != NULL)
    if (s->val.eNumeric.expression->type->slutType->kind == kType_Int)
      return;   /* ok */
  if (s->val.eNumeric.expression->type->slutType->kind != kType_Array) {
    fprintf(msgFile,
      "%d: Fejl: numeric operator ikke mulig med typen <%s> men kun \
med typen <array og int>\n",
        s->linienr,
        type_Kind_Txt(s->val.eNumeric.expression->type->slutType));
    ++alvorligFejl;
```

```
  }
}
void symbolsCheckTerm_Num (sTerm *s)
{
  s->type = makeType_Int();
}
void symbolsCheckTerm_True (sTerm *s)
{
  s->type = makeType_Bool();
}
void symbolsCheckTerm_False (sTerm *s)
{
  s->type = makeType_Bool();
}
void symbolsCheckTerm_Null (sTerm *s)
{
  s->type = makeType_Null();
}
/*------------------------------------------------------------------*/
void symbolsCheckAct_list (sAct_list *s, sHead *head)
{
  sPar_decl_list *wPar_decl_list = head->val.eHead.par_decl_list;
  sAct_list *wAct_list = s;
  int antalVar = 1;
  int antalExp = 1;
  sVar_decl_list *wVar_decl_list;
  sExp_list *wExp_list;

  if (wPar_decl_list->kind == kPar_decl_list_Empty &&
      wAct_list->kind == kAct_list_Empty) {
    return;
  }

  /* check af parametre antal */
  wVar_decl_list = wPar_decl_list->val.eList.var_decl_list;
  while ( wVar_decl_list->kind == kVar_decl_list_List ) {
    ++antalVar;
    wVar_decl_list = wVar_decl_list->val.eList.var_decl_list;
  }
  wExp_list = wAct_list->val.eList.exp_list;
  while ( wExp_list->kind == kExp_list_List) {
    ++antalExp;
    wExp_list = wExp_list->val.eList.exp_list;
  }
  if ( antalVar != antalExp ) {
    fprintf(msgFile,
      "%d: Fejl: i antal argumenter(%d) i kald af funktion %s<%d> - \
forventet(%d)\n",
      s->linienr,
      antalExp, head->val.eHead.id, head->val.eHead.id_linienr, antalVar);
    ++alvorligFejl;
  }
  else {
    /* check af parametre typer  recursivt for at få rækkefølge
     på parametre-alvorligFejl forfra */
    symbolsCheckExp_list (s->val.eList.exp_list,
                          wPar_decl_list->val.eList.var_decl_list);
  }
}
/*------------------------------------------------------------------*/
void symbolsCheckExp_list (sExp_list *exp_list, sVar_decl_list *var_list)
{
  switch(exp_list->kind)
```

```
  {
    case kExp_list_List:
        symbolsCheckExp_list_List       (exp_list, var_list);break;
    case kExp_list_Expression:
        symbolsCheckExp_list_Expression (exp_list, var_list);break;
  }
}
void symbolsCheckExp_list_List (sExp_list *exp_list,
                                sVar_decl_list *var_list)
{
  symbolsCheckExp_list(exp_list->val.eList.exp_list,
                       var_list->val.eList.var_decl_list);
  symbolsCheckExpression(exp_list->val.eList.expression);
  if (symbolsCheckExp_Var_typer (exp_list->val.eList.expression,
                                 var_list->val.eList.var_type)) {
    /* gem reference til variabel */
    exp_list->val.eList.var_type = var_list->val.eList.var_type;
  }
}
void symbolsCheckExp_list_Expression (sExp_list *exp_list,
                                      sVar_decl_list *var_list)
{
  symbolsCheckExpression(exp_list->val.eExpression.expression);
  if (symbolsCheckExp_Var_typer (exp_list->val.eExpression.expression,
                                 var_list->val.eVar.var_type)) {
    /* gem reference til variabel */
    exp_list->val.eExpression.var_type = var_list->val.eVar.var_type;
  }

}
bool symbolsCheckExp_Var_typer (sExpression *exp, sVar_type *var)
{
  if (!equalTypes(exp->type, var->val.eVar.type)) {
    fprintf(msgFile,
      "%d: Fejl: argument type <%s> passer ikke med parameter type <%s>\n",
        exp->linienr,
        type_Kind_Txt(exp->type), type_Kind_Txt(var->val.eVar.type));
    ++alvorligFejl;
    return false;
  }
  return true;
}
```

Funktioner til udskrift af tony AST i tekstformat

tonyAST_Pretty_TXT.h

```
/* ----------------------------------------------------------------------
  Definition af funktioner til udskrift af
  TONY abstract syntax tree i TXT format
  Programmør: Bjørk Busch
  Historik:  2005.03.26  Udgave til rapport 3 færdig
  Historik:  2005.05.10  kosmetik i source
----------------------------------------------------------------------*/
void openprettyTxtTxt();
void prettyTxtProgram (sProgram *s);
void prettyTxtFunction (sFunction *s);
void prettyTxtHead (sHead *s);
void prettyTxtTail (sTail *s);
/*--------------------------------------------------------------------*/
void prettyTxtType (sType *s);
void prettyTxtType_Id (sType *s);
```

```
void prettyTxtType_Int (sType *s);
void prettyTxtType_Bool (sType *s);
void prettyTxtType_Array (sType *s);
void prettyTxtType_Record (sType *s);
/*-------------------------------------------------------------*/
void prettyTxtPar_decl_list (sPar_decl_list *s);
void prettyTxtPar_decl_list_List (sPar_decl_list *s);
void prettyTxtPar_decl_list_Empty (sPar_decl_list *s);
/*-------------------------------------------------------------*/
void prettyTxtVar_decl_list (sVar_decl_list *s);
void prettyTxtVar_decl_list_List (sVar_decl_list *s);
void prettyTxtVar_decl_list_Var (sVar_decl_list *s);
/*-------------------------------------------------------------*/
void prettyTxtVar_type (sVar_type *s);
/*-------------------------------------------------------------*/
void prettyTxtBody (sBody *s);
/*-------------------------------------------------------------*/
void prettyTxtDecl_list (sDecl_list *s);
void prettyTxtDecl_list_List (sDecl_list *s);
void prettyTxtDecl_list_Empty (sDecl_list *s);
/*-------------------------------------------------------------*/
void prettyTxtDeclaration (sDeclaration *s);
void prettyTxtDeclaration_Type (sDeclaration *s);
void prettyTxtDeclaration_Function (sDeclaration *s);
void prettyTxtDeclaration_Var (sDeclaration *s);
/*-------------------------------------------------------------*/
void prettyTxtStatement_list (sStatement_list *s);
void prettyTxtStatement_list_List (sStatement_list *s);
void prettyTxtStatement_list_Statement (sStatement_list *s);
/*-------------------------------------------------------------*/
void prettyTxtStatement (sStatement *s);
void prettyTxtStatement_Return (sStatement *s);
void prettyTxtStatement_Write (sStatement *s);
void prettyTxtStatement_NewLength (sStatement *s);
void prettyTxtStatement_New (sStatement *s);
void prettyTxtStatement_Assign (sStatement *s);
void prettyTxtStatement_IfElse (sStatement *s);
void prettyTxtStatement_If (sStatement *s);
void prettyTxtStatement_While (sStatement *s);
void prettyTxtStatement_Compound (sStatement *s);
void prettyTxtStatement_ForTo (sStatement *s);
/*-------------------------------------------------------------*/
void prettyTxtVariable (sVariable *s);
void prettyTxtVariable_Id (sVariable *s);
void prettyTxtVariable_Indexed (sVariable *s);
void prettyTxtVariable_Struct (sVariable *s);
/*-------------------------------------------------------------*/
void prettyTxtExpression (sExpression *s);
void prettyTxtExpression_Dyadisk (sExpression *s, char const *operation);
void prettyTxtExpression_Term (sExpression *s);
/*-------------------------------------------------------------*/
void prettyTxtTerm (sTerm *s);
void prettyTxtTerm_Variable (sTerm *s);
void prettyTxtTerm_Call (sTerm *s);
void prettyTxtTerm_Parantes (sTerm *s);
void prettyTxtTerm_Not (sTerm *s);
void prettyTxtTerm_Numeric (sTerm *s);
void prettyTxtTerm_Num (sTerm *s);
void prettyTxtTerm_True (sTerm *s);
void prettyTxtTerm_False (sTerm *s);
void prettyTxtTerm_Null (sTerm *s);
/*-------------------------------------------------------------*/
void prettyTxtAct_list (sAct_list *s);
```

```
void prettyTxtAct_list_List (sAct_list *s);
void prettyTxtAct_list_Empty (sAct_list *s);
/*-----------------------------------------------------------------*/
void prettyTxtExp_list (sExp_list *s);
void prettyTxtExp_list_List (sExp_list *s);
void prettyTxtExp_list_Expression (sExp_list *s);
```

tonyAST_Pretty_TXT.c

```
/* ----------------------------------------------------------------
  Realisering af funktioner til udskrift af
  TONY abstract syntax tree i TXT format
  Programmør: Bjørk Busch
  Historik: 2005.03.26  Udgave til rapport 3 færdig
  Historik: 2005.05.10  kosmetik i source
----------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include "bjbu_std.h"
#include "symbol.h"
#include "tonyAbstractAsm.h"
#include "tonyAST.h"
#include "tonyAST_Make.h"
#include "tonyAST_Pretty_TXT.h"

#define prettyTxt_FILENAME "tonyprettyTxt.txt"
FILE *prettyTxtFile;
extern FILE *msgFile;

int tablevel = 0;
void prettyTab ()
{
  int i = 0;
  for (;i<tablevel;++i)
    fprintf(prettyTxtFile,"    ");
}
void prettyAktStm(sStatement *s)
{
  if (s->aktiv) return;
  fprintf(prettyTxtFile,"inakt#  ");
}


void openprettyTxt()
{
  prettyTxtFile = fopen(prettyTxt_FILENAME,"w");
  if (prettyTxtFile==NULL){
    fprintf(msgFile, "alvorligFejl ved open af xmlfile <%s>\n",
    prettyTxt_FILENAME);
    exit(101);
  }
}

void prettyTxtProgram (sProgram *s)
{
  openprettyTxt();
  fprintf(msgFile,
       "TONY abstract syntax tree udskrives i TXT-format til <%s>\n",
       prettyTxt_FILENAME);
  prettyTxtBody(s->val.eProgram.body);
  fclose(prettyTxtFile);
}
void prettyTxtFunction (sFunction *s)
```

```
{
    prettyTxtHead(s->val.eFunction.head);
  ++tablevel;
  prettyTxtBody(s->val.eFunction.body);
  --tablevel;
    prettyTxtTail(s->val.eFunction.tail);
}
void prettyTxtHead (sHead *s)
{
  prettyTab(); fprintf(prettyTxtFile,"func fd#%s ( ",s->val.eHead.id);
  prettyTxtPar_decl_list(s->val.eHead.par_decl_list);
  fprintf(prettyTxtFile,"):");
  prettyTxtType(s->val.eHead.type);
  fprintf(prettyTxtFile,"\n\n");
}
void prettyTxtTail (sTail *s)
{
  fprintf(prettyTxtFile,"\n");
  prettyTab(); fprintf(prettyTxtFile,"end fd#%s",s->val.eTail.id);
}
/*-------------------------------------------------------------*/
void prettyTxtType (sType *s)
{
    switch(s->kind)
  {
    case kType_Id:    prettyTxtType_Id  (s);break;
    case kType_Int:   prettyTxtType_Int (s);break;
    case kType_Bool:  prettyTxtType_Bool  (s);break;
    case kType_Array: prettyTxtType_Array (s);break;
    case kType_Record:  prettyTxtType_Record(s);break;
  }
}

void prettyTxtType_Id (sType *s)
{
  fprintf(prettyTxtFile,"t#%s",s->val.eId.id);
}
void prettyTxtType_Int (sType *s)
{
  fprintf(prettyTxtFile,"t#int");
}
void prettyTxtType_Bool (sType *s)
{
  fprintf(prettyTxtFile,"t#bool");
}
void prettyTxtType_Array (sType *s)
{
  fprintf(prettyTxtFile,"t#array of ");
  prettyTxtType(s->val.eArray.type);
}
void prettyTxtType_Record (sType *s)
{
  fprintf(prettyTxtFile,"t#record of {\n");
  ++tablevel;
    prettyTxtVar_decl_list(s->val.eRecord.var_decl_list);
  --tablevel;
  fprintf(prettyTxtFile,"\n");
  prettyTab(); fprintf(prettyTxtFile,"}");
}
/*-------------------------------------------------------------*/
void prettyTxtPar_decl_list (sPar_decl_list *s)
{
    switch(s->kind)
```

```
  {
    case kPar_decl_list_List:  prettyTxtPar_decl_list_List (s);break;
    case kPar_decl_list_Empty: prettyTxtPar_decl_list_Empty(s);break;
  }
}
void prettyTxtPar_decl_list_List (sPar_decl_list *s)
{
  fprintf(prettyTxtFile,"\n");
  tablevel = tablevel+2;
  prettyTxtVar_decl_list(s->val.eList.var_decl_list);
  prettyTab();
  tablevel = tablevel-2;
}
void prettyTxtPar_decl_list_Empty (sPar_decl_list *s)
{
}
/*-----------------------------------------------------------------*/
void prettyTxtVar_decl_list (sVar_decl_list *s)
{
  switch(s->kind)
  {
    case kVar_decl_list_List: prettyTxtVar_decl_list_List (s);break;
    case kVar_decl_list_Var:  prettyTxtVar_decl_list_Var  (s);break;
  }
}
void prettyTxtVar_decl_list_List (sVar_decl_list *s)
{
  prettyTxtVar_decl_list(s->val.eList.var_decl_list);
  fprintf(prettyTxtFile,", \n");
  prettyTxtVar_type(s->val.eList.var_type);
}
void prettyTxtVar_decl_list_Var (sVar_decl_list *s)
{
  prettyTxtVar_type(s->val.eVar.var_type);
}
/*-----------------------------------------------------------------*/
void prettyTxtVar_type (sVar_type *s)
{
  prettyTab(); fprintf(prettyTxtFile,"vd#%s: ",s->val.eVar.id);
  prettyTxtType(s->val.eVar.type);
}
/*-----------------------------------------------------------------*/
void prettyTxtBody (sBody *s)
{
  prettyTxtDecl_list(s->val.eBody.decl_list);
  prettyTxtStatement_list(s->val.eBody.statement_list);
}
/*-----------------------------------------------------------------*/
void prettyTxtDecl_list (sDecl_list *s)
{
    switch(s->kind)
  {
    case kDecl_list_List:  prettyTxtDecl_list_List (s);break;
    case kDecl_list_Empty: prettyTxtDecl_list_Empty(s);break;
  }
}
void prettyTxtDecl_list_List (sDecl_list *s)
{
  prettyTxtDecl_list(s->val.eList.decl_list);
  prettyTxtDeclaration(s->val.eList.declaration);
  fprintf(prettyTxtFile,";\n\n");
}
void prettyTxtDecl_list_Empty (sDecl_list *s)
```

```
{
}
/*-------------------------------------------------------------*/
void prettyTxtDeclaration (sDeclaration *s)
{
  switch(s->kind)
  {
    case kDeclaration_Type:     prettyTxtDeclaration_Type    (s);break;
    case kDeclaration_Function: prettyTxtDeclaration_Function(s);break;
    case kDeclaration_Var:      prettyTxtDeclaration_Var     (s);break;
  }
}
void prettyTxtDeclaration_Type (sDeclaration *s)
{
  prettyTab();
  if (s->val.eType.aktiv)
    fprintf(prettyTxtFile,"type td#%s :",s->val.eType.id);
  else
    fprintf(prettyTxtFile,"inakt#type td#%s :",s->val.eType.id);
  prettyTxtType(s->val.eType.type);
}
void prettyTxtDeclaration_Function (sDeclaration *s)
{
  prettyTxtFunction(s->val.eFunction.function);
}
void prettyTxtDeclaration_Var (sDeclaration *s)
{
  prettyTab(); fprintf(prettyTxtFile,"var \n");
  ++tablevel;
  prettyTxtVar_decl_list(s->val.eVar.var_decl_list);
  --tablevel;
}
/*-------------------------------------------------------------*/
void prettyTxtStatement_list (sStatement_list *s)
{
  switch(s->kind)
  {
    case kStatement_list_List:      prettyTxtStatement_list_List     (s);
                                    break;
    case kStatement_list_Statement: prettyTxtStatement_list_Statement(s);
                                    break;
  }
}
void prettyTxtStatement_list_List (sStatement_list *s)
{
  prettyTxtStatement_list(s->val.eList.statement_list);
  prettyTxtStatement(s->val.eList.statement);
}
void prettyTxtStatement_list_Statement (sStatement_list *s)
{
  prettyTxtStatement(s->val.eList.statement);
}
/*-------------------------------------------------------------*/
void prettyTxtStatement (sStatement *s)
{
/*  if (!s->aktiv) return;*/
    switch(s->kind)
  {
    case kStatement_Return:   prettyTxtStatement_Return  (s);break;
    case kStatement_Write:    prettyTxtStatement_Write   (s);break;
    case kStatement_NewLength:prettyTxtStatement_NewLength(s);break;
    case kStatement_New:      prettyTxtStatement_New     (s);break;
    case kStatement_Assign:   prettyTxtStatement_Assign  (s);break;
```

```
    case kStatement_IfElse:   prettyTxtStatement_IfElse  (s);break;
    case kStatement_If:       prettyTxtStatement_If      (s);break;
    case kStatement_While:    prettyTxtStatement_While   (s);break;
    case kStatement_Compound: prettyTxtStatement_Compound(s);break;
    case kStatement_ForTo:    prettyTxtStatement_ForTo   (s);break;
    case kStatement_ForDownto:prettyTxtStatement_ForTo   (s);break;
  }
}
void prettyTxtStatement_Return (sStatement *s)
{
  prettyTab(); prettyAktStm(s); fprintf(prettyTxtFile,"return ");
  prettyTxtExpression(s->val.eReturn.expression);
  fprintf(prettyTxtFile,";\n");
}
void prettyTxtStatement_Write (sStatement *s)
{
  prettyTab(); prettyAktStm(s); fprintf(prettyTxtFile,"write ");
  prettyTxtExpression(s->val.eWrite.expression);
  fprintf(prettyTxtFile,";\n");
}
void prettyTxtStatement_NewLength (sStatement *s)
{
  prettyTab(); prettyAktStm(s); fprintf(prettyTxtFile,"new ");
  prettyTxtVariable(s->val.eNewLength.variable);
  fprintf(prettyTxtFile," of length ");
  prettyTxtExpression(s->val.eNewLength.expression);
  fprintf(prettyTxtFile,";\n");
}
void prettyTxtStatement_New (sStatement *s)
{
  prettyTab(); prettyAktStm(s); fprintf(prettyTxtFile,"new ");
  prettyTxtVariable(s->val.eNew.variable);
  fprintf(prettyTxtFile,";\n");
}
void prettyTxtStatement_Assign (sStatement *s)
{
  prettyTab(); prettyAktStm(s);
  prettyTxtVariable(s->val.eAssign.variable);
  fprintf(prettyTxtFile," = ");
  prettyTxtExpression(s->val.eAssign.expression);
  fprintf(prettyTxtFile,";\n");
}
void prettyTxtStatement_IfElse (sStatement *s)
{
  prettyTab(); prettyAktStm(s); fprintf(prettyTxtFile,"if ");
  prettyTxtExpression(s->val.eIfElse.expression);
  prettyTab(); fprintf(prettyTxtFile," then {\n");
  ++tablevel;
  prettyTxtStatement(s->val.eIfElse.ifStatement);
  --tablevel;
  prettyTab(); fprintf(prettyTxtFile,"}\n");

  prettyTab(); prettyAktStm(s); fprintf(prettyTxtFile,"else {\n");
  ++tablevel;
  prettyTxtStatement(s->val.eIfElse.elseStatement);
  --tablevel;
  prettyTab(); fprintf(prettyTxtFile,"}\n");
}
void prettyTxtStatement_If (sStatement *s)
{
  prettyTab(); prettyAktStm(s); fprintf(prettyTxtFile,"if ");
  prettyTxtExpression(s->val.eIf.expression);
  prettyTab(); fprintf(prettyTxtFile," then {\n");
```

```
  ++tablevel;
  prettyTxtStatement(s->val.eIf.ifStatement);
  --tablevel;
  prettyTab(); fprintf(prettyTxtFile,"}\n");
}
void prettyTxtStatement_While (sStatement *s)
{
  prettyTab(); prettyAktStm(s); fprintf(prettyTxtFile,"while ");
  prettyTxtExpression(s->val.eWhile.expression);
  prettyTab(); fprintf(prettyTxtFile," {\n");
  ++tablevel;
  prettyTxtStatement(s->val.eWhile.statement);
  --tablevel;
  prettyTab(); fprintf(prettyTxtFile,"}\n");
}
void prettyTxtStatement_Compound (sStatement *s)
{
  prettyTab(); prettyAktStm(s); fprintf(prettyTxtFile,"#{\n");
  ++tablevel;
  prettyTxtStatement_list(s->val.eCompound.statement_list);
  --tablevel;
  prettyTab(); fprintf(prettyTxtFile,"#}\n");
}
void prettyTxtStatement_ForTo (sStatement *s)
{
  prettyTab(); prettyAktStm(s); fprintf(prettyTxtFile,"for ");
  prettyTxtVariable(s->val.eForTo.variable);
  fprintf(prettyTxtFile," = ");
  prettyTxtExpression(s->val.eForTo.expressionInit);
  if (s->kind == kStatement_ForTo)
    fprintf(prettyTxtFile," to ");
  else
    fprintf(prettyTxtFile," downto ");
  prettyTxtExpression(s->val.eForTo.expressionTo);
  prettyTab(); fprintf(prettyTxtFile," {\n");
  ++tablevel;
  prettyTxtStatement(s->val.eForTo.statement);
  --tablevel;
  prettyTab(); fprintf(prettyTxtFile,"}\n");
}
/*-------------------------------------------------------------------*/
void prettyTxtVariable (sVariable *s)
{
  switch(s->kind)
  {
    case kVariable_Id:      prettyTxtVariable_Id     (s);break;
    case kVariable_Indexed: prettyTxtVariable_Indexed(s);break;
    case kVariable_Struct:  prettyTxtVariable_Struct (s);break;
  }
}
void prettyTxtVariable_Id (sVariable *s)
{
  fprintf(prettyTxtFile,"v_%s#%s",type_Kind_Txt(s->type),s->val.eId.id);
}
void prettyTxtVariable_Indexed (sVariable *s)
{
  prettyTxtVariable(s->val.eIndexed.variable);
    fprintf(prettyTxtFile,"[");
  prettyTxtExpression(s->val.eIndexed.expression);
    fprintf(prettyTxtFile,"]");
}
void prettyTxtVariable_Struct (sVariable *s)
{
```

```
  prettyTxtVariable(s->val.eStruct.variable);
  fprintf(prettyTxtFile,".v_%s#%s",type_Kind_Txt(s->type),s->val.eStruct.id);
}
/*------------------------------------------------------------------*/
void prettyTxtExpression (sExpression *s)
{
    switch(s->kind)
  {
    case kExp_Eq:   prettyTxtExpression_Dyadisk (s,"==");break;
    case kExp_Neq:  prettyTxtExpression_Dyadisk (s,"!=");break;
    case kExp_Lt:   prettyTxtExpression_Dyadisk (s,"<"); break;
    case kExp_Gt:   prettyTxtExpression_Dyadisk (s,">"); break;
    case kExp_Lteq: prettyTxtExpression_Dyadisk (s,"<=");break;
    case kExp_Gteq: prettyTxtExpression_Dyadisk (s,">=");break;
    case kExp_Add:  prettyTxtExpression_Dyadisk (s,"+"); break;
    case kExp_Sub:  prettyTxtExpression_Dyadisk (s,"-"); break;
    case kExp_Mul:  prettyTxtExpression_Dyadisk (s,"*"); break;
    case kExp_Div:  prettyTxtExpression_Dyadisk (s,"/"); break;
    case kExp_Or:   prettyTxtExpression_Dyadisk (s,"||");break;
    case kExp_And:  prettyTxtExpression_Dyadisk (s,"&&");break;
    case kExp_Term: prettyTxtExpression_Term    (s);     break;

  }
}
void prettyTxtExpression_Dyadisk (sExpression *s, char const *operation)
{
  fprintf(prettyTxtFile,"t_%s#( ",type_Kind_Txt(s->type));
  prettyTxtExpression(s->val.eLeftRight.expressionLeft);
  fprintf(prettyTxtFile," %s ",operation);
  prettyTxtExpression(s->val.eLeftRight.expressionRight);
  fprintf(prettyTxtFile," )");
}
void prettyTxtExpression_Term (sExpression *s)
{
  prettyTxtTerm(s->val.eTerm.term);
}
/*------------------------------------------------------------------*/
void prettyTxtTerm (sTerm *s)
{
  switch(s->kind)
  {
    case kTerm_Variable:  prettyTxtTerm_Variable  (s);break;
    case kTerm_Call:      prettyTxtTerm_Call      (s);break;
    case kTerm_Parantes:  prettyTxtTerm_Parantes  (s);break;
    case kTerm_Not:       prettyTxtTerm_Not       (s);break;
    case kTerm_Numeric:   prettyTxtTerm_Numeric   (s);break;
    case kTerm_Num:       prettyTxtTerm_Num       (s);break;
    case kTerm_True:      prettyTxtTerm_True      (s);break;
    case kTerm_False:     prettyTxtTerm_False     (s);break;
    case kTerm_Null:      prettyTxtTerm_Null      (s);break;
  }
}
void prettyTxtTerm_Variable (sTerm *s)
{
  prettyTxtVariable(s->val.eVariable.variable);
}
void prettyTxtTerm_Call (sTerm *s)
{
  fprintf(prettyTxtFile,"f_%s#%s (",type_Kind_Txt(s->type),s->val.eCall.id);
  prettyTxtAct_list(s->val.eCall.act_list);
  fprintf(prettyTxtFile,")");
}
void prettyTxtTerm_Parantes (sTerm *s)
```

```
{
  fprintf(prettyTxtFile," (");
  prettyTxtExpression(s->val.eParantes.expression);
  fprintf(prettyTxtFile,") ");
}
void prettyTxtTerm_Not (sTerm *s)
{
  fprintf(prettyTxtFile," !(");
  prettyTxtTerm(s->val.eNot.term);
  fprintf(prettyTxtFile,") ");
}
void prettyTxtTerm_Numeric (sTerm *s)
{
  fprintf(prettyTxtFile," |");
  prettyTxtExpression(s->val.eNumeric.expression);
  fprintf(prettyTxtFile,"| ");
}
void prettyTxtTerm_Num (sTerm *s)
{
  fprintf(prettyTxtFile,"c#%d",s->val.eNum.num);
}
void prettyTxtTerm_True (sTerm *s)
{
  fprintf(prettyTxtFile,"c#true");
}
void prettyTxtTerm_False (sTerm *s)
{
  fprintf(prettyTxtFile,"c#false");
}
void prettyTxtTerm_Null (sTerm *s)
{
  fprintf(prettyTxtFile,"c#null");
}
/*------------------------------------------------------------------*/
void prettyTxtAct_list (sAct_list *s)
{
  switch(s->kind)
  {
    case kAct_list_List:  prettyTxtAct_list_List  (s);break;
    case kAct_list_Empty: prettyTxtAct_list_Empty (s);break;
  }
}
void prettyTxtAct_list_List (sAct_list *s)
{
  prettyTxtExp_list(s->val.eList.exp_list);
}
void prettyTxtAct_list_Empty (sAct_list *s)
{
}
/*------------------------------------------------------------------*/
void prettyTxtExp_list (sExp_list *s)
{
    switch(s->kind)
  {
    case kExp_list_List:       prettyTxtExp_list_List       (s);break;
    case kExp_list_Expression: prettyTxtExp_list_Expression (s);break;
  }
}
void prettyTxtExp_list_List (sExp_list *s)
{
  prettyTxtExp_list(s->val.eList.exp_list);
  fprintf(prettyTxtFile,", ");
  prettyTxtExpression(s->val.eList.expression);
```

```
}
void prettyTxtExp_list_Expression (sExp_list *s)
{
  prettyTxtExpression(s->val.eExpression.expression);
}
```

Funktioner til udskrift af tony AST i XML-format

tonyAST_Pretty_XML.h

```
/* ------------------------------------------------------------------------
  Definition af funktioner til udskrift af TONY abstract
  syntax tree i XML format
  Programmør: Bjørk Busch
  Historik: 2005.03.26  Udgave til rapport 3 færdig
  Historik: 2005.05.10  kosmetik i source
------------------------------------------------------------------------*/
void openprettyXmlXml();
void prettyXmlProgram (sProgram *s);
void prettyXmlFunction (sFunction *s);
void prettyXmlHead (sHead *s);
void prettyXmlTail (sTail *s);
/*----------------------------------------------------------------*/
void prettyXmlType (sType *s);
void prettyXmlType_Id (sType *s);
void prettyXmlType_Int (sType *s);
void prettyXmlType_Bool (sType *s);
void prettyXmlType_Array (sType *s);
void prettyXmlType_Record (sType *s);
/*----------------------------------------------------------------*/
void prettyXmlPar_decl_list (sPar_decl_list *s);
void prettyXmlPar_decl_list_List (sPar_decl_list *s);
void prettyXmlPar_decl_list_Empty (sPar_decl_list *s);
/*----------------------------------------------------------------*/
void prettyXmlVar_decl_list (sVar_decl_list *s);
void prettyXmlVar_decl_list_List (sVar_decl_list *s);
void prettyXmlVar_decl_list_Var (sVar_decl_list *s);
/*----------------------------------------------------------------*/
void prettyXmlVar_type (sVar_type *s);
/*----------------------------------------------------------------*/
void prettyXmlBody (sBody *s);
/*----------------------------------------------------------------*/
void prettyXmlDecl_list (sDecl_list *s);
void prettyXmlDecl_list_List (sDecl_list *s);
void prettyXmlDecl_list_Empty (sDecl_list *s);
/*----------------------------------------------------------------*/
void prettyXmlDeclaration (sDeclaration *s);
void prettyXmlDeclaration_Type (sDeclaration *s);
void prettyXmlDeclaration_Function (sDeclaration *s);
void prettyXmlDeclaration_Var (sDeclaration *s);
/*----------------------------------------------------------------*/
void prettyXmlStatement_list (sStatement_list *s);
void prettyXmlStatement_list_List (sStatement_list *s);
void prettyXmlStatement_list_Statement (sStatement_list *s);
/*----------------------------------------------------------------*/
void prettyXmlStatement (sStatement *s);
void prettyXmlStatement_Return (sStatement *s);
void prettyXmlStatement_Write (sStatement *s);
void prettyXmlStatement_NewLength (sStatement *s);
void prettyXmlStatement_New (sStatement *s);
void prettyXmlStatement_Assign (sStatement *s);
void prettyXmlStatement_IfElse (sStatement *s);
```

```
void prettyXmlStatement_If (sStatement *s);
void prettyXmlStatement_While (sStatement *s);
void prettyXmlStatement_Compound (sStatement *s);
void prettyXmlStatement_ForTo (sStatement *s);
/*-----------------------------------------------------------------*/
void prettyXmlVariable (sVariable *s);
void prettyXmlVariable_Id (sVariable *s);
void prettyXmlVariable_Indexed (sVariable *s);
void prettyXmlVariable_Struct (sVariable *s);
/*-----------------------------------------------------------------*/
void prettyXmlExpression (sExpression *s);
void prettyXmlExpression_Dyadisk (sExpression *s, char const *operation);
void prettyXmlExpression_Term (sExpression *s);
/*-----------------------------------------------------------------*/
void prettyXmlTerm (sTerm *s);
void prettyXmlTerm_Variable (sTerm *s);
void prettyXmlTerm_Call (sTerm *s);
void prettyXmlTerm_Parantes (sTerm *s);
void prettyXmlTerm_Not (sTerm *s);
void prettyXmlTerm_Numeric (sTerm *s);
void prettyXmlTerm_Num (sTerm *s);
void prettyXmlTerm_True (sTerm *s);
void prettyXmlTerm_False (sTerm *s);
void prettyXmlTerm_Null (sTerm *s);
/*-----------------------------------------------------------------*/
void prettyXmlAct_list (sAct_list *s);
void prettyXmlAct_list_List (sAct_list *s);
void prettyXmlAct_list_Empty (sAct_list *s);
/*-----------------------------------------------------------------*/
void prettyXmlExp_list (sExp_list *s);
void prettyXmlExp_list_List (sExp_list *s);
void prettyXmlExp_list_Expression (sExp_list *s);
```

tonyAST_Pretty_XML.c

```
/* ------------------------------------------------------------------------
  Realisering af funktioner til udskrift af TONY abstract
  syntax tree i XML format
  Programmør: Bjørk Busch
  Historik: 2005.03.26  Udgave til rapport 3 færdig
  Historik: 2005.05.10  kosmetik i source
------------------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include "bjbu_std.h"
#include "symbol.h"
#include "tonyAbstractAsm.h"
#include "tonyAST.h"
#include "tonyAST_Make.h"
#include "tonyAST_Pretty_XML.h"

#define prettyXml_FILENAME "tonyprettyXml.xml"
FILE *prettyXmlFile;
extern FILE *msgFile;

void openprettyXml()
{
  prettyXmlFile = fopen(prettyXml_FILENAME,"w");
  if (prettyXmlFile==NULL){
    fprintf(msgFile, "alvorligFejl ved open af xmlfile <%s>\n",
                     prettyXml_FILENAME);
    exit(101);
  }
```

```
}

void prettyXmlProgram (sProgram *s)
{
  openprettyXml();
  fprintf(msgFile,
     "TONY abstract syntax tree udskrives i XML-format til <%s>\n",
     prettyXml_FILENAME);
  fprintf(prettyXmlFile,"<program linienr='l %d'>\n",s->linienr);
  prettyXmlBody(s->val.eProgram.body);
  fprintf(prettyXmlFile,"</program>\n");
  fclose(prettyXmlFile);
}
void prettyXmlFunction (sFunction *s)
{
  fprintf(prettyXmlFile,"<function linienr='l %d'>\n",s->linienr);
  prettyXmlHead(s->val.eFunction.head);
  prettyXmlBody(s->val.eFunction.body);
  prettyXmlTail(s->val.eFunction.tail);
  fprintf(prettyXmlFile,"</function>\n");
}
void prettyXmlHead (sHead *s)
{
  fprintf(prettyXmlFile,"<head linienr='l %d'>\n",s->linienr);
  fprintf(prettyXmlFile,"<id>%s</id>\n",s->val.eHead.id);
  prettyXmlPar_decl_list(s->val.eHead.par_decl_list);
  prettyXmlType(s->val.eHead.type);
  fprintf(prettyXmlFile,"</head>\n");
}
void prettyXmlTail (sTail *s)
{
  fprintf(prettyXmlFile,"<tail linienr='l %d'>\n",s->linienr);
  fprintf(prettyXmlFile,"<id>%s</id>\n",s->val.eTail.id);
  fprintf(prettyXmlFile,"</tail>\n");
}
/*------------------------------------------------------------------*/
void prettyXmlType (sType *s)
{
  fprintf(prettyXmlFile,"<type linienr='l %d'>\n",s->linienr);
  switch(s->kind)
  {
    case kType_Id:     prettyXmlType_Id    (s);break;
    case kType_Int:    prettyXmlType_Int   (s);break;
    case kType_Bool:   prettyXmlType_Bool  (s);break;
    case kType_Array:  prettyXmlType_Array (s);break;
    case kType_Record: prettyXmlType_Record(s);break;
  }
  fprintf(prettyXmlFile,"</type>\n");
}

void prettyXmlType_Id (sType *s)
{
  fprintf(prettyXmlFile,"<id>%s</id>\n",s->val.eId.id);
}
void prettyXmlType_Int (sType *s)
{
  fprintf(prettyXmlFile,"<type>int</type>\n");
}
void prettyXmlType_Bool (sType *s)
{
  fprintf(prettyXmlFile,"<type>bool</type>\n");
}
void prettyXmlType_Array (sType *s)
```

```
{
  fprintf(prettyXmlFile,"<array_of>\n");
  prettyXmlType(s->val.eArray.type);
  fprintf(prettyXmlFile,"</array_of>\n");
}
void prettyXmlType_Record (sType *s)
{
  fprintf(prettyXmlFile,"<record_of>\n");
  prettyXmlVar_decl_list(s->val.eRecord.var_decl_list);
  fprintf(prettyXmlFile,"</record_of>\n");
}
/*-------------------------------------------------------------------*/
void prettyXmlPar_decl_list (sPar_decl_list *s)
{
  switch(s->kind)
  {
    case kPar_decl_list_List:  prettyXmlPar_decl_list_List (s);break;
    case kPar_decl_list_Empty: prettyXmlPar_decl_list_Empty(s);break;
  }
}
void prettyXmlPar_decl_list_List (sPar_decl_list *s)
{
  fprintf(prettyXmlFile,"<Par_decl_list linienr='l %d'>\n",s->linienr);
  prettyXmlVar_decl_list(s->val.eList.var_decl_list);
  fprintf(prettyXmlFile,"</Par_decl_list>\n");
}
void prettyXmlPar_decl_list_Empty (sPar_decl_list *s)
{
}
/*-------------------------------------------------------------------*/
void prettyXmlVar_decl_list (sVar_decl_list *s)
{
  switch(s->kind)
  {
    case kVar_decl_list_List: prettyXmlVar_decl_list_List (s);break;
    case kVar_decl_list_Var:  prettyXmlVar_decl_list_Var  (s);break;
  }
}
void prettyXmlVar_decl_list_List (sVar_decl_list *s)
{
  prettyXmlVar_decl_list(s->val.eList.var_decl_list);
/*  fprintf(prettyXmlFile,"<seperator></seperator>\n",s->linienr);*/
  prettyXmlVar_type(s->val.eList.var_type);
}
void prettyXmlVar_decl_list_Var (sVar_decl_list *s)
{
  prettyXmlVar_type(s->val.eVar.var_type);
}
/*-------------------------------------------------------------------*/
void prettyXmlVar_type (sVar_type *s)
{
  fprintf(prettyXmlFile,"<var_type linienr='l %d'>\n",s->linienr);
  fprintf(prettyXmlFile,"<id>%s</id>\n",s->val.eVar.id);
  prettyXmlType(s->val.eVar.type);
  fprintf(prettyXmlFile,"</var_type>\n");
}
/*-------------------------------------------------------------------*/
void prettyXmlBody (sBody *s)
{
  fprintf(prettyXmlFile,"<boddy linienr='l %d'>\n",s->linienr);

  fprintf(prettyXmlFile,"<decl_List>\n");
  prettyXmlDecl_list(s->val.eBody.decl_list);
```

```
  fprintf(prettyXmlFile,"</decl_List>\n");

  fprintf(prettyXmlFile,"<statement_list>\n");
  prettyXmlStatement_list(s->val.eBody.statement_list);
  fprintf(prettyXmlFile,"</statement_list>\n");

  fprintf(prettyXmlFile,"</boddy>\n");
}
/*-----------------------------------------------------------------*/
void prettyXmlDecl_list (sDecl_list *s)
{
  switch(s->kind)
  {
    case kDecl_list_List: prettyXmlDecl_list_List (s);break;
    case kDecl_list_Empty:  prettyXmlDecl_list_Empty  (s);break;
  }
}
void prettyXmlDecl_list_List (sDecl_list *s)
{
  prettyXmlDecl_list(s->val.eList.decl_list);
  prettyXmlDeclaration(s->val.eList.declaration);
}
void prettyXmlDecl_list_Empty (sDecl_list *s)
{
}
/*-----------------------------------------------------------------*/
void prettyXmlDeclaration (sDeclaration *s)
{
  fprintf(prettyXmlFile,"<declaration linienr='l %d'>\n",s->linienr);
  switch(s->kind)
  {
    case kDeclaration_Type:     prettyXmlDeclaration_Type    (s);break;
    case kDeclaration_Function: prettyXmlDeclaration_Function(s);break;
    case kDeclaration_Var:      prettyXmlDeclaration_Var     (s);break;
  }
  fprintf(prettyXmlFile,"</declaration>\n");
}
void prettyXmlDeclaration_Type (sDeclaration *s)
{
  if (!s->val.eType.aktiv) return;  /* deaktiveret i weed-fase */
    fprintf(prettyXmlFile,"<id>%s</id>\n",s->val.eType.id);
  prettyXmlType(s->val.eType.type);
}
void prettyXmlDeclaration_Function (sDeclaration *s)
{
  prettyXmlFunction(s->val.eFunction.function);
}
void prettyXmlDeclaration_Var (sDeclaration *s)
{
  prettyXmlVar_decl_list(s->val.eVar.var_decl_list);
}
/*-----------------------------------------------------------------*/
void prettyXmlStatement_list (sStatement_list *s)
{
  switch(s->kind)
  {
    case kStatement_list_List:      prettyXmlStatement_list_List    (s);
                                    break;
    case kStatement_list_Statement: prettyXmlStatement_list_Statement (s);
                                    break;
  }
}
void prettyXmlStatement_list_List (sStatement_list *s)
```

```
{
  prettyXmlStatement_list(s->val.eList.statement_list);
  prettyXmlStatement(s->val.eList.statement);
}
void prettyXmlStatement_list_Statement (sStatement_list *s)
{
  prettyXmlStatement(s->val.eList.statement);
}
/*-----------------------------------------------------------------*/
void prettyXmlStatement (sStatement *s)
{
  if (!s->aktiv) return;
  fprintf(prettyXmlFile,"<statement linienr='l %d'>\n",s->linienr);
  switch(s->kind)
  {
    case kStatement_Return:   prettyXmlStatement_Return (s);break;
    case kStatement_Write:    prettyXmlStatement_Write  (s);break;
    case kStatement_NewLength:prettyXmlStatement_NewLength(s);break;
    case kStatement_New:      prettyXmlStatement_New    (s);break;
    case kStatement_Assign:   prettyXmlStatement_Assign (s);break;
    case kStatement_IfElse:   prettyXmlStatement_IfElse (s);break;
    case kStatement_If:       prettyXmlStatement_If     (s);break;
    case kStatement_While:    prettyXmlStatement_While  (s);break;
    case kStatement_Compound: prettyXmlStatement_Compound (s);break;
    case kStatement_ForTo:    prettyXmlStatement_ForTo  (s);break;
    case kStatement_ForDownto:prettyXmlStatement_ForTo  (s);break;
  }
  fprintf(prettyXmlFile,"</statement>\n");
}
void prettyXmlStatement_Return (sStatement *s)
{
  fprintf(prettyXmlFile,"<return linienr='l %d'>\n",s->linienr);
  prettyXmlExpression(s->val.eReturn.expression);
  fprintf(prettyXmlFile,"</return>\n");
}
void prettyXmlStatement_Write (sStatement *s)
{
  fprintf(prettyXmlFile,"<write linienr='l %d'>\n",s->linienr);
  prettyXmlExpression(s->val.eWrite.expression);
  fprintf(prettyXmlFile,"</write>\n");
}
void prettyXmlStatement_NewLength (sStatement *s)
{
  fprintf(prettyXmlFile,"<new_length linienr='l %d'>\n",s->linienr);
  prettyXmlVariable(s->val.eNewLength.variable);
  prettyXmlExpression(s->val.eNewLength.expression);
  fprintf(prettyXmlFile,"</new_length>\n");
}
void prettyXmlStatement_New (sStatement *s)
{
  fprintf(prettyXmlFile,"<new linienr='l %d'>\n",s->linienr);
  prettyXmlVariable(s->val.eNew.variable);
  fprintf(prettyXmlFile,"</new>\n");
}
void prettyXmlStatement_Assign (sStatement *s)
{
  fprintf(prettyXmlFile,"<assign linienr='l %d'>\n",s->linienr);
  prettyXmlVariable(s->val.eAssign.variable);
  prettyXmlExpression(s->val.eAssign.expression);
  fprintf(prettyXmlFile,"</assign>\n");
}
void prettyXmlStatement_IfElse (sStatement *s)
{
```

```
  fprintf(prettyXmlFile,"<if_else linienr='l %d'>\n",s->linienr);
  prettyXmlExpression(s->val.eIfElse.expression);

  fprintf(prettyXmlFile,"<if_statements>\n");
  prettyXmlStatement(s->val.eIfElse.ifStatement);
  fprintf(prettyXmlFile,"</if_statements>\n");

  fprintf(prettyXmlFile,"<else_statements>\n");
  prettyXmlStatement(s->val.eIfElse.elseStatement);
  fprintf(prettyXmlFile,"</else_statements>\n");

  fprintf(prettyXmlFile,"</if_else>\n");
}
void prettyXmlStatement_If (sStatement *s)
{
  fprintf(prettyXmlFile,"<if linienr='l %d'>\n",s->linienr);
  prettyXmlExpression(s->val.eIf.expression);
  prettyXmlStatement(s->val.eIf.ifStatement);
  fprintf(prettyXmlFile,"</if>\n");
}
void prettyXmlStatement_While (sStatement *s)
{
  fprintf(prettyXmlFile,"<while linienr='l %d'>\n",s->linienr);
  prettyXmlExpression(s->val.eWhile.expression);
  prettyXmlStatement(s->val.eWhile.statement);
  fprintf(prettyXmlFile,"</while>\n");
}
void prettyXmlStatement_Compound (sStatement *s)
{
  fprintf(prettyXmlFile,"<compound linienr='l %d'>\n",s->linienr);
  prettyXmlStatement_list(s->val.eCompound.statement_list);
  fprintf(prettyXmlFile,"</compound>\n");
}
void prettyXmlStatement_ForTo (sStatement *s)
{
  if (s->kind == kStatement_ForTo)
    fprintf(prettyXmlFile,"<for_to linienr='l %d'>\n",s->linienr);
  else
    fprintf(prettyXmlFile,"<for_downto linienr='l %d'>\n",s->linienr);
  prettyXmlVariable(s->val.eForTo.variable);
  prettyXmlExpression(s->val.eForTo.expressionInit);
  prettyXmlExpression(s->val.eForTo.expressionTo);
  prettyXmlStatement(s->val.eForTo.statement);
  if (s->kind == kStatement_ForTo)
    fprintf(prettyXmlFile,"</for_to>\n");
  else
    fprintf(prettyXmlFile,"</for_downto>\n");
}
/*-----------------------------------------------------------------*/
void prettyXmlVariable (sVariable *s)
{
  switch(s->kind)
  {
    case kVariable_Id:      prettyXmlVariable_Id      (s);break;
    case kVariable_Indexed: prettyXmlVariable_Indexed (s);break;
    case kVariable_Struct:  prettyXmlVariable_Struct  (s);break;
  }
}
void prettyXmlVariable_Id (sVariable *s)
{
    fprintf(prettyXmlFile,
      "<variabel_id type='%s'>%s</variabel_id>\n",
      type_Kind_Txt(s->type),s->val.eId.id);
```

```
}
void prettyXmlVariable_Indexed (sVariable *s)
{
  fprintf(prettyXmlFile,
    "<indexed_variable linienr='l %d' type='%s'>\n",
    s->linienr,type_Kind_Txt(s->type));
  prettyXmlVariable(s->val.eIndexed.variable);
  prettyXmlExpression(s->val.eIndexed.expression);
  fprintf(prettyXmlFile,"</indexed_variable>\n");
}
void prettyXmlVariable_Struct (sVariable *s)
{
  fprintf(prettyXmlFile,
    "<struct_variable linienr='l %d' type='%s'>\n",
    s->linienr,type_Kind_Txt(s->type));
  prettyXmlVariable(s->val.eStruct.variable);
  fprintf(prettyXmlFile,
    "<variabel_id type='%s'>%s</variabel_id>\n",
    type_Kind_Txt(s->type),s->val.eStruct.id);
  fprintf(prettyXmlFile,"</struct_variable>\n");
}
/*-----------------------------------------------------------------*/
void prettyXmlExpression (sExpression *s)
{
  switch(s->kind)
  {
    case kExp_Eq:   prettyXmlExpression_Dyadisk (s,"eq");  break;
    case kExp_Neq:  prettyXmlExpression_Dyadisk (s,"neq"); break;
    case kExp_Lt:   prettyXmlExpression_Dyadisk (s,"lt");  break;
    case kExp_Gt:   prettyXmlExpression_Dyadisk (s,"gt");  break;
    case kExp_Lteq: prettyXmlExpression_Dyadisk (s,"lteq");break;
    case kExp_Gteq: prettyXmlExpression_Dyadisk (s,"gteq");break;
    case kExp_Add:  prettyXmlExpression_Dyadisk (s,"add"); break;
    case kExp_Sub:  prettyXmlExpression_Dyadisk (s,"sub"); break;
    case kExp_Mul:  prettyXmlExpression_Dyadisk (s,"mul"); break;
    case kExp_Div:  prettyXmlExpression_Dyadisk (s,"div"); break;
    case kExp_Or:   prettyXmlExpression_Dyadisk (s,"or");  break;
    case kExp_And:  prettyXmlExpression_Dyadisk (s,"and"); break;
    case kExp_Term: prettyXmlExpression_Term    (s);       break;
  }
}
void prettyXmlExpression_Dyadisk (sExpression *s, char const *operation)
{
  fprintf(prettyXmlFile,
    "<expression operation='%s' linienr='l %d' type='%s'>\n",
    operation,s->linienr,type_Kind_Txt(s->type));
  prettyXmlExpression(s->val.eLeftRight.expressionLeft);
  prettyXmlExpression(s->val.eLeftRight.expressionRight);
  fprintf(prettyXmlFile,"</expression>\n");
}
void prettyXmlExpression_Term (sExpression *s)
{
  fprintf(prettyXmlFile,
    "<expression linienr='l %d' type='%s'>\n",
    s->linienr,type_Kind_Txt(s->type));
  prettyXmlTerm(s->val.eTerm.term);
  fprintf(prettyXmlFile,"</expression>\n");
}
/*-----------------------------------------------------------------*/
void prettyXmlTerm (sTerm *s)
{
  switch(s->kind)
  {
```

```
    case kTerm_Variable:  prettyXmlTerm_Variable  (s);break;
    case kTerm_Call:      prettyXmlTerm_Call      (s);break;
    case kTerm_Parantes:  prettyXmlTerm_Parantes  (s);break;
    case kTerm_Not:       prettyXmlTerm_Not       (s);break;
    case kTerm_Numeric:   prettyXmlTerm_Numeric   (s);break;
    case kTerm_Num:       prettyXmlTerm_Num       (s);break;
    case kTerm_True:      prettyXmlTerm_True      (s);break;
    case kTerm_False:     prettyXmlTerm_False     (s);break;
    case kTerm_Null:      prettyXmlTerm_Null      (s);break;
  }
}
void prettyXmlTerm_Variable (sTerm *s)
{
  prettyXmlVariable(s->val.eVariable.variable);
}
void prettyXmlTerm_Call (sTerm *s)
{
  fprintf(prettyXmlFile,
    "<call linienr='l %d' type='%s'>\n",
    s->linienr,type_Kind_Txt(s->type));
  fprintf(prettyXmlFile,"<id>%s</id>\n",s->val.eCall.id);
  prettyXmlAct_list(s->val.eCall.act_list);
  fprintf(prettyXmlFile,"</call>\n");
}
void prettyXmlTerm_Parantes (sTerm *s)
{
  fprintf(prettyXmlFile,
    "<parantes linienr='l %d' type='%s'>\n",
    s->linienr,type_Kind_Txt(s->type));
  prettyXmlExpression(s->val.eParantes.expression);
  fprintf(prettyXmlFile,"</parantes>\n");
}
void prettyXmlTerm_Not (sTerm *s)
{
  fprintf(prettyXmlFile,
    "<not linienr='l %d' type='%s'>\n",
    s->linienr,type_Kind_Txt(s->type));
  prettyXmlTerm(s->val.eNot.term);
  fprintf(prettyXmlFile,"</not>\n");
}
void prettyXmlTerm_Numeric (sTerm *s)
{
  fprintf(prettyXmlFile,
    "<numeric linienr='l %d' type='%s'>\n",
    s->linienr,type_Kind_Txt(s->type));
  prettyXmlExpression(s->val.eNumeric.expression);
  fprintf(prettyXmlFile,"</numeric>\n");
}
void prettyXmlTerm_Num (sTerm *s)
{
  fprintf(prettyXmlFile,"<const>%d</const>\n",s->val.eNum.num);
}
void prettyXmlTerm_True (sTerm *s)
{
  fprintf(prettyXmlFile,"<const>true</const>\n");
}
void prettyXmlTerm_False (sTerm *s)
{
  fprintf(prettyXmlFile,"<const>false</const>\n");
}
void prettyXmlTerm_Null (sTerm *s)
{
  fprintf(prettyXmlFile,"<const>null</const>\n");
```

```
}
/*-------------------------------------------------------------*/
void prettyXmlAct_list (sAct_list *s)
{
  switch(s->kind)
  {
    case kAct_list_List:  prettyXmlAct_list_List  (s);break;
    case kAct_list_Empty: prettyXmlAct_list_Empty (s);break;
  }
}
void prettyXmlAct_list_List (sAct_list *s)
{
  fprintf(prettyXmlFile,"<act_list linienr='l %d'>\n",s->linienr);
  prettyXmlExp_list(s->val.eList.exp_list);
  fprintf(prettyXmlFile,"</act_list>\n");
}
void prettyXmlAct_list_Empty (sAct_list *s)
{
}
/*-------------------------------------------------------------*/
void prettyXmlExp_list (sExp_list *s)
{
  switch(s->kind)
  {
    case kExp_list_List:       prettyXmlExp_list_List      (s);break;
    case kExp_list_Expression: prettyXmlExp_list_Expression(s);break;
  }
}
void prettyXmlExp_list_List (sExp_list *s)
{
  prettyXmlExp_list(s->val.eList.exp_list);
  prettyXmlExpression(s->val.eList.expression);
}
void prettyXmlExp_list_Expression (sExp_list *s)
{
  prettyXmlExpression(s->val.eExpression.expression);
}
```

Funktioner til udskrift af symboltabeller med krydsreferencer

tonySymbols_Xref.c

```
/* ---------------------------------------------------------------
  Realisering af funktioner til opbygning af symboltabeller
  Programmør: Bjørk Busch
  Historik: 2005.03.26
  Historik: 2005.05.10  kosmetik i source
---------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include "bjbu_std.h"
#include "symbol.h"
#include "tonyAbstractAsm.h"
#include "tonyAST.h"


#define SYMBOL_XREF_FILENAME "tonysymbol_xref.txt"
FILE *symbolXrefFile;
extern FILE *msgFile;

void symbolsXrefProgram (sProgram *s);
void symbolsXrefFunction (sFunction *s);
void symbolsXrefType (sType *s, char *name, int linienr);
```

```
void symbolsXrefType_Id (sType *s, char *name, int linienr);
void symbolsXrefType_Int (sType *s, char *name, int linienr);
void symbolsXrefType_Bool (sType *s, char *name, int linienr);
void symbolsXrefType_Array (sType *s, char *name, int linienr);
void symbolsXrefType_Record (sType *s, char *name, int linienr);
void symbolsXrefPar_decl_list (sPar_decl_list *s);
void symbolsXrefPar_decl_list_List (sPar_decl_list *s);
void symbolsXrefPar_decl_list_Empty (sPar_decl_list *s);
void symbolsXrefVar_decl_list (sVar_decl_list *s);
void symbolsXrefVar_decl_list_List (sVar_decl_list *s);
void symbolsXrefVar_decl_list_Var (sVar_decl_list *s);
void symbolsXrefVar_type (sVar_type *s);
void symbolsXrefBody (sBody *s);
void symbolsXrefDecl_list (sDecl_list *s);
void symbolsXrefDecl_list_List (sDecl_list *s);
void symbolsXrefDecl_list_Empty (sDecl_list *s);
void symbolsXrefDeclaration (sDeclaration *s);
void symbolsXrefDeclaration_Type (sDeclaration *s);
void symbolsXrefDeclaration_Function (sDeclaration *s);
void symbolsXrefDeclaration_Var (sDeclaration *s);
/*-----------------------------------------------------------------*/
void openSymbolXrefFile();
void printSymbolXrefOneSYMBOL_Xref (XREF *xref);
void printSymbolXrefOneSYMBOL (SYMBOL *s);
void printSymbolXrefOneSymbolTable(SymbolTable *t, char *name, int linienr);



/*-----------------------------------------------------------------
Implementering
-----------------------------------------------------------------*/
void symbolsXrefProgram (sProgram *s)
{
    openSymbolXrefFile();
  fprintf(msgFile,
    "TONY SYMBOL XREF udskrives til %s\n",SYMBOL_XREF_FILENAME);
  printSymbolXrefOneSymbolTable(s->symbolTable,"MAIN",0);
  symbolsXrefBody(s->val.eProgram.body);
    fclose(symbolXrefFile);
}
void symbolsXrefFunction (sFunction *s)
{
  printSymbolXrefOneSymbolTable(
                 s->symbolTable,s->val.eFunction.head->val.eHead.id,
                 s->val.eFunction.head->linienr);
    symbolsXrefBody(s->val.eFunction.body);
}
/*-----------------------------------------------------------------*/
void symbolsXrefType (sType *s, char *name, int linienr)
{
  switch(s->kind)
  {
    case kType_Id:     symbolsXrefType_Id    (s, name, linienr);break;
    case kType_Int:    symbolsXrefType_Int   (s, name, linienr);break;
    case kType_Bool:   symbolsXrefType_Bool  (s, name, linienr);break;
    case kType_Array:  symbolsXrefType_Array (s, name, linienr);break;
    case kType_Record: symbolsXrefType_Record(s, name, linienr);break;
  }
}

void symbolsXrefType_Id (sType *s, char *name, int linienr)
{
}
```

```
void symbolsXrefType_Int (sType *s, char *name, int linienr)
{
}
void symbolsXrefType_Bool (sType *s, char *name, int linienr)
{
}
void symbolsXrefType_Array (sType *s, char *name, int linienr)
{
    symbolsXrefType(s->val.eArray.type, name, linienr);
}
void symbolsXrefType_Record (sType *s, char *name, int linienr)
{
  printSymbolXrefOneSymbolTable(s->val.eRecord.symbolTable,name, linienr);
    symbolsXrefVar_decl_list(s->val.eRecord.var_decl_list);
}
/*-------------------------------------------------------------------*/
void symbolsXrefPar_decl_list (sPar_decl_list *s)
{
  switch(s->kind)
  {
    case kPar_decl_list_List:  symbolsXrefPar_decl_list_List (s);break;
    case kPar_decl_list_Empty: symbolsXrefPar_decl_list_Empty(s);break;
  }
}
void symbolsXrefPar_decl_list_List (sPar_decl_list *s)
{
  symbolsXrefVar_decl_list(s->val.eList.var_decl_list);
}
void symbolsXrefPar_decl_list_Empty (sPar_decl_list *s)
{
}
/*-------------------------------------------------------------------*/
void symbolsXrefVar_decl_list (sVar_decl_list *s)
{
  switch(s->kind)
  {
    case kVar_decl_list_List: symbolsXrefVar_decl_list_List (s);break;
    case kVar_decl_list_Var:  symbolsXrefVar_decl_list_Var  (s);break;
  }
}
void symbolsXrefVar_decl_list_List (sVar_decl_list *s)
{
  symbolsXrefVar_decl_list(s->val.eList.var_decl_list);
  symbolsXrefVar_type(s->val.eList.var_type);
}
void symbolsXrefVar_decl_list_Var (sVar_decl_list *s)
{
  symbolsXrefVar_type(s->val.eVar.var_type);
}
/*-------------------------------------------------------------------*/
void symbolsXrefVar_type (sVar_type *s)
{
  symbolsXrefType(s->val.eVar.type, s->val.eVar.id, s->linienr);
}
/*-------------------------------------------------------------------*/
void symbolsXrefBody (sBody *s)
{
  symbolsXrefDecl_list(s->val.eBody.decl_list);
}
/*-------------------------------------------------------------------*/
void symbolsXrefDecl_list (sDecl_list *s)
{
  switch(s->kind)
```

```
  {
    case kDecl_list_List:  symbolsXrefDecl_list_List (s);break;
    case kDecl_list_Empty: symbolsXrefDecl_list_Empty(s);break;
  }
}
void symbolsXrefDecl_list_List (sDecl_list *s)
{
  symbolsXrefDecl_list(s->val.eList.decl_list);
  symbolsXrefDeclaration(s->val.eList.declaration);
}
void symbolsXrefDecl_list_Empty (sDecl_list *s)
{
}
/*-----------------------------------------------------------------*/
void symbolsXrefDeclaration (sDeclaration *s)
{
  switch(s->kind)
  {
    case kDeclaration_Type:     symbolsXrefDeclaration_Type     (s);break;
    case kDeclaration_Function: symbolsXrefDeclaration_Function(s);break;
    case kDeclaration_Var:      symbolsXrefDeclaration_Var      (s);break;
  }
}
void symbolsXrefDeclaration_Type (sDeclaration *s)
{
  symbolsXrefType(s->val.eType.type, s->val.eType.id, s->linienr);
}
void symbolsXrefDeclaration_Function (sDeclaration *s)
{
  symbolsXrefFunction(s->val.eFunction.function);
}
void symbolsXrefDeclaration_Var (sDeclaration *s)
{
  symbolsXrefVar_decl_list(s->val.eVar.var_decl_list);
}
/*-----------------------------------------------------------------*/



/*-------------------------------------------------------------------
Klargør fil til udskrift af symbol Xref
-------------------------------------------------------------------*/
void openSymbolXrefFile(){
    symbolXrefFile = fopen(SYMBOL_XREF_FILENAME,"w");
    if (symbolXrefFile==NULL){
        fprintf(msgFile,
          "alvorligFejl ved open af symbolXrefFile <%s>\n",
          SYMBOL_XREF_FILENAME);
        exit(1);
    }
}
/*-------------------------------------------------------------------
Udskriver eet symbol
-------------------------------------------------------------------*/
void printSymbolXrefOneSYMBOL_Xref (XREF *xref) {
    if (xref->next != NULL){
        printSymbolXrefOneSYMBOL_Xref (xref->next);
        fprintf(symbolXrefFile,", ");
    }
    fprintf(symbolXrefFile,"%d", xref->linienr);
}

void printSymbolXref_sType (sType *s, int level) {
```

```
  if (s == NULL){
    fprintf(symbolXrefFile," ikke defineret");
    return;
  }
  switch(s->kind)
  {
    case kType_Id:
        if (s->val.eId.type == NULL)
          fprintf(symbolXrefFile," %s =>", s->val.eId.id);
        else
          fprintf(symbolXrefFile," %s<%d> =>",
            s->val.eId.id, s->val.eId.type->linienr);
        if (level < 4) /* sikring mod cyklisk */
          printSymbolXref_sType (s->slutType,level+1);
        break;
    case kType_Int: fprintf(symbolXrefFile," int");
        break;
    case kType_Bool: fprintf(symbolXrefFile," bool");
        break;
    case kType_Array:
        fprintf(symbolXrefFile," array of");
        printSymbolXref_sType (s->val.eArray.type,level+1);
        break;
    case kType_Record: fprintf(symbolXrefFile," record<%d>",s->linienr);
        break;
    default:fprintf(symbolXrefFile," ????");
  }
  fflush(symbolXrefFile);
}

void printSymbolXrefOneSYMBOL_SymbolType (SYMBOL *s) {
  struct sDeclaration *sdecl;
  struct sVar_type *svar;
  struct sHead *shead;
  switch(s->kind)
  {
    case symType: fprintf(symbolXrefFile," \ttype =");
        sdecl = s->astNode;
        printSymbolXref_sType(sdecl->val.eType.type,0);
        break;
    case symHead: fprintf(symbolXrefFile," \tfunction af typen:");
        shead = s->astNode;
        printSymbolXref_sType(shead->val.eHead.type,0);
        break;
    case symVar: fprintf(symbolXrefFile," \tvariabel af typen:");
        svar = s->astNode;
        printSymbolXref_sType(svar->val.eVar.type,0);
        break;
  }
    fflush(symbolXrefFile);
}

void printSymbolXrefOneSYMBOL (SYMBOL *s) {
  XREF *xref = s->xref;
  fprintf(symbolXrefFile,"%s\t<%d> ", s->name, s->linienr);
  printSymbolXrefOneSYMBOL_SymbolType(s);
  if  (xref != NULL)
  {
    fprintf(symbolXrefFile,"   ref: %d", s->antalXref);
    if (s->antalXref > 0) {
      fprintf(symbolXrefFile,"\t<");
      printSymbolXrefOneSYMBOL_Xref (xref);
      fprintf(symbolXrefFile,">");
```

```
    }
  }
  fprintf(symbolXrefFile,"\n\n");
}
/*---------------------------------------------------------------------
Udskriver en enkelt symboltabel på Dumpfilen
---------------------------------------------------------------------*/
void printSymbolXrefOneSymbolTable(SymbolTable *t, char *name, int linienr) {
    int i;
    SYMBOL *s;
  if (t == NULL) return;
  fprintf(symbolXrefFile,"-------------------------------------------\n\n");
  fprintf(symbolXrefFile,
    "SymbolTable for %s:<%d>  tabelnr:%d scope-level:%d - start\n\n",
              name,linienr,t->symboltabelnr,t->level);
    for (i=0;i<HashSize;i++){
        if (t->table[i] != NULL) {
            s = t->table[i];
            do  {
                printSymbolXrefOneSYMBOL (s);
                s = s->next;
            } while (s != NULL);
        }
    }
    fflush(symbolXrefFile);
}
```

Ressource fasen

Funktioner til beregning af feltstørrelser og offset samt parametre

tonyCodeOffset.c

```
/* ---------------------------------------------------------------------
  Realisering af funktioner til beregning af offset og længder
  på typer og variable
  Programmør: Bjørk Busch
  Historik: 2005.05
  Historik: 2005.05.10  kosmetik i source
---------------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include "bjbu_std.h"
#include "symbol.h"
#include "tonyAbstractAsm.h"
#include "tonyAST.h"

extern FILE *msgFile;


void opencodeOffsetTxt();
void codeOffsetProgram (sProgram *s);
void codeOffsetFunction (sFunction *s);
void codeOffsetHead (sHead *s);
void codeOffsetTail (sTail *s);
/*---------------------------------------------------------------------*/
void codeOffsetType (sType *s);
void codeOffsetType_Id (sType *s);
void codeOffsetType_Int (sType *s);
void codeOffsetType_Bool (sType *s);
void codeOffsetType_Array (sType *s);
void codeOffsetType_Record (sType *s);
/*---------------------------------------------------------------------*/
void codeOffsetPar_decl_list (sPar_decl_list *s);
void codeOffsetPar_decl_list_List (sPar_decl_list *s);
void codeOffsetPar_decl_list_Empty (sPar_decl_list *s);
/*---------------------------------------------------------------------*/
void codeOffsetVar_decl_list (sVar_decl_list *s);
void codeOffsetVar_decl_list_List (sVar_decl_list *s);
void codeOffsetVar_decl_list_Var (sVar_decl_list *s);
/*---------------------------------------------------------------------*/
void codeOffsetVar_type (sVar_type *s);
/*---------------------------------------------------------------------*/
void codeOffsetBody (sBody *s);
/*---------------------------------------------------------------------*/
void codeOffsetDecl_list (sDecl_list *s);
void codeOffsetDecl_list_List (sDecl_list *s);
void codeOffsetDecl_list_Empty (sDecl_list *s);
/*---------------------------------------------------------------------*/
void codeOffsetDeclaration (sDeclaration *s);
void codeOffsetDeclaration_Type (sDeclaration *s);
void codeOffsetDeclaration_Function (sDeclaration *s);
void codeOffsetDeclaration_Var (sDeclaration *s);
/*---------------------------------------------------------------------*/

/* til styring af offset beregninger */
int codeOffset_ScopeLevel = 0;
int max_codeOffset_ScopeLevel = 0;
int codeOffset_Offset = 0;
```

```
int labelnr = 0;      /* for unique id */


void codeOffsetProgram (sProgram *s)
{
  fprintf(msgFile, "TONY offset beregninger start\n");
  codeOffsetBody(s->val.eProgram.body);
  s->maxScopeLevel = max_codeOffset_ScopeLevel;
  fprintf(msgFile, "TONY offset beregninger slut\n");
}
void codeOffsetFunction (sFunction *s)
{
  int save_codeOffset_ScopeLevel = codeOffset_ScopeLevel;
  int save_codeOffset_Offset = codeOffset_Offset;
  codeOffset_ScopeLevel = save_codeOffset_ScopeLevel+1;
  if (max_codeOffset_ScopeLevel < codeOffset_ScopeLevel)
    max_codeOffset_ScopeLevel = codeOffset_ScopeLevel;

  codeOffset_Offset = 0;
  codeOffsetHead(s->val.eFunction.head);

  codeOffset_Offset = 0;
  codeOffsetBody(s->val.eFunction.body);

  codeOffsetTail(s->val.eFunction.tail);

  codeOffset_Offset = save_codeOffset_Offset;
  codeOffset_ScopeLevel = save_codeOffset_ScopeLevel;
}
void codeOffsetHead (sHead *s)
{
  ++labelnr;
  s->labelnr = labelnr;
  s->labelEntry = asMakeLabel(s->val.eHead.id,s->labelnr,s->linienr,"");
  s->labelExit= asMakeLabel2(s->val.eHead.id,s->labelnr,"end",s->linienr,"");
  codeOffsetPar_decl_list(s->val.eHead.par_decl_list);
  codeOffsetType(s->val.eHead.type);
}
void codeOffsetTail (sTail *s)
{
}
/*-------------------------------------------------------------------*/
void codeOffsetType (sType *s)
{
    switch(s->kind)
  {
    case kType_Id:    codeOffsetType_Id (s);break;
    case kType_Int:   codeOffsetType_Int  (s);break;
    case kType_Bool:  codeOffsetType_Bool (s);break;
    case kType_Array: codeOffsetType_Array(s);break;
    case kType_Record:  codeOffsetType_Record(s);break;
  }
}

void codeOffsetType_Id (sType *s)
{
  s->dataLength = s->slutType->dataLength;
  if (s->dataLength==0)  /* selv reference */
    s->dataLength = 4;
}
void codeOffsetType_Int (sType *s)
{
  s->dataLength = 4;  /* afsæt 4 byte til int */
```

```
}
void codeOffsetType_Bool (sType *s)
{
  s->dataLength = 4;  /* afsæt 4 byte til bool */
}
void codeOffsetType_Array (sType *s)
{
  codeOffsetType(s->val.eArray.type);
  s->val.eArray.elmLength = s->val.eArray.type->dataLength;
  s->dataLength = 4;  /* afsæt 4 byte til adressen på array */
}
void codeOffsetType_Record (sType *s)
{
  int save_codeOffset_ScopeLevel = codeOffset_ScopeLevel;
  int save_codeOffset_Offset = codeOffset_Offset;
  codeOffset_ScopeLevel = -1;/* -1 betyder reccord og alt andet er level */
  codeOffset_Offset = 0;
  codeOffsetVar_decl_list(s->val.eRecord.var_decl_list);
  s->val.eRecord.elmLength = codeOffset_Offset;
  codeOffset_Offset = save_codeOffset_Offset;
  codeOffset_ScopeLevel = save_codeOffset_ScopeLevel;
  s->dataLength = 4;  /* afsæt 4 byte  til adressen på record */
}
/*-----------------------------------------------------------------*/
void codeOffsetPar_decl_list (sPar_decl_list *s)
{
  switch(s->kind)
  {
    case kPar_decl_list_List:  codeOffsetPar_decl_list_List  (s);break;
    case kPar_decl_list_Empty: codeOffsetPar_decl_list_Empty (s);break;
  }
  s->dataLength = codeOffset_Offset;
}
void codeOffsetPar_decl_list_List (sPar_decl_list *s)
{
  codeOffsetVar_decl_list(s->val.eList.var_decl_list);
}
void codeOffsetPar_decl_list_Empty (sPar_decl_list *s)
{
}
/*-----------------------------------------------------------------*/
void codeOffsetVar_decl_list (sVar_decl_list *s)
{
    switch(s->kind)
  {
    case kVar_decl_list_List: codeOffsetVar_decl_list_List  (s);break;
    case kVar_decl_list_Var:  codeOffsetVar_decl_list_Var   (s);break;
  }
}
void codeOffsetVar_decl_list_List (sVar_decl_list *s)
{
  codeOffsetVar_decl_list(s->val.eList.var_decl_list);
  codeOffsetVar_type(s->val.eList.var_type);
}
void codeOffsetVar_decl_list_Var (sVar_decl_list *s)
{
  codeOffsetVar_type(s->val.eVar.var_type);
}
/*-----------------------------------------------------------------*/
void codeOffsetVar_type (sVar_type *s)
{
  ++labelnr;
  s->labelnr = labelnr;
```

```
  codeOffsetType(s->val.eVar.type);
  s->dataOffset = codeOffset_Offset;
  s->scopeLevel = codeOffset_ScopeLevel;
   /* opskriv offset adresse */
  codeOffset_Offset = codeOffset_Offset + s->val.eVar.type->dataLength;
}
/*------------------------------------------------------------------*/
void codeOffsetBody (sBody *s)
{
  codeOffsetDecl_list(s->val.eBody.decl_list);
  s->dataLength = codeOffset_Offset;
}
/*------------------------------------------------------------------*/
void codeOffsetDecl_list (sDecl_list *s)
{
  switch(s->kind)
  {
    case kDecl_list_List:  codeOffsetDecl_list_List  (s);break;
    case kDecl_list_Empty: codeOffsetDecl_list_Empty (s);break;
  }
}
void codeOffsetDecl_list_List (sDecl_list *s)
{
  codeOffsetDecl_list(s->val.eList.decl_list);
  codeOffsetDeclaration(s->val.eList.declaration);
}
void codeOffsetDecl_list_Empty (sDecl_list *s)
{
}
/*------------------------------------------------------------------*/
void codeOffsetDeclaration (sDeclaration *s)
{
  switch(s->kind)
  {
    case kDeclaration_Type:     codeOffsetDeclaration_Type      (s);break;
    case kDeclaration_Function: codeOffsetDeclaration_Function  (s);break;
    case kDeclaration_Var:      codeOffsetDeclaration_Var       (s);break;
  }
}
void codeOffsetDeclaration_Type (sDeclaration *s)
{
  if (s->val.eType.aktiv)
    codeOffsetType(s->val.eType.type);
}
void codeOffsetDeclaration_Function (sDeclaration *s)
{
  codeOffsetFunction(s->val.eFunction.function);
}
void codeOffsetDeclaration_Var (sDeclaration *s)
{
  codeOffsetVar_decl_list(s->val.eVar.var_decl_list);
}
/*------------------------------------------------------------------*/
```

Direkte kodegennerering

Funktioner til kodegennering direkte fra AST

tonyCodeAsm.c

```
/* ------------------------------------------------------------------
  Realisering af funktioner til kodegennerering
  Programmør: Bjørk Busch
  Historik: 2005.04.
  Historik: 2005.05.10  kosmetik i source
------------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include "bjbu_std.h"
#include "symbol.h"
#include "tonyAbstractAsm.h"
#include "tonyAST.h"

#define codeAsm_FILENAME "Asm_tonyCode_direct.s"
FILE *codeAsmFile;
extern FILE *msgFile;

extern int tonyHeapSize;          /* Størrelse på heap */
/* 1=rækkefølge stigende adresser, -1 giver faldende */
int codeAsmOffsetFortegn_parameter = 1;
int codeAsmOffsetFortegn_lokale = 1;
int codeAsmOffsetFortegn_record = 1;
bool indexcheck = false;

void opencodeAsmTxt();
void codeAsmProgram (sProgram *s);
void codeAsmFunction (sFunction *s);
/*-----------------------------------------------------------------*/
void codeAsmType (sType *s);
void codeAsmType_Id (sType *s);
void codeAsmType_Int (sType *s);
void codeAsmType_Bool (sType *s);
void codeAsmType_Array (sType *s);
void codeAsmType_Record (sType *s);
/*-----------------------------------------------------------------*/
void codeAsmVar_decl_list (sVar_decl_list *s);
void codeAsmVar_decl_list_List (sVar_decl_list *s);
void codeAsmVar_decl_list_Var (sVar_decl_list *s);
/*-----------------------------------------------------------------*/
void codeAsmVar_type (sVar_type *s);
/*-----------------------------------------------------------------*/
void codeAsmDataDecl_list (sDecl_list *s);
void codeAsmDataDecl_list_List (sDecl_list *s);
void codeAsmFunctionDecl_list (sDecl_list *s);
void codeAsmFunctionDecl_list_List (sDecl_list *s);
/*-----------------------------------------------------------------*/
void codeAsmDataDeclaration (sDeclaration *s);
void codeAsmFunctionDeclaration (sDeclaration *s);
void codeAsmDeclaration_Type (sDeclaration *s);
void codeAsmDeclaration_Function (sDeclaration *s);
void codeAsmDeclaration_Var (sDeclaration *s);
/*-----------------------------------------------------------------*/
void codeAsmStatement_list (sStatement_list *s);
void codeAsmStatement_list_List (sStatement_list *s);
void codeAsmStatement_list_Statement (sStatement_list *s);
/*-----------------------------------------------------------------*/
```

```
void codeAsmStatement (sStatement *s);
void codeAsmStatement_Return (sStatement *s);
void codeAsmStatement_Write (sStatement *s);
void codeAsmStatement_NewLength (sStatement *s);
void codeAsmStatement_New (sStatement *s);
void codeAsmStatement_Assign (sStatement *s);
void codeAsmStatement_IfElse (sStatement *s);
void codeAsmStatement_If (sStatement *s);
void codeAsmStatement_While (sStatement *s);
void codeAsmStatement_Compound (sStatement *s);
void codeAsmStatement_ForTo (sStatement *s);
/*-------------------------------------------------------------*/
void codeAsmVariable (sVariable *s, int operant);
void codeAsmVariable_Id (sVariable *s, int operant);
void codeAsmVariable_Indexed (sVariable *s, int operant);
void codeAsmVariable_Struct (sVariable *s, int operant);
void codeAsmVariableAdresse (sVariable *s, int operant);
void codeAsmVariableAdresse_Id (sVariable *s, int operant);
void codeAsmVariableAdresse_Indexed (sVariable *s, int operant);
void codeAsmVariableAdresse_Struct (sVariable *s, int operant);
/*-------------------------------------------------------------*/
void codeAsmExpression (sExpression *s);
void codeAsmExpressionRecursiv (sExpression *s, int operant);
void codeAsmExpression_Dyadisk (sExpression *s, char const *operation,
                                int operant);
void codeAsmExpression_Term (sExpression *s, int operant);
/*-------------------------------------------------------------*/
void codeAsmTerm (sTerm *s, int operant);
void codeAsmTerm_Variable (sTerm *s, int operant);
void codeAsmTerm_Call (sTerm *s, int operant);
void codeAsmTerm_Parantes (sTerm *s, int operant);
void codeAsmTerm_Not (sTerm *s, int operant);
void codeAsmTerm_Numeric (sTerm *s, int operant);
void codeAsmTerm_Num (sTerm *s, int operant);
void codeAsmTerm_True (sTerm *s, int operant);
void codeAsmTerm_False (sTerm *s, int operant);
void codeAsmTerm_Null (sTerm *s, int operant);
/*-------------------------------------------------------------*/
void codeAsmAct_list (sAct_list *s);
void codeAsmAct_list_List (sAct_list *s);
void codeAsmAct_list_Empty (sAct_list *s);
/*-------------------------------------------------------------*/
void codeAsmExp_list (sExp_list *s);
void codeAsmExp_list_List (sExp_list *s);
void codeAsmExp_list_Expression (sExp_list *s);


extern sType *getSlutType(sType *type);




int codeAsm_ScopeLevel = 0;
int codeAsmOffsetFortegn = 1;
int codeAsmOffset = 0;
char codeEndlabel[80];
int codeJmplabelnr = 0;



char *codeOperantreg(int operant)
{
```

```
  if (operant == 1) return "%eax";
  else              return "%ebx";
}

int codeFunctionNr = 0;

void opencodeAsm()
{
/*  codeAsmFile = stdout;*/
  codeAsmFile = fopen(codeAsm_FILENAME,"w");
  if (codeAsmFile==NULL){
    fprintf(msgFile, "alvorligFejl ved open af xmlfile <%s>\n",
      codeAsm_FILENAME);
    exit(1);
  }
}

void codeAsmProgram (sProgram *s)
{
  int i = 0;
  char *fname = "_main";
  opencodeAsm();
  fprintf(msgFile, "TONY kodegennerering udskrives i TXT-format til <%s>\n",
    codeAsm_FILENAME);

  ++codeFunctionNr;
  fprintf(codeAsmFile,"# Oversat fra TONY\n# Compiler skrevet af %s\n\n",
      "Bjørk Boye Busch - bjbu@tietgen.dk");
  fprintf(codeAsmFile,".data\n");
  fprintf(codeAsmFile,"_int_format:\n");
  fprintf(codeAsmFile,".ascii\t\"%%d\\n\\0\"\n");
  fprintf(codeAsmFile,"_zerodivide_format:\n");
  fprintf(codeAsmFile,
    ".ascii\t\"runtime fejl: nul-division i linie %%d\\n\\0\"\n");
  fprintf(codeAsmFile,"_outofmemory_format:\n");
  fprintf(codeAsmFile,".ascii\t\"runtime fejl: out of memory i \
linie %%d - tonyHeapSize(%d)\\n\\0\"\n",tonyHeapSize);
  fprintf(codeAsmFile,"_nonpositivelements_format:\n");
  fprintf(codeAsmFile,".ascii\t\"runtime fejl: arrayallokering med <= 0 \
elementer i linie %%d\\n\\0\"\n");
  fprintf(codeAsmFile,"_indexunderflow_format:\n");
  fprintf(codeAsmFile,
    ".ascii\t\"runtime fejl: index underloeb i linie %%d\\n\\0\"\n");
  fprintf(codeAsmFile,"_indexoverflow_format:\n");
  fprintf(codeAsmFile,
    ".ascii\t\"runtime fejl: index overloeb i linie %%d\\n\\0\"\n");

  fprintf(codeAsmFile,".align\t4\n");
  for (i=0;i<=s->maxScopeLevel;++i)
    fprintf(codeAsmFile,
      "_scopeadr_%d:\n\t.long -1\t#adr. til lokale data på scopelevel:%d\n",
      i,i);

  fprintf(codeAsmFile,"_myheapfreeadr:\n");
  fprintf(codeAsmFile,
    "\t.long\t_myheapspace\t#adresse på ledig heap-hukommelse\n");
  fprintf(codeAsmFile,"_myheapspace:\n");
  fprintf(codeAsmFile,"\t.space\t%d\t#heap-space\n",tonyHeapSize);
  fprintf(codeAsmFile,"_aftermyheapspace:\n");
  fprintf(codeAsmFile,
    "\t.long\t0\t#4 byte ekstra da længde gemmes før test\n");

  fprintf(codeAsmFile,"\n");
```

```
  fprintf(codeAsmFile,".text\n\n");
  fprintf(codeAsmFile,".globl _main\t\t#entry for windows\n");
  fprintf(codeAsmFile,".globl main\t\t#entry for linux\n");

  fprintf(codeAsmFile,"\n#Data adresser defineret i dette scope\n");

  codeAsmOffsetFortegn = codeAsmOffsetFortegn_lokale;
  if (codeAsmOffsetFortegn == 1)
    codeAsmOffset = - s->val.eProgram.body->dataLength;
  else
    codeAsmOffset = 0;

  codeAsmDataDecl_list(s->val.eProgram.body->val.eBody.decl_list);

  fprintf(codeAsmFile,"\n");
  fprintf(codeAsmFile,"_main:\t\t#entry for windows\n");
  fprintf(codeAsmFile,"main:\t\t#entry for linux\n");
  sprintf(codeEndlabel,"main_end");
    fprintf(codeAsmFile,"\tpushl\t%%ebp\n");
  fprintf(codeAsmFile,"\tmovl\t%%esp,%%ebp\n");
  fprintf(codeAsmFile,
    "\tmovl\t%%ebp,_scopeadr_%d\t#gem adr. på data til senere scopes\n",
    codeAsm_ScopeLevel);
  fprintf(codeAsmFile,
    "\tsubl\t$%d,%%esp\t\t\t\t# reserver memory for lokale data\n",
    s->val.eProgram.body->dataLength);

  fprintf(codeAsmFile,"\n");
  codeAsmStatement_list(s->val.eProgram.body->val.eBody.statement_list);

  fprintf(codeAsmFile,"\n\tmovl\t$0,%%eax\t#return værdi 0\n");
  fprintf(codeAsmFile,"%s:\n",codeEndlabel);
  fprintf(codeAsmFile,"\tmovl\t%%ebp,%%esp\n");
  fprintf(codeAsmFile,"\tpopl\t%%ebp\n");
  fprintf(codeAsmFile,"\tret\n");
  fprintf(codeAsmFile,"\n");
  fprintf(codeAsmFile,"#-------------------------------------------------\n");



  fprintf(codeAsmFile,"_indexunderflow:\n");
  fprintf(codeAsmFile,"\tpushl\t%%eax\n");
  fprintf(codeAsmFile,"\tpushl\t$_indexunderflow_format\n");
  fprintf(codeAsmFile,"\tcall\tprintf\n");
  fprintf(codeAsmFile,"\taddl\t$8,%%esp\t# Ryd op paa stakken\n");
  fprintf(codeAsmFile,"\n\tmovl\t$2,%%eax\t#return værdi 2\n");
  fprintf(codeAsmFile,"\tjmp\t_exit\t#afbryd program\n");

  fprintf(codeAsmFile,"_indexoverflow:\n");
  fprintf(codeAsmFile,"\tpushl\t%%eax\n");
  fprintf(codeAsmFile,"\tpushl\t$_indexoverflow_format\n");
  fprintf(codeAsmFile,"\tcall\tprintf\n");
  fprintf(codeAsmFile,"\taddl\t$8,%%esp\t# Ryd op paa stakken\n");
  fprintf(codeAsmFile,"\n\tmovl\t$2,%%eax\t#return værdi 2\n");
  fprintf(codeAsmFile,"\tjmp\t_exit\t#afbryd program\n");

  fprintf(codeAsmFile,"_zerodivide:\n");
  fprintf(codeAsmFile,"\tpushl\t%%eax\n");
  fprintf(codeAsmFile,"\tpushl\t$_zerodivide_format\n");
  fprintf(codeAsmFile,"\tcall\tprintf\n");
  fprintf(codeAsmFile,"\taddl\t$8,%%esp\t# Ryd op paa stakken\n");
  fprintf(codeAsmFile,"\n\tmovl\t$3,%%eax\t#return værdi 3\n");
  fprintf(codeAsmFile,"\tjmp\t_exit\t#afbryd program\n");
```

```
  fprintf(codeAsmFile,"_nonpositivelements:\n");
  fprintf(codeAsmFile,"\tpushl\t%%eax\n");
  fprintf(codeAsmFile,"\tpushl\t$_nonpositivelements_format\n");
  fprintf(codeAsmFile,"\tcall\tprintf\n");
  fprintf(codeAsmFile,"\taddl\t$8,%%esp\t# Ryd op paa stakken\n");
  fprintf(codeAsmFile,"\n\tmovl\t$4,%%eax\t#return værdi 4\n");
  fprintf(codeAsmFile,"\tjmp\t_exit\t#afbryd program\n");

  fprintf(codeAsmFile,"_outofmemory:\n");
  fprintf(codeAsmFile,"\tpushl\t%%eax\n");
  fprintf(codeAsmFile,"\tpushl\t$_outofmemory_format\n");
  fprintf(codeAsmFile,"\tcall\tprintf\n");
  fprintf(codeAsmFile,"\taddl\t$8,%%esp\t# Ryd op paa stakken\n");
  fprintf(codeAsmFile,"\n\tmovl\t$6,%%eax\t#return værdi 6\n");
  fprintf(codeAsmFile,"\tjmp\t_exit\t#afbryd program\n");

  fprintf(codeAsmFile,"_exit:\t#afbryd program\n");
  fprintf(codeAsmFile,
    "\tmovl\t_scopeadr_%d,%%ebp\t#restore adr. på data main-scope\n",
    codeAsm_ScopeLevel);
  fprintf(codeAsmFile,"\tmovl\t%%ebp,%%esp\n");
  fprintf(codeAsmFile,"\tpopl\t%%ebp\n");
  fprintf(codeAsmFile,"\tret\n");
  fprintf(codeAsmFile,"#-------------------------------------------------\n");

  /* næste funktion */
  codeAsmFunctionDecl_list(s->val.eProgram.body->val.eBody.decl_list);

  fclose(codeAsmFile);
}
void codeAsmFunction (sFunction *s)
{
  int save_codeAsm_ScopeLevel = codeAsm_ScopeLevel;
  ++codeFunctionNr;
  codeAsm_ScopeLevel = save_codeAsm_ScopeLevel + 1;

  fprintf(codeAsmFile,".globl %s_%d\n",s->val.eFunction.head->val.eHead.id,
    s->val.eFunction.head->labelnr);
  if (s->val.eFunction.head->val.eHead.par_decl_list->kind
      == kPar_decl_list_List) {

    codeAsmOffsetFortegn = codeAsmOffsetFortegn_parameter;
    if (codeAsmOffsetFortegn==1)
      codeAsmOffset = 8;
    else
      codeAsmOffset = 8
        + s->val.eFunction.head->val.eHead.par_decl_list->dataLength;

    fprintf(codeAsmFile,"\n#Parameter adresser i dette scope\n");
    codeAsmVar_decl_list(
    s->val.eFunction.head->val.eHead.par_decl_list->val.eList.var_decl_list);
  }
  fprintf(codeAsmFile,"\n#Data adresser defineret i dette scope\n");

  codeAsmOffsetFortegn = codeAsmOffsetFortegn_lokale;
  if (codeAsmOffsetFortegn == 1)
    codeAsmOffset = - s->val.eFunction.body->dataLength;
  else
    codeAsmOffset = 0;

  codeAsmDataDecl_list(s->val.eFunction.body->val.eBody.decl_list);
```

```
  fprintf(codeAsmFile,"\n%s_%d:\n",s->val.eFunction.head->val.eHead.id,
    s->val.eFunction.head->labelnr);
  sprintf(codeEndlabel,"%s_%d_end",s->val.eFunction.head->val.eHead.id,
    s->val.eFunction.head->labelnr);
  fprintf(codeAsmFile,"\tpushl\t%%ebp\n");
  fprintf(codeAsmFile,"\tmovl\t%%esp,%%ebp\n");
  fprintf(codeAsmFile,
    "\tmovl\t%%ebp,_scopeadr_%d\t#gem adr. på data til senere scopes\n",
    codeAsm_ScopeLevel);
  fprintf(codeAsmFile,
    "\tsubl\t$%d,%%esp\t\t\t\t# reserver memory for lokale data\n",
    s->val.eFunction.body->dataLength);

  fprintf(codeAsmFile,"\tpushl\t%%ebx\n");
  fprintf(codeAsmFile,"\tpushl\t%%esi\n");
  fprintf(codeAsmFile,"\tpushl\t%%edi\n");

  fprintf(codeAsmFile,"\n");
  codeAsmStatement_list(s->val.eFunction.body->val.eBody.statement_list);

  fprintf(codeAsmFile,"\n%s:\n",codeEndlabel);
  fprintf(codeAsmFile,"\tpopl\t%%edi\n");
  fprintf(codeAsmFile,"\tpopl\t%%esi\n");
  fprintf(codeAsmFile,"\tpopl\t%%ebx\n");
  fprintf(codeAsmFile,"\tmovl\t%%ebp,%%esp\n");
  fprintf(codeAsmFile,"\tpopl\t%%ebp\n");
  fprintf(codeAsmFile,"\tret\n");
  fprintf(codeAsmFile,"\n");
  fprintf(codeAsmFile,"#-----------------------------------------------\n");

  /* næste funktion */
  codeAsmFunctionDecl_list(s->val.eFunction.body->val.eBody.decl_list);

  codeAsm_ScopeLevel = save_codeAsm_ScopeLevel;
}
/*-----------------------------------------------------------------*/
void codeAsmType (sType *s)
{
  switch(s->kind)
  {
    case kType_Id:     codeAsmType_Id    (s);break;
    case kType_Int:    codeAsmType_Int   (s);break;
    case kType_Bool:   codeAsmType_Bool  (s);break;
    case kType_Array:  codeAsmType_Array (s);break;
    case kType_Record: codeAsmType_Record(s);break;
  }
}

void codeAsmType_Id (sType *s)
{
}
void codeAsmType_Int (sType *s)
{
}
void codeAsmType_Bool (sType *s)
{
}
void codeAsmType_Array (sType *s)
{
    codeAsmType(s->val.eArray.type);
}
void codeAsmType_Record (sType *s)
{
```

```
  int save_codeAsm_ScopeLevel = codeAsm_ScopeLevel;
  int save_codeAsmOffset = codeAsmOffset;
  int save_codeAsmOffsetFortegn = codeAsmOffsetFortegn;

  codeAsmOffsetFortegn = codeAsmOffsetFortegn_record;
  if (codeAsmOffsetFortegn == 1)  codeAsmOffset = 0;
  else                            codeAsmOffset = s->val.eRecord.elmLength;

  fprintf(codeAsmFile,"\t\t\t\t\t\t\t#record of{\n");
  codeAsmVar_decl_list(s->val.eRecord.var_decl_list);
  fprintf(codeAsmFile,"\t\t\t\t\t\t\t#\t} end of record\n");

  codeAsmOffsetFortegn = save_codeAsmOffsetFortegn;
  codeAsmOffset = save_codeAsmOffset;
  codeAsm_ScopeLevel = save_codeAsm_ScopeLevel;
}
/*-----------------------------------------------------------------*/
void codeAsmVar_decl_list (sVar_decl_list *s)
{
  switch(s->kind)
  {
    case kVar_decl_list_List: codeAsmVar_decl_list_List (s);break;
    case kVar_decl_list_Var:  codeAsmVar_decl_list_Var  (s);break;
  }
}
void codeAsmVar_decl_list_List (sVar_decl_list *s)
{
  codeAsmVar_decl_list(s->val.eList.var_decl_list);
  codeAsmVar_type(s->val.eList.var_type);
}
void codeAsmVar_decl_list_Var (sVar_decl_list *s)
{
  codeAsmVar_type(s->val.eVar.var_type);
}
/*-----------------------------------------------------------------*/
void codeAsmVar_type (sVar_type *s)
{
  int offset;

  if (codeAsmOffsetFortegn == 1)
    offset = codeAsmOffset+(s->dataOffset*codeAsmOffsetFortegn);
  else
    offset = codeAsmOffset+(s->dataOffset*codeAsmOffsetFortegn)
              -s->val.eVar.type->dataLength;
  fprintf(codeAsmFile,".equ \t%s_%d, %d  \t#%s ",s->val.eVar.id, s->labelnr,
      offset,type_Kind_Txt(getSlutType(s->val.eVar.type)));
  if (s->scopeLevel == codeAsm_ScopeLevel)
    fprintf(codeAsmFile,"  lokal adr. på ebp\n");
  else if (s->scopeLevel < 0)
    fprintf(codeAsmFile,"  indirekte adresse\n");
  else
    fprintf(codeAsmFile,"  lokal adr. fra andet scope\n");

  codeAsmType(s->val.eVar.type);
}
/*-----------------------------------------------------------------*/
void codeAsmDataDecl_list (sDecl_list *s)
{
  switch(s->kind)
  {
    case kDecl_list_List:  codeAsmDataDecl_list_List (s);break;
    case kDecl_list_Empty: break;
  }
```

```
}
void codeAsmDataDecl_list_List (sDecl_list *s)
{
  codeAsmDataDecl_list(s->val.eList.decl_list);
  codeAsmDataDeclaration(s->val.eList.declaration);
}
void codeAsmFunctionDecl_list (sDecl_list *s)
{
  switch(s->kind)
  {
    case kDecl_list_List:  codeAsmFunctionDecl_list_List (s);break;
    case kDecl_list_Empty: break;
  }
}
void codeAsmFunctionDecl_list_List (sDecl_list *s)
{
  codeAsmFunctionDecl_list(s->val.eList.decl_list);
  codeAsmFunctionDeclaration(s->val.eList.declaration);
}
/*-----------------------------------------------------------------*/
void codeAsmDataDeclaration (sDeclaration *s)
{
  switch(s->kind)
  {
    case kDeclaration_Type:     codeAsmDeclaration_Type (s);break;
    case kDeclaration_Function: break;
    case kDeclaration_Var:      codeAsmDeclaration_Var  (s);break;
  }
}
void codeAsmFunctionDeclaration (sDeclaration *s)
{
  switch(s->kind)
  {
    case kDeclaration_Type:     break;
    case kDeclaration_Function: codeAsmDeclaration_Function (s);break;
    case kDeclaration_Var:      break;
  }
}
void codeAsmDeclaration_Type (sDeclaration *s)
{
  sType *type;
  if (!s->val.eType.aktiv) return;
  type = s->val.eType.type;
  while (type->kind == kType_Array)
    type = type->val.eArray.type;
  if (type->kind != kType_Record) return;
  fprintf(codeAsmFile,"\t\t\t\t\t\t\t#type %s {\n", s->val.eType.id);
  codeAsmType(s->val.eType.type);
  fprintf(codeAsmFile,"\t\t\t\t\t\t\t# } end type %s\n", s->val.eType.id);
}
void codeAsmDeclaration_Function (sDeclaration *s)
{
  codeAsmFunction(s->val.eFunction.function);
}
void codeAsmDeclaration_Var (sDeclaration *s)
{
  codeAsmVar_decl_list(s->val.eVar.var_decl_list);
}
/*-----------------------------------------------------------------*/
void codeAsmStatement_list (sStatement_list *s)
{
  switch(s->kind)
  {
```

```
    case kStatement_list_List:     codeAsmStatement_list_List     (s);break;
    case kStatement_list_Statement:codeAsmStatement_list_Statement(s);break;
  }
}
void codeAsmStatement_list_List (sStatement_list *s)
{
  codeAsmStatement_list(s->val.eList.statement_list);
  codeAsmStatement(s->val.eList.statement);
}
void codeAsmStatement_list_Statement (sStatement_list *s)
{
  codeAsmStatement(s->val.eList.statement);
}
/*-------------------------------------------------------------*/
void codeAsmStatement (sStatement *s)
{
  if (!s->aktiv) return;
    switch(s->kind)
  {
    case kStatement_Return:   codeAsmStatement_Return (s);break;
    case kStatement_Write:    codeAsmStatement_Write  (s);break;
    case kStatement_NewLength:codeAsmStatement_NewLength(s);break;
    case kStatement_New:      codeAsmStatement_New    (s);break;
    case kStatement_Assign:   codeAsmStatement_Assign (s);break;
    case kStatement_IfElse:   codeAsmStatement_IfElse (s);break;
    case kStatement_If:       codeAsmStatement_If     (s);break;
    case kStatement_While:    codeAsmStatement_While  (s);break;
    case kStatement_Compound: codeAsmStatement_Compound (s);break;
    case kStatement_ForTo:    codeAsmStatement_ForTo  (s);break;
    case kStatement_ForDownto:codeAsmStatement_ForTo  (s);break;
  }
}
void codeAsmStatement_Return (sStatement *s)
{
  fprintf(codeAsmFile,"\n");
  fprintf(codeAsmFile,"# return linie:%d\n",s->linienr);
  codeAsmExpression(s->val.eReturn.expression);
  fprintf(codeAsmFile,
   "\tjmp\t%s\t#return da alle typer fylder (max) 4 byte - resultat i eax\n",
   codeEndlabel);
  fprintf(codeAsmFile,"\n");
}
void codeAsmStatement_Write (sStatement *s)
{
  fprintf(codeAsmFile,"#write linie:%d\n",s->linienr);
  codeAsmExpression(s->val.eWrite.expression);
  fprintf(codeAsmFile,"\tpushl\t%%eax\n");
  fprintf(codeAsmFile,"\tpushl\t$_int_format\n");
  fprintf(codeAsmFile,"\tcall\tprintf\n");
  fprintf(codeAsmFile,"\taddl\t$8,%%esp\t# Ryd op paa stakken\n");
  fprintf(codeAsmFile,"\n");
}
void codeAsmStatement_NewLength (sStatement *s)
{
  sType *arrayType = getSlutType(s->val.eNewLength.variable->type);
  fprintf(codeAsmFile,"\n");
  fprintf(codeAsmFile,"#new length linie:%d\n",s->linienr);
  codeAsmExpression(s->val.eNewLength.expression);

  ++codeJmplabelnr;
  fprintf(codeAsmFile,"\torl\t%%eax,%%eax\t#test positivt antal elem.\n");
  fprintf(codeAsmFile,"\tjg\t_retheap1_%d\n",codeJmplabelnr);
  fprintf(codeAsmFile,"\tmovl\t$%d,%%eax\t#linienr på fejl\n",s->linienr);
```

```
  fprintf(codeAsmFile,"\tjmp\t_nonpositivelements\n");
  fprintf(codeAsmFile,"_retheap1_%d:\n",codeJmplabelnr);

  fprintf(codeAsmFile,"\tpushl\t%%eax\t#Save antal elm\n");

  fprintf(codeAsmFile,"\tmovl\t$%d,%%ebx\t#elementlængde\n",
    arrayType->val.eArray.elmLength);
  fprintf(codeAsmFile,
    "\timull\t%%ebx\t#memory_size = elementer*elementlængde + \n");
  fprintf(codeAsmFile,"\taddl\t$4,%%eax\t#  plads til antals variabel\n");

  fprintf(codeAsmFile,"\tmovl\t_myheapfreeadr,%%esi\n");
  fprintf(codeAsmFile,
    "\tmovl\t%%eax,0(%%esi)\t#gem allokeret memory længde\n");
  fprintf(codeAsmFile,"\taddl\t%%esi,%%eax\t#ny free-adresse\n");
  fprintf(codeAsmFile,"\taddl\t$4,%%eax\t#  beregnes\n");
  fprintf(codeAsmFile,"\tmovl\t%%eax,_myheapfreeadr\t#og gemmes\n");

  ++codeJmplabelnr;
  fprintf(codeAsmFile,
    "\tcmpl\t$_aftermyheapspace,%%eax\t#test for out of memory\n");
  fprintf(codeAsmFile,"\tjng\t_retheap2_%d\n",codeJmplabelnr);
  fprintf(codeAsmFile,"\tmovl\t$%d,%%eax\t#linienr på fejl\n",s->linienr);
  fprintf(codeAsmFile,"\tjmp\t_outofmemory\n");
  fprintf(codeAsmFile,"_retheap2_%d:\n",codeJmplabelnr);

  fprintf(codeAsmFile,
"\tleal\t8(%%esi),%%eax\t#adresse på array: offset -8 size og -4 antal\n");

  fprintf(codeAsmFile,"\tpopl\t%%ebx\t#Hent antal elm\n");
  fprintf(codeAsmFile,
    "\tmovl\t%%ebx,-4(%%eax)\t#Gem antal elm. i array -4\n");

  fprintf(codeAsmFile,"# store i variabel\n");
  codeAsmVariable(s->val.eNewLength.variable, -1);
  fprintf(codeAsmFile,"\n");
}
void codeAsmStatement_New (sStatement *s)
{
  sType *recordType = getSlutType(s->val.eNew.variable->type);
  fprintf(codeAsmFile,"\n");
  fprintf(codeAsmFile,"#new linie:%d\n",s->linienr);

  fprintf(codeAsmFile,"\tmovl\t$%d,%%ebx\t#recordlaengde\n",
    recordType->val.eRecord.elmLength);
  fprintf(codeAsmFile,"\tmovl\t_myheapfreeadr,%%eax\n");
  fprintf(codeAsmFile,
    "\tmovl\t%%ebx,0(%%eax)\t#gem allokeret memory længde\n");
  fprintf(codeAsmFile,"\taddl\t%%ebx,%%eax\t#ny free-adresse\n");
  fprintf(codeAsmFile,"\taddl\t$4,%%eax\t#  beregnes\n");
  fprintf(codeAsmFile,"\tmovl\t%%eax,_myheapfreeadr\t#og gemmes\n");

  ++codeJmplabelnr;
  fprintf(codeAsmFile,
    "\tcmpl\t$_aftermyheapspace,%%eax\t#test for out of memory\n");
  fprintf(codeAsmFile,"\tjng\t_retheap_%d\n",codeJmplabelnr);
  fprintf(codeAsmFile,"\tmovl\t$%d,%%eax\t#linienr på fejl\n",s->linienr);
  fprintf(codeAsmFile,"\tjmp\t_outofmemory\n");
  fprintf(codeAsmFile,"_retheap_%d:\n",codeJmplabelnr);

  fprintf(codeAsmFile,"\tsubl\t$%d,%%eax\t#adr = freeadr-recordlaengde\n",
    recordType->val.eRecord.elmLength);
```

```
  codeAsmVariable(s->val.eNew.variable, -1);
  fprintf(codeAsmFile,"\n");
}
void codeAsmStatement_Assign (sStatement *s)
{
  fprintf(codeAsmFile,"\n");
  fprintf(codeAsmFile,"# assign linie:%d\n",s->linienr);
  codeAsmExpression(s->val.eAssign.expression);
  codeAsmVariable(s->val.eAssign.variable,-1);
  fprintf(codeAsmFile,"\n");
}
void codeAsmStatement_IfElse (sStatement *s)
{
  int labelnr = ++codeJmplabelnr;
  fprintf(codeAsmFile,"\n");
  fprintf(codeAsmFile,"# if-then-else linie:%d\n",s->linienr);
  codeAsmExpression(s->val.eIfElse.expression);
  fprintf(codeAsmFile,"\tcmpl\t$1,%%eax\t#if true\n");
  fprintf(codeAsmFile,"\tjne\t_else_%d\n",labelnr);
  codeAsmStatement(s->val.eIfElse.ifStatement);
  fprintf(codeAsmFile,"\tjmp\t_if_%d_end\n",labelnr);
  fprintf(codeAsmFile,"_else_%d:\n",labelnr);
  codeAsmStatement(s->val.eIfElse.elseStatement);
  fprintf(codeAsmFile,"_if_%d_end:\n",labelnr);
  fprintf(codeAsmFile,"\n");
}
void codeAsmStatement_If (sStatement *s)
{
  int labelnr = ++codeJmplabelnr;
  fprintf(codeAsmFile,"\n");
  fprintf(codeAsmFile,"# if-then linie:%d\n",s->linienr);
  codeAsmExpression(s->val.eIf.expression);
  fprintf(codeAsmFile,"\tcmpl\t$1,%%eax\t#if true\n");
  fprintf(codeAsmFile,"\tjne\t_if_%d_end\n",labelnr);
  codeAsmStatement(s->val.eIf.ifStatement);
  fprintf(codeAsmFile,"_if_%d_end:\n",labelnr);
  fprintf(codeAsmFile,"\n");
}
void codeAsmStatement_While (sStatement *s)
{
  int labelnr = ++codeJmplabelnr;
  fprintf(codeAsmFile,"\n");
  fprintf(codeAsmFile,"# while linie:%d\n",s->linienr);
  fprintf(codeAsmFile,"_while_%d:\n",labelnr);
  codeAsmExpression(s->val.eWhile.expression);
  fprintf(codeAsmFile,"\tcmpl\t$1,%%eax\t#while true\n");
  fprintf(codeAsmFile,"\tjne\t_while_%d_end\n",labelnr);
  codeAsmStatement(s->val.eWhile.statement);
  fprintf(codeAsmFile,"\tjmp\t_while_%d\n",labelnr);
  fprintf(codeAsmFile,"_while_%d_end:\n",labelnr);
  fprintf(codeAsmFile,"\n");
}
void codeAsmStatement_Compound (sStatement *s)
{
  codeAsmStatement_list(s->val.eCompound.statement_list);
}
void codeAsmStatement_ForTo (sStatement *s)
{
  int labelnr = ++codeJmplabelnr;
  fprintf(codeAsmFile,"\n");
  fprintf(codeAsmFile,"# for to linie:%d\n",s->linienr);
  codeAsmExpression(s->val.eForTo.expressionInit);
  codeAsmVariable(s->val.eForTo.variable,-1);
```

```
  /* beregn to-exp i ebx */
  codeAsmExpressionRecursiv(s->val.eForTo.expressionTo,2);
  fprintf(codeAsmFile,
    "\tpushl\t%%ebx\t#gem stopvalue på stak til at teste mod\n");
  fprintf(codeAsmFile,"_for_%d_test:\n",labelnr);
  fprintf(codeAsmFile,"\tcmpl\t0(%%esp),%%eax\t#test mod to vaerdi\n");
  if (s->kind == kStatement_ForTo)
    fprintf(codeAsmFile,"\tjg\t_for_%d_end\n",labelnr);
  else
    fprintf(codeAsmFile,"\tjl\t_for_%d_end\n",labelnr);
  fprintf(codeAsmFile,
    "\tpushl\t%%eax\t#gem counter til næste gennemloeb\n");
  codeAsmStatement(s->val.eForTo.statement);
  fprintf(codeAsmFile,
    "\tpopl\t%%eax\t#reetabler og juster counter for næste gennemloeb\n");
  if (s->kind == kStatement_ForTo)
    fprintf(codeAsmFile,"\tincl\t%%eax\t#step +1\n");
  else
    fprintf(codeAsmFile,"\tdecl\t%%eax\t#step -1\n");
  codeAsmVariable(s->val.eForTo.variable,-1); /* save ny vaerdi */
  fprintf(codeAsmFile,"\tjmp\t_for_%d_test\n",labelnr);
  fprintf(codeAsmFile,"_for_%d_end:\n",labelnr);
  fprintf(codeAsmFile,"\taddl\t$4,%%esp\t# Fjern stopværdi fra stakken\n");
  fprintf(codeAsmFile,"\n");
}
/*-----------------------------------------------------------------*/
void codeAsmVariable (sVariable *s, int operant)
{
  switch(s->kind)
  {
    case kVariable_Id:      codeAsmVariable_Id     (s, operant);break;
    case kVariable_Indexed: codeAsmVariable_Indexed(s, operant);break;
    case kVariable_Struct:  codeAsmVariable_Struct (s, operant);break;
  }
}
void codeAsmVariable_Id (sVariable *s, int operant)
{
  char *reg;

  if (s->val.eId.var_type != NULL)
    fprintf(codeAsmFile,"#v%d_%s#%s\n",s->val.eId.var_type->linienr,
      type_Kind_Txt(s->type),s->val.eId.id);
  else
    fprintf(codeAsmFile,"#v_%s#%s\n",type_Kind_Txt(s->type),s->val.eId.id);

  if (operant == 1)   reg = "%eax";
  else if (operant ==2) reg = "%ebx";

  if (s->val.eId.var_type->scopeLevel == codeAsm_ScopeLevel) {
    if (operant >= 1)
      fprintf(codeAsmFile,
        "\tmovl\t%s_%d(%%ebp),%s\t#Load lokal variabel\n",
        s->val.eId.var_type->val.eVar.id,
        s->val.eId.var_type->labelnr, reg );
    else
      fprintf(codeAsmFile,
        "\tmovl\t%%eax,%s_%d(%%ebp)\t#Store lokal variabel\n",
        s->val.eId.var_type->val.eVar.id, s->val.eId.var_type->labelnr );
  }
  else if (s->val.eId.var_type->scopeLevel >= 0) {
    if (operant >= 1) {
      fprintf(codeAsmFile,"\tpush\t%%esi\n");
      fprintf(codeAsmFile,
```

```
        "\tmovl\t_scopeadr_%d,%%esi\t#Load adr. på data fra andet scope\n",
        s->val.eId.var_type->scopeLevel);
      fprintf(codeAsmFile,
        "\tmovl\t%s_%d(%%esi),%s\t#Load lokal variabel i andet scope\n",
        s->val.eId.var_type->val.eVar.id,
        s->val.eId.var_type->labelnr, reg );
      fprintf(codeAsmFile,"\tpop\t%%esi\n");
    }
    else {
      fprintf(codeAsmFile,"\tpush\t%%edi\n");
      fprintf(codeAsmFile,
        "\tmovl\t_scopeadr_%d,%%edi\t#Load adr. på data fra andet scope\n",
        s->val.eId.var_type->scopeLevel);
      fprintf(codeAsmFile,
        "\tmovl\t%%eax,%s_%d(%%edi)\t#Store variabel i andet scope\n",
        s->val.eId.var_type->val.eVar.id, s->val.eId.var_type->labelnr );
      fprintf(codeAsmFile,"\tpop\t%%edi\n");
    }
  }
  else
    fprintf(codeAsmFile,"???????? alvorligFejl i adresseberegninger\n");
}
void codeAsmVariable_Indexed (sVariable *s, int operant)
{
  if (operant < 1) {
    fprintf(codeAsmFile,"# indexed store\n");
    fprintf(codeAsmFile,"\tpushl\t%%eax\t# gem værdi\n");
  }
  else {
    fprintf(codeAsmFile,"# indexed load\n");
    if (operant == 2)
      fprintf(codeAsmFile,"\tpushl\t%%eax\t# gem værdi\n");
  }

  /* beregn base adresse */
  codeAsmVariableAdresse(s, operant); /* skal levere base-adresse i eax */

  fprintf(codeAsmFile,"# indexed load/store\n");
  if (operant == 1){    /* load */
    fprintf(codeAsmFile,"\tmovl\t%%eax,%%esi\n");
    fprintf(codeAsmFile,"\tmovl\t0(%%esi),%%eax\n");
  }
  else if (operant == 2){   /* load */
    fprintf(codeAsmFile,"\tmovl\t%%eax,%%esi\n");
    fprintf(codeAsmFile,"\tpopl\t%%eax\t# hent værdi igen\n");
    fprintf(codeAsmFile,"\tmovl\t0(%%esi),%%ebx\n");
  }
  else{          /* store */
    fprintf(codeAsmFile,"\tmovl\t%%eax,%%edi\n");
    fprintf(codeAsmFile,"\tpopl\t%%eax\t# hent værdi igen\n");
    fprintf(codeAsmFile,"\tmovl\t%%eax,0(%%edi)\n");
  }
}
void codeAsmVariable_Struct (sVariable *s, int operant)
{
  if (operant < 1) {
    fprintf(codeAsmFile,"# struct store\n");
    fprintf(codeAsmFile,"\tpushl\t%%eax\t# gem værdi\n");
  }
  else {
    fprintf(codeAsmFile,"# struct load\n");
    if (operant == 2)
      fprintf(codeAsmFile,"\tpushl\t%%eax\t# gem værdi\n");
```

```
  }

  /* beregn base adresse */
  /* skal levere base-adresse i eax */
  codeAsmVariableAdresse(s->val.eStruct.variable, operant);

  fprintf(codeAsmFile,"# struct load/store\n");
  if (operant == 1){    /* load */
    fprintf(codeAsmFile,
      "\tmovl\t%s_%d(%%eax),%%eax\t#base adr. på struktur-variabel i eax\n",
      s->val.eId.var_type->val.eVar.id, s->val.eId.var_type->labelnr);
  }
  else if (operant == 2){   /* load */
    fprintf(codeAsmFile,
      "\tmovl\t%s_%d(%%eax),%%ebx\t# base adr. på struktur-variabel i eax\n",
      s->val.eId.var_type->val.eVar.id, s->val.eId.var_type->labelnr);
    fprintf(codeAsmFile,"\tpopl\t%%eax\t# hent 1.op værdi igen\n");
  }
  else{         /* store */
    fprintf(codeAsmFile,"\tpopl\t%%edx\t# hent værdi igen\n");
    fprintf(codeAsmFile,
      "\tmovl\t%%edx,%s_%d(%%eax)\t# base adr. på struktur-variabel i edi\n",
      s->val.eId.var_type->val.eVar.id, s->val.eId.var_type->labelnr);
  }
}

void codeAsmVariableAdresse (sVariable *s, int operant)
{
  switch(s->kind)
  {
    case kVariable_Id:      codeAsmVariableAdresse_Id     (s,operant);break;
    case kVariable_Indexed: codeAsmVariableAdresse_Indexed(s,operant);break;
    case kVariable_Struct:  codeAsmVariableAdresse_Struct (s,operant);break;
  }
}
void codeAsmVariableAdresse_Id (sVariable *s, int operant)
{
  fprintf(codeAsmFile,"# hent adr.\n");
  if (s->val.eId.var_type->scopeLevel == codeAsm_ScopeLevel)
    fprintf(codeAsmFile,
      "\tmovl\t%s_%d(%%ebp),%%eax\t# variabel adresse i eax\n",
      s->val.eId.var_type->val.eVar.id, s->val.eId.var_type->labelnr);
  else if (s->val.eId.var_type->scopeLevel  >= 0) {
    fprintf(codeAsmFile,"\tpush\t%%esi\n");
    fprintf(codeAsmFile,
    "\tmovl\t_scopeadr_%d,%%esi\t#hent adr. på lokaldata fra andet scope\n",
      s->val.eId.var_type->scopeLevel);
    fprintf(codeAsmFile,
"\tmovl\t%s_%d(%%esi),%%eax\t# variabel adressse fra andet scope i eax\n",
      s->val.eId.var_type->val.eVar.id, s->val.eId.var_type->labelnr);
    fprintf(codeAsmFile,"\tpop\t%%esi\n");
  }
}
void codeAsmVariableAdresse_Indexed (sVariable *s, int operant)
{
  fprintf(codeAsmFile,"# calc indexed adr.\n");
  fprintf(codeAsmFile,"\tpushl\t%%ebx\t# save\n");
  fprintf(codeAsmFile,"\tpushl\t%%edx\t# save\n");
  codeAsmExpressionRecursiv(s->val.eIndexed.expression, 1);

  fprintf(codeAsmFile,"\tmovl\t$%d,%%ebx\t# hent elm. længde ebx\n",
    s->val.eIndexed.variable->type->slutType->val.eArray.elmLength);
  fprintf(codeAsmFile,"\timul\t%%ebx\t# eax = elm. offset\n");
```

```
  fprintf(codeAsmFile,"\tpushl\t%%eax\t# save elm. offset\n");
  /* skal levere adresse i eax */
  codeAsmVariableAdresse(s->val.eIndexed.variable, 0);
  if (s->val.eIndexed.variable->kind == kVariable_Indexed)
    fprintf(codeAsmFile,
      "\tmovl\t0(%%eax),%%eax\t# adr paa array elm. adresse i eax\n");
  fprintf(codeAsmFile,"\tpopl\t%%ebx\t# restore elm. offset\n");

  if (indexcheck)  {
    ++codeJmplabelnr;
    fprintf(codeAsmFile,"\torl\t%%ebx,%%ebx\t#test for index underflow\n");
    fprintf(codeAsmFile,"\tjge\t_index1_%d\n",codeJmplabelnr);
    fprintf(codeAsmFile,"\tmovl\t$%d,%%eax\t#linienr på fejl\n",s->linienr);
    fprintf(codeAsmFile,"\tjmp\t_indexunderflow\n");
    fprintf(codeAsmFile,"_index1_%d:\n",codeJmplabelnr);
    fprintf(codeAsmFile,
      "\tcmpl\t%%ebx,-8(%%eax)\t#test for index overflow = offset fejl\n");
    fprintf(codeAsmFile,"\tjnle\t_index2_%d\n",codeJmplabelnr);
    fprintf(codeAsmFile,"\tmovl\t$%d,%%eax\t#linienr på fejl\n",s->linienr);
    fprintf(codeAsmFile,"\tjmp\t_indexoverflow\n");
    fprintf(codeAsmFile,"_index2_%d:\n",codeJmplabelnr);
  }
  if (operant==0)
    fprintf(codeAsmFile,
      "\tmovl\t0(%%ebx,%%eax),%%eax\t# elm. adresse i eax\n");
  else
    fprintf(codeAsmFile,
      "\tleal\t0(%%ebx,%%eax),%%eax\t# adr paa array elm. adresse i eax\n");

  fprintf(codeAsmFile,"\tpopl\t%%edx\t# reetabler\n");
  fprintf(codeAsmFile,"\tpopl\t%%ebx\t# reetabler\n");
}
void codeAsmVariableAdresse_Struct (sVariable *s, int operant)
{
  fprintf(codeAsmFile,"# calc struct adr.\n");
  codeAsmVariableAdresse(s->val.eStruct.variable, 0);
  fprintf(codeAsmFile,"\tmovl\t%s_%d(%%eax),%%eax\t# struct adresse i eax\n",
    s->val.eId.var_type->val.eVar.id, s->val.eId.var_type->labelnr);
}
/*-----------------------------------------------------------------*/
void codeAsmExpression (sExpression *s)
{
  codeAsmExpressionRecursiv (s, 1);
}
void codeAsmExpressionRecursiv (sExpression *s, int operant)
{
    switch(s->kind)
  {
    case kExp_Eq:   codeAsmExpression_Dyadisk (s,"==",operant);break;
    case kExp_Neq:  codeAsmExpression_Dyadisk (s,"!=",operant);break;
    case kExp_Lt:   codeAsmExpression_Dyadisk (s,"<",operant); break;
    case kExp_Gt:   codeAsmExpression_Dyadisk (s,">",operant); break;
    case kExp_Lteq: codeAsmExpression_Dyadisk (s,"<=",operant);break;
    case kExp_Gteq: codeAsmExpression_Dyadisk (s,">=",operant);break;
    case kExp_Add:  codeAsmExpression_Dyadisk (s,"+",operant); break;
    case kExp_Sub:  codeAsmExpression_Dyadisk (s,"-",operant); break;
    case kExp_Mul:  codeAsmExpression_Dyadisk (s,"*",operant); break;
    case kExp_Div:  codeAsmExpression_Dyadisk (s,"/",operant); break;
    case kExp_Or:   codeAsmExpression_Dyadisk (s,"||",operant);break;
    case kExp_And:  codeAsmExpression_Dyadisk (s,"&&",operant);break;
    case kExp_Term: codeAsmExpression_Term    (s,operant);     break;
```

```
  }
}
void codeAsmDyadiskInstruction (sExpression *s)
{
  char *jmp;
  switch(s->kind)
  {
    case kExp_Eq:
    case kExp_Neq:
    case kExp_Lt:
    case kExp_Gt:
    case kExp_Lteq:
    case kExp_Gteq:
      ++codeJmplabelnr;
      fprintf(codeAsmFile,"\tcmpl %%ebx,%%eax\t#sammenlignings exp\n");
      switch(s->kind)
      {
        case kExp_Eq:   jmp = "je";   break;
        case kExp_Neq:  jmp = "jne";  break;
        case kExp_Lt:   jmp = "jl";   break;
        case kExp_Gt:   jmp = "jg";   break;
        case kExp_Lteq: jmp = "jle";  break;
        case kExp_Gteq: jmp = "jge";  break;
      }
      fprintf(codeAsmFile,"\t%s\t_exp_%d_true\n",jmp,codeJmplabelnr);
      fprintf(codeAsmFile,"\tmovl\t$0,%%eax\t#exp false\n");
      fprintf(codeAsmFile,"\tjmp\t_exp_%d_end\n",codeJmplabelnr);
      fprintf(codeAsmFile,"_exp_%d_true:\n",codeJmplabelnr);
      fprintf(codeAsmFile,"\tmovl\t$1,%%eax\t#exp true\n");
      fprintf(codeAsmFile,"_exp_%d_end:\n",codeJmplabelnr);
      break;
    case kExp_Add:
      fprintf(codeAsmFile,"\taddl\t%%ebx,%%eax\t#+\n");
      break;
    case kExp_Sub:
      fprintf(codeAsmFile,"\tsubl\t%%ebx,%%eax\t#-\n");
      break;
    case kExp_Mul:
      fprintf(codeAsmFile,"\timull\t%%ebx,%%eax\t#*\n");
      break;
    case kExp_Div:
      ++codeJmplabelnr;
      fprintf(codeAsmFile,"\torl\t%%ebx,%%ebx\t# test for zero-divide\n");
      fprintf(codeAsmFile,"\tjnz\t_divide_%d\n",codeJmplabelnr);
      fprintf(codeAsmFile,
        "\tmovl\t$%d,%%eax\t#linienr på fejl\n",s->linienr);
      fprintf(codeAsmFile,"\tjmp\t_zerodivide\n");
      fprintf(codeAsmFile,"_divide_%d:\n",codeJmplabelnr);
      fprintf(codeAsmFile,"\tcltd\n");
      fprintf(codeAsmFile,"\tidivl\t%%ebx\t#/\n");
      break;
    case kExp_Or:
      fprintf(codeAsmFile,"\torl\t%%ebx,%%eax\t#||\n");
      break;
    case kExp_And:
      fprintf(codeAsmFile,"\tandl\t%%ebx,%%eax\t#&&\n");
  }
}
void codeAsmExpression_Dyadisk (sExpression *s,
                                char const *operation,
                                int operant)
{
  if (operant!=1){
```

```
    fprintf(codeAsmFile,"\tpushl\t%%eax\t# gem 1. operant eax\n");
  }
  /* beregn 1. op i eax */
  codeAsmExpressionRecursiv(s->val.eLeftRight.expressionLeft,1);
  /* beregn 2. op i ebx */
  codeAsmExpressionRecursiv(s->val.eLeftRight.expressionRight,2);
  codeAsmDyadiskInstruction (s);
  if (operant!=1){
    fprintf(codeAsmFile,"\tmovl\t%%eax,%%ebx\t# aflever i 2. operant ebx\n");
    fprintf(codeAsmFile,"\tpopl\t%%eax\t# reetabeler 1. operant eax\n");
  }
}
void codeAsmExpression_Term (sExpression *s, int operant)
{
  codeAsmTerm(s->val.eTerm.term, operant);
}
/*-------------------------------------------------------------------*/
void codeAsmTerm (sTerm *s, int operant)
{
  switch(s->kind)
  {
    case kTerm_Variable:  codeAsmTerm_Variable(s,operant);break;
    case kTerm_Call:      codeAsmTerm_Call  (s,operant);break;
    case kTerm_Parantes:  codeAsmTerm_Parantes(s,operant);break;
    case kTerm_Not:       codeAsmTerm_Not   (s,operant);break;
    case kTerm_Numeric:   codeAsmTerm_Numeric (s,operant);break;
    case kTerm_Num:       codeAsmTerm_Num   (s,operant);break;
    case kTerm_True:      codeAsmTerm_True  (s,operant);break;
    case kTerm_False:     codeAsmTerm_False (s,operant);break;
    case kTerm_Null:      codeAsmTerm_Null  (s,operant);break;
  }
}
void codeAsmTerm_Variable (sTerm *s, int operant)
{
  codeAsmVariable(s->val.eVariable.variable, operant);
}
void codeAsmTerm_Call (sTerm *s, int operant)
{
  int parmBlokSize = s->val.eCall.head->val.eHead.par_decl_list->dataLength;

  if (operant!=1){
    fprintf(codeAsmFile,"\tpushl\t%%eax\t# gem 1. operant eax\n");
  }
  if (parmBlokSize != 0) {
    fprintf(codeAsmFile,"\tpushl\t%%edi\t\t\t\t# save edi\n");
    fprintf(codeAsmFile,
      "\tsubl\t$%d,%%esp\t\t\t\t# reserver memory for parameterblok\n",
      parmBlokSize);
    fprintf(codeAsmFile,
      "\tmovl\t%%esp,%%edi\t\t\t\t# etabler base-reference til parametre\n");
    fprintf(codeAsmFile,"\tsubl\t$8,%%edi\t\t\t\t# \n");
  }
  codeAsmAct_list(s->val.eCall.act_list);
  fprintf(codeAsmFile,"\tcall\t%s_%d\n",s->val.eCall.head->val.eHead.id,
    s->val.eCall.head->labelnr);
  if (parmBlokSize != 0) {
    fprintf(codeAsmFile,
      "\taddl\t$%d,%%esp\t\t\t\t# frigiv parameterblok\n",parmBlokSize);
    fprintf(codeAsmFile,"\tpopl\t%%edi\t\t\t\t# restore edi\n");
  }
  if (operant!=1){
    fprintf(codeAsmFile,"\tmovl\t%%eax,%%ebx\t# aflever i 2. operant ebx\n");
    fprintf(codeAsmFile,"\tpopl\t%%eax\t# reetabeler 1. operant eax\n");
```

```
  }
}
void codeAsmTerm_Parantes (sTerm *s, int operant)
{
  codeAsmExpressionRecursiv(s->val.eParantes.expression, operant);
}
void codeAsmTerm_Not (sTerm *s, int operant)
{
  codeAsmTerm(s->val.eNot.term, operant);
  fprintf(codeAsmFile,
    "\txorl\t$1,%s\t#not %d.operant\n",codeOperantreg(operant),operant);
}
void codeAsmTerm_Numeric (sTerm *s, int operant)
{
  int labelnr;
  fprintf(codeAsmFile,"#numeric\n");
  codeAsmExpressionRecursiv(s->val.eNumeric.expression, operant);
  if (s->val.eNumeric.expression->type->kind == kType_Int) {
    labelnr = ++codeJmplabelnr;
    fprintf(codeAsmFile,
      "\tcmpl\t$0,%s\t#numeric %d.operant\n",
      codeOperantreg(operant),operant);
    fprintf(codeAsmFile,"\tjge\t_positiv_%d\n",labelnr);
    fprintf(codeAsmFile,
      "\tnegl\t%s\t#vend fortegn %d.operant\n",
      codeOperantreg(operant),operant);
    fprintf(codeAsmFile,"_positiv_%d:\n",labelnr);

  }
  else if (s->val.eNumeric.expression->type->slutType->kind == kType_Array) {
      fprintf(codeAsmFile,
        "\tmovl\t-4(%s),%s\t#array laengde til %d.operant\n",
        codeOperantreg(operant),codeOperantreg(operant),operant);
  }
}
void codeAsmTerm_Num (sTerm *s, int operant)
{
  fprintf(codeAsmFile,"\tmovl\t$%d,%s\t#num %d.operant\n",
    s->val.eNum.num,codeOperantreg(operant),operant);
}
void codeAsmTerm_True (sTerm *s, int operant)
{
  fprintf(codeAsmFile,"\tmovl\t$1,%s\t#true %d.operant\n",
    codeOperantreg(operant),operant);
}
void codeAsmTerm_False (sTerm *s, int operant)
{
  fprintf(codeAsmFile,"\txorl\t%s,%s\t#false %d.operant\n",
    codeOperantreg(operant),codeOperantreg(operant),operant);
}
void codeAsmTerm_Null (sTerm *s, int operant)
{
  fprintf(codeAsmFile,"\tmovl\t$-1,%s\t#null %d.operant\n",
    codeOperantreg(operant),operant);
}
/*------------------------------------------------------------------*/
void codeAsmAct_list (sAct_list *s)
{
  switch(s->kind)
  {
    case kAct_list_List:  codeAsmAct_list_List  (s);break;
    case kAct_list_Empty: codeAsmAct_list_Empty (s);break;
  }
```

```
}
void codeAsmAct_list_List (sAct_list *s)
{
  codeAsmExp_list(s->val.eList.exp_list);
}
void codeAsmAct_list_Empty (sAct_list *s)
{
}
/*------------------------------------------------------------------*/
void codeAsmExp_list (sExp_list *s)
{
  switch(s->kind)
  {
    case kExp_list_List:       codeAsmExp_list_List      (s);break;
    case kExp_list_Expression: codeAsmExp_list_Expression(s);break;
  }
}
void codeAsmExp_list_List (sExp_list *s)
{
  codeAsmExp_list(s->val.eList.exp_list);
  codeAsmExpressionRecursiv(s->val.eList.expression, 1);
  fprintf(codeAsmFile,"\tmovl\t%%eax,%s_%d(%%edi)\t# store parameter\n",
    s->val.eList.var_type->val.eVar.id, s->val.eList.var_type->labelnr);
}
void codeAsmExp_list_Expression (sExp_list *s)
{
  codeAsmExpressionRecursiv(s->val.eExpression.expression, 1);
  fprintf(codeAsmFile,"\tmovl\t%%eax,%s_%d(%%edi)\t# store parameter\n",
    s->val.eExpression.var_type->val.eVar.id,
    s->val.eExpression.var_type->labelnr);
}
```

Code fasen

Strukturer og funktioner til opbygning af abstract assembler kode

tonyAbstractAsm.h

```
/* ------------------------------------------------------------------------
  Definition af strukturer for TONY abstract assember code
  Programmør: Bjørk Busch
  Historik:  2005.04.30
  Historik: 2005.05.10  kosmetik i source
------------------------------------------------------------------------*/
typedef struct asAdr {
  union {
    struct { int reg; } eReg;
    struct { struct asElm *offset; int base;
        int faktor; int disp; } eMem;   /* explicit angivelser */
    struct { int offset; int base;
        int faktor; int disp; } eMemEx; /* explicit angivelser */
    struct { int imm; } eImm;          /* Immidiate (value)   */
    struct { struct asElm *label;
        } eLabel;   /* memory label med implicit segment ... */
  } val;
  enum {aaReg, aaMem, aaMemEx, aaImm, aaLabelValue, aaLabelAdr} kind;
}asAdr;

typedef struct asElm {
  bool slettet;
  struct asElm *next;
  int linienr;
  char *comment;
  enum {akComment, akDef, akOffset, akLabel,  /* spec. */
  akPush, akPop, akMul, akDiv, akInc, akDec, akNeg, /* 1 op */
  akMove, akLoadAdr, akAdd, akSub, akAnd, akOr, akXor, akCmp, /*2 op*/
  akJump, akCall, akReturn, akCltd} kind;
  union {
      struct { char *def; } eDef;
      struct { char *name; int nr; int offset; } eOffset;
      struct { char *name; int nr; char *suffix; } eLabel;
      struct { struct asAdr *adr; } eInstr1;
      struct { struct asAdr *fAdr; struct asAdr *tAdr; } eInstr2;
      struct { struct asElm *label;
         enum {akJmp, akJe, akJne, akJl, akJg, akJle, akJge,
         akJnl, akJng} jump;
      } eJump;
      struct { struct asElm *label; } eCall;
  } val;
}asElm;

enum {nreg, eax, ebx, ecx, edx, esi, edi, ebp, esp} reg;

asAdr *asMakeAdrReg (int reg);
asAdr *asMakeAdrMem (asElm *offset, int base, int faktor, int disp);
asAdr *asMakeAdrMemEx (int offset, int base, int faktor, int disp);
asAdr *asMakeAdrImm (int imm);
asAdr *asMakeAdrLabelValue (asElm *label);
asAdr *asMakeAdrLabelAdr (asElm *label);

asElm *addAsmElm (asElm *elm);

asElm *asMakeComment (int linienr, char *comment);
asElm *asMakeDef (char *def, int linienr, char *comment);
```

```
asElm *asMakeOffset (char *name, int nr, int offset,
         int linienr, char *comment);
asElm *asMakeLabel (char *name, int nr, int linienr, char *comment);
asElm *asMakeLabel2 (char *name, int nr, char *suffix,
         int linienr, char *comment);
asElm *asMakeInstr1 (int kind, asAdr *adr, int linienr, char *comment);
asElm *asMakeInstr2 (int kind, asAdr *fAdr, asAdr *tAdr,
         int linienr, char *comment);
asElm *asMakeJump (int jump, asElm *label, int linienr, char *comment);
asElm *asMakeCall (asElm *label, int linienr, char *comment);
asElm *asMakeReturn (int linienr, char *comment);
asElm *asMakeCltd (int linienr, char *comment);

void prettyAsmElm (asElm *e);
```

tonyAbstractAsm.c

```
/* ------------------------------------------------------------------
  Realisering af funktioner til kodegennerering
  Programmør: Bjørk Busch
  Historik:  2005.04.30
  Historik: 2005.05.10  kosmetik i source
------------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include "bjbu_std.h"
#include "tonyAbstractAsm.h"

struct asElm *asmListeHead;    /* head på abstract assembler liste */
struct asElm *asmListeTail;    /* tail på abstract assembler liste */

/*-----------------------------------------------------------------
Funktioner for opbygning af Abstract assembler-Strukturer
-----------------------------------------------------------------*/
asAdr *asMakeAdrReg (int reg)
{
  asAdr *a;
  a = NEW(asAdr);
  a->kind = aaReg;
  a->val.eReg.reg = reg;
  return a;
}
asAdr *asMakeAdrMem (asElm *offset, int base, int faktor, int disp)
{
  asAdr *a;
  a = NEW(asAdr);
  a->kind = aaMem;
  a->val.eMem.offset = offset;
  a->val.eMem.base = base;
  a->val.eMem.faktor = faktor;
  a->val.eMem.disp = disp;
  return a;
}
asAdr *asMakeAdrMemEx (int offset, int base, int faktor, int disp)
{
  asAdr *a;
  a = NEW(asAdr);
  a->kind = aaMemEx;
  a->val.eMemEx.offset = offset;
  a->val.eMemEx.base = base;
  a->val.eMemEx.faktor = faktor;
  a->val.eMemEx.disp = disp;
  return a;
```

```
}
asAdr *asMakeAdrImm (int imm)
{
  asAdr *a;
  a = NEW(asAdr);
  a->kind = aaImm;
  a->val.eImm.imm = imm;
  return a;
}
asAdr *asMakeAdrLabelValue (asElm *label)
{
  asAdr *a;
  a = NEW(asAdr);
  a->kind = aaLabelValue;
  a->val.eLabel.label = label;
  return a;
}
asAdr *asMakeAdrLabelAdr (asElm *label)
{
  asAdr *a;
  a = NEW(asAdr);
  a->kind = aaLabelAdr;
  a->val.eLabel.label = label;
  return a;
}


asElm *addAsmElm (asElm *elm)
{
  if (asmListeHead == NULL) {
    asmListeHead = elm;
    asmListeTail = elm;
  }
  else {
    asmListeTail->next = elm;
    asmListeTail = elm;
  }
  return elm;
}


asElm *asMakeComment (int linienr, char *comment)
{
  asElm *e;
  e = NEW(asElm);
  e->slettet = false;
  e->next = NULL;
  e->kind = akComment;
  e->linienr = linienr;
  if (comment == NULL) e->comment = "";
  else e->comment = NewStringCopy(comment);
  return e;
}
asElm *asMakeDef (char *def, int linienr, char *comment)
{
  asElm *e;
  e = NEW(asElm);
  e->slettet = false;
  e->next = NULL;
  e->kind = akDef;
  e->linienr = linienr;
  if (comment == NULL) e->comment = "";
  else e->comment = NewStringCopy(comment);
```

```
  e->val.eDef.def = NewStringCopy(def);
  return e;
}
asElm *asMakeOffset (char *name, int nr, int offset,
           int linienr, char *comment)
{
  asElm *e;
  e = NEW(asElm);
  e->slettet = false;
  e->next = NULL;
  e->kind = akOffset;
  e->linienr = linienr;
  if (comment == NULL) e->comment = "";
  else e->comment = NewStringCopy(comment);
  e->val.eOffset.name = NewStringCopy(name);
  e->val.eOffset.nr = nr;
  e->val.eOffset.offset = offset;
  return e;
}
asElm *asMakeLabel (char *name, int nr, int linienr, char *comment)
{
  asElm *e;
  e = NEW(asElm);
  e->slettet = false;
  e->next = NULL;
  e->kind = akLabel;
  e->linienr = linienr;
  if (comment == NULL) e->comment = "";
  else e->comment = NewStringCopy(comment);
  e->val.eLabel.name = NewStringCopy(name);
  e->val.eLabel.nr = nr;
  e->val.eLabel.suffix = "";
  return e;
}
asElm *asMakeLabel2 (char *name, int nr, char *suffix,
           int linienr, char *comment)
{
  asElm *e;
  e = NEW(asElm);
  e->slettet = false;
  e->next = NULL;
  e->kind = akLabel;
  e->linienr = linienr;
  if (comment == NULL) e->comment = "";
  else e->comment = NewStringCopy(comment);
  e->val.eLabel.name = NewStringCopy(name);
  e->val.eLabel.nr = nr;
  if (suffix == NULL) e->val.eLabel.suffix = "";
  else e->val.eLabel.suffix = NewStringCopy(suffix);
  return e;
}
asElm *asMakeInstr1 (int kind, asAdr *adr, int linienr, char *comment)
{
  asElm *e;
  e = NEW(asElm);
  e->slettet = false;
  e->next = NULL;
  e->kind = kind;
  e->linienr = linienr;
  if (comment == NULL) e->comment = "";
  else e->comment = NewStringCopy(comment);
  e->val.eInstr1.adr = adr;
  return e;
```

```
}
asElm *asMakeInstr2 (int kind, asAdr *fAdr, asAdr *tAdr,
           int linienr, char *comment)
{
  asElm *e;
  e = NEW(asElm);
  e->slettet = false;
  e->next = NULL;
  e->kind = kind;
  e->linienr = linienr;
  if (comment == NULL) e->comment = "";
  else e->comment = NewStringCopy(comment);
  e->val.eInstr2.fAdr = fAdr;
  e->val.eInstr2.tAdr = tAdr;
  return e;
}
asElm *asMakeJump (int jump, asElm *label, int linienr, char *comment)
{
  asElm *e;
  e = NEW(asElm);
  e->slettet = false;
  e->next = NULL;
  e->kind = akJump;
  e->linienr = linienr;
  if (comment == NULL) e->comment = "";
  else e->comment = NewStringCopy(comment);
  e->val.eJump.jump = jump;
  e->val.eJump.label = label;
  return e;
}
asElm *asMakeCall (asElm *label, int linienr, char *comment)
{
  asElm *e;
  e = NEW(asElm);
  e->slettet = false;
  e->next = NULL;
  e->kind = akCall;
  e->linienr = linienr;
  if (comment == NULL) e->comment = "";
  else e->comment = NewStringCopy(comment);
  e->val.eCall.label = label;
  return e;
}
asElm *asMakeReturn (int linienr, char *comment)
{
  asElm *e;
  e = NEW(asElm);
  e->slettet = false;
  e->next = NULL;
  e->kind = akReturn;
  e->linienr = linienr;
  if (comment == NULL) e->comment = "";
  else e->comment = NewStringCopy(comment);
  return e;
}
asElm *asMakeCltd (int linienr, char *comment)
{
  asElm *e;
  e = NEW(asElm);
  e->slettet = false;
  e->next = NULL;
  e->kind = akCltd;
  e->linienr = linienr;
```

```
  if (comment == NULL) e->comment = "";
  else e->comment = NewStringCopy(comment);
  return e;
}
```

Funktioner til kodegennerering fra AST til abstract assembler

tonyAbstractAsmCode.c

```
/* ------------------------------------------------------------------
  Realisering af funktioner til kodegennerering
  Programmør: Bjørk Busch
  Historik:  2005.04.30
  Historik:  2005.05.09
  Historik: 2005.05.10  kosmetik i source
  Historik: 2005.05.14  fejretning i adresser + dumprutiner
------------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include "bjbu_std.h"
#include "symbol.h"
#include "tonyAbstractAsm.h"
#include "tonyAST.h"

extern bool debugWrite;    /* debug facilitet*/
extern bool debugRegister; /* debug facilitet*/
extern bool debugHeap;     /* debug facilitet*/
extern int  debugHeapSize; /* debug facilitet */

extern bool recordArray;

/* head på abstract assembler liste */
struct asElm *asmListeHead = NULL;
/* tail på abstract assembler liste */
struct asElm *asmListeTail = NULL;

extern FILE *msgFile;

/* Størrelse på heap */
extern int tonyHeapSize;
/* styrer rækkefølge af main og funktioner */
extern functionTopdown;
/* 1=rækkefølge stigende adresser, -1 giver faldende */
int abstractAsmOffsetFortegn_parameter = 1;
int abstractAsmOffsetFortegn_lokale = 1;
int abstractAsmOffsetFortegn_record = 1;


void openabstractAsmTxt();
void abstractAsmProgram (sProgram *s);
void abstractAsmFunction (sFunction *s);
/*-------------------------------------------------------------*/
void abstractAsmType (sType *s);
void abstractAsmType_Id (sType *s);
void abstractAsmType_Int (sType *s);
void abstractAsmType_Bool (sType *s);
void abstractAsmType_Array (sType *s);
void abstractAsmType_Record (sType *s);
/*-------------------------------------------------------------*/
void abstractAsmVar_decl_list (sVar_decl_list *s);
void abstractAsmVar_decl_list_List (sVar_decl_list *s);
void abstractAsmVar_decl_list_Var (sVar_decl_list *s);
```

```
/*-------------------------------------------------------------*/
void abstractAsmVar_type (sVar_type *s);
/*-------------------------------------------------------------*/
void abstractAsmDataDecl_list (sDecl_list *s);
void abstractAsmDataDecl_list_List (sDecl_list *s);
void abstractAsmFunctionDecl_list (sDecl_list *s);
void abstractAsmFunctionDecl_list_List (sDecl_list *s);
/*-------------------------------------------------------------*/
void abstractAsmDataDeclaration (sDeclaration *s);
void abstractAsmFunctionDeclaration (sDeclaration *s);
void abstractAsmDeclaration_Type (sDeclaration *s);
void abstractAsmDeclaration_Function (sDeclaration *s);
void abstractAsmDeclaration_Var (sDeclaration *s);
/*-------------------------------------------------------------*/
void abstractAsmStatement_list (sStatement_list *s);
void abstractAsmStatement_list_List (sStatement_list *s);
void abstractAsmStatement_list_Statement (sStatement_list *s);
/*-------------------------------------------------------------*/
void abstractAsmStatement (sStatement *s);
void abstractAsmStatement_Return (sStatement *s);
void abstractAsmStatement_Write (sStatement *s);
void abstractAsmStatement_NewLength (sStatement *s);
void abstractAsmStatement_New (sStatement *s);
void abstractAsmStatement_Assign (sStatement *s);
void abstractAsmStatement_IfElse (sStatement *s);
void abstractAsmStatement_If (sStatement *s);
void abstractAsmStatement_While (sStatement *s);
void abstractAsmStatement_Compound (sStatement *s);
void abstractAsmStatement_ForTo (sStatement *s);
/*-------------------------------------------------------------*/
void abstractAsmVariable (sVariable *s, int operant);
void abstractAsmVariable_Id (sVariable *s, int operant);
void abstractAsmVariable_Indexed (sVariable *s, int operant);
void abstractAsmVariable_Struct (sVariable *s, int operant);
void abstractAsmVariableAdresse (sVariable *s, int operant);
void abstractAsmVariableAdresse_Id (sVariable *s, int operant);
void abstractAsmVariableAdresse_Indexed (sVariable *s, int operant);
void abstractAsmVariableAdresse_Struct (sVariable *s, int operant);
/*-------------------------------------------------------------*/
void abstractAsmExpression (sExpression *s);
void abstractAsmExpressionRecursiv (sExpression *s, int operant);
void abstractAsmExpression_Dyadisk (
  sExpression *s, char const *operation, int operant);
void abstractAsmExpression_Term (sExpression *s, int operant);
/*-------------------------------------------------------------*/
void abstractAsmTerm (sTerm *s, int operant);
void abstractAsmTerm_Variable (sTerm *s, int operant);
void abstractAsmTerm_Call (sTerm *s, int operant);
void abstractAsmTerm_Parantes (sTerm *s, int operant);
void abstractAsmTerm_Not (sTerm *s, int operant);
void abstractAsmTerm_Numeric (sTerm *s, int operant);
void abstractAsmTerm_Num (sTerm *s, int operant);
void abstractAsmTerm_True (sTerm *s, int operant);
void abstractAsmTerm_False (sTerm *s, int operant);
void abstractAsmTerm_Null (sTerm *s, int operant);
/*-------------------------------------------------------------*/
void abstractAsmAct_list (sAct_list *s);
void abstractAsmAct_list_List (sAct_list *s);
void abstractAsmAct_list_Empty (sAct_list *s);
/*-------------------------------------------------------------*/
void abstractAsmExp_list (sExp_list *s);
void abstractAsmExp_list_List (sExp_list *s);
void abstractAsmExp_list_Expression (sExp_list *s);
```

```
extern sType *getSlutType(sType *type);


int abstractAsm_ScopeLevel = 0;
int abstractAsmOffsetFortegn = 1;
int abstractAsmOffset = 0;
char endlabel[80];
char abstractText[80];
int jmplabelnr = 0;


/*---------------------------------------------------------------------
Faste globale labels
---------------------------------------------------------------------*/
asElm *labPrintf;
asElm *labMainStartWin;
asElm *labMainStartLinux;
asElm *labMainEnd;
asElm *labIntFormat;
asElm *labIndexunderflowFormat;
asElm *labIndexoverflowFormat;
asElm *labZerodivideFormat;
asElm *labNonpositivelementsFormat;
asElm *labOutofmemoryFormat;
asElm *labMyheapfreeadr;
asElm *labMyheapspace;
asElm *labAftermyheapspace;
asElm *labIndexunderflow;
asElm *labIndexoverflow;
asElm *labZerodivide;
asElm *labNonpositivelements;
asElm *labOutofmemory;
asElm *labMainExit;

asElm *labScopeAdr[100];  /* max. 100 scopes */

asElm *labFentry;  /* funktions entry */
asElm *labFexit;  /* funktions exit */

asElm *labDumpRegFormat;
asElm *labDumpMemFormat;
asElm *labDumpReg;
asElm *labDumpHeap;
asElm *labDumpNr;

void initLabels ()
{
  int i;
  labPrintf = asMakeLabel("printf",-1,0,"");
  labMainStartWin = asMakeLabel("_main",-1,0,"entry for windows");
  labMainStartLinux = asMakeLabel("main",-1,0,"entry for linux");
  labMainEnd = asMakeLabel("main_end",-1,0,"");
  labIntFormat = asMakeLabel("_int_format",-1,0,"");
  labIndexunderflowFormat = asMakeLabel("_indexunderflow_format",-1,0,"");
  labIndexoverflowFormat = asMakeLabel("_indexoverflow_format",-1,0,"");
  labZerodivideFormat = asMakeLabel("_zerodivide_format",-1,0,"");
  labNonpositivelementsFormat = asMakeLabel("_nonpositivelements_format",
                                            -1,0,"");
  labOutofmemoryFormat = asMakeLabel("_outofmemory_format",-1,0,"");

  labMyheapfreeadr = asMakeLabel("_myheapfreeadr",-1,0,"");
  labMyheapspace = asMakeLabel("_myheapspace",-1,0,"");
```

```
  labAftermyheapspace = asMakeLabel("_aftermyheapspace",-1,0,"");

  labIndexunderflow = asMakeLabel("_indexunderflow",-1,0,"");
  labIndexoverflow =  asMakeLabel("_indexoverflow",-1,0,"");
  labZerodivide = asMakeLabel("_zerodivide",-1,0,"");
  labNonpositivelements = asMakeLabel("_nonpositivelements",-1,0,"");
  labOutofmemory = asMakeLabel("_outofmemory",-1,0,"");
  labMainExit = asMakeLabel("_exit",-1,0,"");
  for (i=0;i<=99;++i) {
    labScopeAdr[i] = asMakeLabel("_scopeadr",i,
                                 0,"frame-adr. til scopelevel");
  }


  labDumpNr = asMakeLabel("_dumpNr",-1,0,"");
  labDumpRegFormat = asMakeLabel("_dumpReg_format",-1,0,"");
  labDumpMemFormat = asMakeLabel("_dumpMem_format",-1,0,"");
  labDumpReg = asMakeLabel("_dumpReg",-1,0,"");
  labDumpHeap = asMakeLabel("_dumpHeap",-1,0,"");
}

/*---------------------------------------------------------------------------*/
/* debug rutiner                                                             */
/*---------------------------------------------------------------------------*/
void makeDebugFormatLabels ()
{
  addAsmElm(labDumpNr);
  addAsmElm(asMakeDef(".long 0",0,""));
  addAsmElm(labDumpRegFormat);
  addAsmElm(asMakeDef(
  ".ascii  \"%d a%d b%d c%d d%d si%d di%d sp%d\\n\\0\"",0,""));
  addAsmElm(labDumpMemFormat);
  addAsmElm(asMakeDef(
  ".ascii  \"%d :%d\\n\\0\"",0,""));
}
void makeDebugFunction ()
{
  asElm *labHeapDumpStart;
  asElm *labHeapDumpEnd;
  int i;
  ++jmplabelnr;
  labHeapDumpStart = asMakeLabel2("_heapdump",jmplabelnr,"start",0,"");;
  labHeapDumpEnd   = asMakeLabel2("_heapdump",jmplabelnr,"end",0,"");;


  addAsmElm(asMakeComment(0,"debug rutiner"));
  addAsmElm(labDumpReg);
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(esp),0,""));
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(edi),0,""));
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(esi),0,""));
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(edx),0,""));
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(ecx),0,""));
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(ebx),0,""));
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(eax),0,""));
  addAsmElm(asMakeInstr1(akPush,asMakeAdrLabelValue(labDumpNr),0,""));
  addAsmElm(asMakeInstr1(akPush,asMakeAdrLabelAdr(labDumpRegFormat),0,""));
  addAsmElm(asMakeCall(labPrintf,0,""));
  addAsmElm(asMakeInstr2(akAdd,asMakeAdrImm(8),asMakeAdrReg(esp),
            0,"Fjern format igen"));
  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(eax),0,""));
  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(ebx),0,""));
  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(ecx),0,""));
  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(edx),0,""));
```

```
  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(esi),0,""));
  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(edi),0,""));
  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(esp),0,""));
  addAsmElm(asMakeReturn(0,""));
  addAsmElm(asMakeComment(0,""));

  addAsmElm(labDumpHeap);
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(esp),0,""));
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(edi),0,""));
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(esi),0,""));
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(edx),0,""));
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(ecx),0,""));
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(ebx),0,""));
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(eax),0,""));

  addAsmElm(asMakeInstr1(akPush,asMakeAdrLabelValue(labMyheapfreeadr),0,""));
  addAsmElm(asMakeInstr1(akPush,asMakeAdrLabelAdr(labMyheapfreeadr),0,""));
  addAsmElm(asMakeInstr1(akPush,asMakeAdrLabelAdr(labDumpMemFormat),0,""));
  addAsmElm(asMakeCall(labPrintf,0,""));
  addAsmElm(asMakeInstr2(akAdd,asMakeAdrImm(12),asMakeAdrReg(esp),
            0,"Fjern format igen"));

  addAsmElm(asMakeInstr2(akLoadAdr,
            asMakeAdrLabelValue(labMyheapspace),asMakeAdrReg(eax),
            0,""));
  addAsmElm(asMakeInstr2(akMove,asMakeAdrImm(debugHeapSize),
            asMakeAdrReg(ecx),0,""));
  addAsmElm(labHeapDumpStart);
  addAsmElm(asMakeInstr1(akDec,asMakeAdrReg(ecx),
            0,"Flere ?"));
  addAsmElm(asMakeJump(akJl,labHeapDumpEnd,0,"slut -> hop"));
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(ecx),0,""));

  addAsmElm(asMakeInstr2(akMove,
            asMakeAdrMemEx(0,eax,0,nreg),
            asMakeAdrReg(edx),
            0,"adr"));
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(edx),0,""));
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(eax),0,""));
  addAsmElm(asMakeInstr1(akPush,asMakeAdrLabelAdr(labDumpMemFormat),0,""));
  addAsmElm(asMakeCall(labPrintf,0,""));
  addAsmElm(asMakeInstr2(akAdd,asMakeAdrImm(4),asMakeAdrReg(esp),
            0,"Fjern format igen"));
  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(eax),0,""));
  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(edx),0,""));
  addAsmElm(asMakeInstr2(akAdd,asMakeAdrImm(4),asMakeAdrReg(eax),0,"adr"));

  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(ecx),0,""));
  addAsmElm(asMakeJump(akJmp,labHeapDumpStart,0,"igen"));
  addAsmElm(labHeapDumpEnd);

  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(eax),0,""));
  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(ebx),0,""));
  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(ecx),0,""));
  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(edx),0,""));
  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(esi),0,""));
  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(edi),0,""));
  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(esp),0,""));
  addAsmElm(asMakeReturn(0,""));
  addAsmElm(asMakeComment(0,""));
}
void dumpRegister (int nr)
{
```

```
  if (!debugWrite) return;
  if (!debugRegister) return;
  addAsmElm(asMakeInstr2(akMove,asMakeAdrImm(nr),
            asMakeAdrLabelValue(labDumpNr),
            0,"Gem nr til dumprutine"));
  addAsmElm(asMakeCall(labDumpReg,0,""));
}
void dumpHeap ()
{
  if (!debugWrite) return;
  if (!debugHeap) return;
  addAsmElm(asMakeCall(labDumpHeap,0,""));
}
/*---------------------------------------------------------------------------*/


void abstractAsmProgram (sProgram *s)
{
  int i = 0;
  char *fname = "_main";

  fprintf(msgFile, "TONY kodegennerering af abstract assembler start\r\n");

  asmListeHead = NULL;       /* head på abstract assembler liste */
  asmListeTail = NULL;       /* tail på abstract assembler liste */
  initLabels();        /* initier std. labels */

  addAsmElm(asMakeDef(".data",0,""));
  addAsmElm(asMakeComment(0,""));

if (debugWrite) makeDebugFormatLabels ();

  addAsmElm(labIntFormat);
  addAsmElm(asMakeDef(".ascii \"%d\\n\\0\"",0,""));
  addAsmElm(labZerodivideFormat);
  addAsmElm(asMakeDef(
    ".ascii  \"runtime fejl: nul-division i linie %d\\n\\0\"",0,""));
  addAsmElm(labOutofmemoryFormat);
  sprintf(abstractText,
".ascii  \"runtime fejl: out of memory i linie %%d - heapsize(%d)\\n\\0\"",
tonyHeapSize);
  addAsmElm(asMakeDef(abstractText,0,""));
  addAsmElm(labNonpositivelementsFormat);
  addAsmElm(asMakeDef(
      ".ascii  \"runtime fejl: array <= 0 elementer i linie %d\\n\\0\"",
      0,""));
  addAsmElm(labIndexunderflowFormat);
  addAsmElm(asMakeDef(
      ".ascii  \"runtime fejl: index underloeb i linie %d\\n\\0\"",0,""));
  addAsmElm(labIndexoverflowFormat);
  addAsmElm(asMakeDef(
      ".ascii  \"runtime fejl: index overloeb i linie %d\\n\\0\"",0,""));
  addAsmElm(asMakeDef(".align  4",0,""));
  for (i=0;i<=s->maxScopeLevel;++i) {
    addAsmElm(labScopeAdr[i]);
    sprintf(abstractText,"adr. til lokale data paa scope-level %d",i);
    addAsmElm(asMakeDef(".long -1",0,abstractText));
  }

  addAsmElm(labMyheapfreeadr);
  addAsmElm(asMakeDef(".long  _myheapspace",0,
            "adresse på ledig heap-hukommelse"));
  addAsmElm(labMyheapspace);
```

```
  sprintf(abstractText,"  .space  %d",tonyHeapSize);
  addAsmElm(asMakeDef(abstractText,0,"heap-space"));
  addAsmElm(labAftermyheapspace);
  addAsmElm(asMakeDef(".long  0",0,
            "4 byte ekstra da længde gemmes før test"));

  addAsmElm(asMakeComment(0,""));
  addAsmElm(asMakeDef(".text",0,""));

if (debugWrite) makeDebugFunction ();

  /* funktionsliste */
  if (!functionTopdown)
    abstractAsmFunctionDecl_list(s->val.eProgram.body->val.eBody.decl_list);

  labFexit = labMainExit;
  addAsmElm(asMakeComment(0,""));
  sprintf(abstractText,".globl %s",labMainStartWin->val.eLabel.name);
  addAsmElm(asMakeDef(abstractText,0,"#entry for windows"));
  sprintf(abstractText,".globl %s",labMainStartLinux->val.eLabel.name);
  addAsmElm(asMakeDef(abstractText,0,"#entry for linux"));

  abstractAsmOffsetFortegn = abstractAsmOffsetFortegn_lokale;
  if (abstractAsmOffsetFortegn == 1)
    abstractAsmOffset = - s->val.eProgram.body->dataLength;
  else
    abstractAsmOffset = 0;

  addAsmElm(asMakeComment(0,""));
  addAsmElm(asMakeComment(0,"Data adresser defineret i dette scope"));
  addAsmElm(asMakeComment(0,""));

  abstractAsmDataDecl_list(s->val.eProgram.body->val.eBody.decl_list);
  addAsmElm(asMakeComment(0,""));

  addAsmElm(labMainStartWin);
  addAsmElm(labMainStartLinux);
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(ebp),0,"save ebp"));
  addAsmElm(asMakeInstr2(akMove,asMakeAdrReg(esp),asMakeAdrReg(ebp),
            0,"set frame ptr"));
  addAsmElm(asMakeInstr2(akMove,asMakeAdrReg(ebp),
            asMakeAdrLabelValue(labScopeAdr[abstractAsm_ScopeLevel]),0,
            "gem frame ptr til andre scopes"));
  addAsmElm(asMakeInstr2(akSub,
            asMakeAdrImm(s->val.eProgram.body->dataLength),
            asMakeAdrReg(esp),0,"reserver memory for lokale data"));
/*
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(ebx),0,""));
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(ecx),0,""));
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(edx),0,""));
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(esi),0,""));
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(edi),0,""));
*/
  addAsmElm(asMakeComment(0,"main insructions start"));

  abstractAsmStatement_list(s->val.eProgram.body->val.eBody.statement_list);

  addAsmElm(asMakeComment(0,"main insructions slut"));
  addAsmElm(asMakeInstr2(akXor,asMakeAdrReg(eax),asMakeAdrReg(eax),
      0,"afslut normal med returværdi 0"));

  addAsmElm(labMainEnd);
```

```
/*
  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(edi),0,""));
  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(esi),0,""));
  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(edx),0,""));
  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(ecx),0,""));
  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(ebx),0,""));
*/
  addAsmElm(asMakeInstr2(akMove,asMakeAdrReg(ebp),asMakeAdrReg(esp),
      0,"restore stak ptr"));
  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(ebp),
      0,"restore caler frame-ptr"));
  addAsmElm(asMakeReturn(0,""));
  addAsmElm(asMakeComment(0,"----------------------------"));

  addAsmElm(labIndexunderflow);
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(eax),0,""));
  addAsmElm(asMakeInstr1(akPush,
      asMakeAdrLabelAdr(labIndexunderflowFormat),0,""));
  addAsmElm(asMakeCall(labPrintf,0,""));
  addAsmElm(asMakeInstr2(akAdd,asMakeAdrImm(8),asMakeAdrReg(esp),
      0,"Ryd op paa stak"));
  addAsmElm(asMakeInstr2(akMove,asMakeAdrImm(2),asMakeAdrReg(eax),
      0,"Returvaerdi 2"));
  addAsmElm(asMakeJump(akJmp,labMainExit,0,""));

  addAsmElm(asMakeComment(0,""));

  addAsmElm(labIndexoverflow);
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(eax),0,""));
  addAsmElm(asMakeInstr1(akPush,
      asMakeAdrLabelAdr(labIndexoverflowFormat),0,""));
  addAsmElm(asMakeCall(labPrintf,0,""));
  addAsmElm(asMakeInstr2(akAdd,asMakeAdrImm(8),asMakeAdrReg(esp),
      0,"Ryd op paa stak"));
  addAsmElm(asMakeInstr2(akMove,asMakeAdrImm(2),asMakeAdrReg(eax),
      0,"Returvaerdi 2"));
  addAsmElm(asMakeJump(akJmp,labMainExit,0,""));

  addAsmElm(asMakeComment(0,""));

  addAsmElm(labZerodivide);
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(eax),0,""));
  addAsmElm(asMakeInstr1(akPush,
      asMakeAdrLabelAdr(labZerodivideFormat),0,""));
  addAsmElm(asMakeCall(labPrintf,0,""));
  addAsmElm(asMakeInstr2(akAdd,asMakeAdrImm(8),asMakeAdrReg(esp),
      0,"Ryd op paa stak"));
  addAsmElm(asMakeInstr2(akMove,asMakeAdrImm(3),asMakeAdrReg(eax),
      0,"Returvaerdi 3"));
  addAsmElm(asMakeJump(akJmp,labMainExit,0,""));

  addAsmElm(asMakeComment(0,""));

  addAsmElm(labNonpositivelements);
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(eax),0,""));
  addAsmElm(asMakeInstr1(akPush,
      asMakeAdrLabelAdr(labNonpositivelementsFormat),0,""));
  addAsmElm(asMakeCall(labPrintf,0,""));
  addAsmElm(asMakeInstr2(akAdd,asMakeAdrImm(8),asMakeAdrReg(esp),
      0,"Ryd op paa stak"));
  addAsmElm(asMakeInstr2(akMove,asMakeAdrImm(4),asMakeAdrReg(eax),
      0,"Returvaerdi 4"));
  addAsmElm(asMakeJump(akJmp,labMainExit,0,""));
```

```
  addAsmElm(asMakeComment(0,""));

  addAsmElm(labOutofmemory);
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(eax),0,""));
  addAsmElm(asMakeInstr1(akPush,
      asMakeAdrLabelAdr(labOutofmemoryFormat),0,""));
  addAsmElm(asMakeCall(labPrintf,0,""));
  addAsmElm(asMakeInstr2(akAdd,asMakeAdrImm(8),asMakeAdrReg(esp),
      0,"Ryd op paa stak"));
  addAsmElm(asMakeInstr2(akMove,asMakeAdrImm(6),asMakeAdrReg(eax),
      0,"Returvaerdi 6"));
  addAsmElm(asMakeJump(akJmp,labMainExit,0,""));

  addAsmElm(asMakeComment(0,""));

  addAsmElm(labMainExit);
  addAsmElm(asMakeInstr2(akMove,
      asMakeAdrLabelValue(labScopeAdr[0]),
      asMakeAdrReg(ebp),0,
      "restore frame ptr til main scope"));
  addAsmElm(asMakeInstr2(akMove, asMakeAdrReg(ebp),
      asMakeAdrReg(esp),0, "restore main start stak ptr"));
  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(ebp),0,""));
  addAsmElm(asMakeReturn(0,"return fra main"));
  addAsmElm(asMakeComment(0,"---------------------------"));

  /* funktionsliste */
  if (functionTopdown)
    abstractAsmFunctionDecl_list(s->val.eProgram.body->val.eBody.decl_list);

  addAsmElm(asMakeComment(0,"end of program"));

  fprintf(msgFile, "TONY kodegennerering af abstract assembler slut\r\n");
}

void abstractAsmFunction (sFunction *s)
{
  int save_abstractAsm_ScopeLevel = abstractAsm_ScopeLevel;
  labFentry = s->val.eFunction.head->labelEntry;
  labFexit = s->val.eFunction.head->labelExit;

  abstractAsm_ScopeLevel = save_abstractAsm_ScopeLevel + 1;

  addAsmElm(asMakeComment(0,""));
  /* næste funktions level*/
  if (!functionTopdown) {
  fprintf(msgFile, "BottomUp\n");
    abstractAsmFunctionDecl_list(s->val.eFunction.body->val.eBody.decl_list);
  }

  sprintf(abstractText,"#function %s_%d",
          labFentry->val.eLabel.name, labFentry->val.eLabel.nr);
  addAsmElm(asMakeComment(s->val.eFunction.head->linienr,abstractText));

  if (s->val.eFunction.head->val.eHead.par_decl_list->kind
      == kPar_decl_list_List) {
    addAsmElm(asMakeComment(0,""));
    abstractAsmOffsetFortegn = abstractAsmOffsetFortegn_parameter;
    if (abstractAsmOffsetFortegn==1)
      abstractAsmOffset = 8;
    else
      abstractAsmOffset = 8
```

```
      + s->val.eFunction.head->val.eHead.par_decl_list->dataLength;

    addAsmElm(asMakeComment(0,"Parameter adresser i dette scope"));
    abstractAsmVar_decl_list(
s->val.eFunction.head->val.eHead.par_decl_list->val.eList.var_decl_list);
  }
  addAsmElm(asMakeComment(0,""));
  addAsmElm(asMakeComment(0,"Data adresser defineret i dette scope"));
  abstractAsmOffsetFortegn = abstractAsmOffsetFortegn_lokale;
  if (abstractAsmOffsetFortegn == 1)
    abstractAsmOffset = - s->val.eFunction.body->dataLength;
  else
    abstractAsmOffset = 0;

  abstractAsmDataDecl_list(s->val.eFunction.body->val.eBody.decl_list);
  addAsmElm(asMakeComment(0,""));

  addAsmElm(labFentry);
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(ebp),0,"save ebp"));
  addAsmElm(asMakeInstr2(akMove,asMakeAdrReg(esp),asMakeAdrReg(ebp),
            0,"set frame ptr"));
  addAsmElm(asMakeInstr2(akMove,asMakeAdrReg(ebp),
            asMakeAdrLabelValue(labScopeAdr[abstractAsm_ScopeLevel]),0,
            "gem frame ptr til andre scopes"));
  addAsmElm(asMakeInstr2(akSub,
    asMakeAdrImm(s->val.eFunction.body->dataLength),
    asMakeAdrReg(esp),0,"reserver memory for lokale data"));
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(ebx),0,""));
/*  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(ecx),0,""));*/
/*  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(edx),0,""));*/
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(esi),0,""));
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(edi),0,""));
  addAsmElm(asMakeComment(0,"funktion instructions start"));

  abstractAsmStatement_list(s->val.eFunction.body->val.eBody.statement_list);

  addAsmElm(asMakeComment(0,"funktion instructions slut"));
  addAsmElm(labFexit);
  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(edi),0,""));
  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(esi),0,""));
/*  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(edx),0,""));*/
/*  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(ecx),0,""));*/
  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(ebx),0,""));
  addAsmElm(asMakeInstr2(akMove,asMakeAdrReg(ebp),asMakeAdrReg(esp),
            0,"restore stak ptr"));
  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(ebp),
            0,"restore caler frame-ptr"));
  addAsmElm(asMakeReturn(0,""));
  addAsmElm(asMakeComment(0,"----------------------------"));

  /* næste funktions level*/
  if (functionTopdown)
    abstractAsmFunctionDecl_list(s->val.eFunction.body->val.eBody.decl_list);

  abstractAsm_ScopeLevel = save_abstractAsm_ScopeLevel;
}
/*-------------------------------------------------------------*/
void abstractAsmType (sType *s)
{
  switch(s->kind)
  {
    case kType_Id:     abstractAsmType_Id     (s);break;
    case kType_Int:    abstractAsmType_Int    (s);break;
```

```
    case kType_Bool:   abstractAsmType_Bool   (s);break;
    case kType_Array:  abstractAsmType_Array  (s);break;
    case kType_Record: abstractAsmType_Record (s);break;
  }
}

void abstractAsmType_Id (sType *s)
{
}
void abstractAsmType_Int (sType *s)
{
}
void abstractAsmType_Bool (sType *s)
{
}
void abstractAsmType_Array (sType *s)
{
    abstractAsmType(s->val.eArray.type);
}
void abstractAsmType_Record (sType *s)
{
  int save_abstractAsm_ScopeLevel = abstractAsm_ScopeLevel;
  int save_abstractAsmOffset = abstractAsmOffset;
  int save_abstractAsmOffsetFortegn = abstractAsmOffsetFortegn;

  abstractAsmOffsetFortegn = abstractAsmOffsetFortegn_record;
  if (abstractAsmOffsetFortegn == 1)
    abstractAsmOffset = 0;
  else
    abstractAsmOffset = s->val.eRecord.elmLength;

  addAsmElm(asMakeComment(s->val.eRecord.var_decl_list->linienr
            ,"record of"));

  abstractAsmVar_decl_list(s->val.eRecord.var_decl_list);

  addAsmElm(asMakeComment(s->val.eRecord.var_decl_list->linienr
            ,"end of record"));

  abstractAsmOffsetFortegn = save_abstractAsmOffsetFortegn;
  abstractAsmOffset = save_abstractAsmOffset;
  abstractAsm_ScopeLevel = save_abstractAsm_ScopeLevel;
}
/*-----------------------------------------------------------------*/
void abstractAsmVar_decl_list (sVar_decl_list *s)
{
  switch(s->kind)
  {
    case kVar_decl_list_List: abstractAsmVar_decl_list_List (s);break;
    case kVar_decl_list_Var:  abstractAsmVar_decl_list_Var  (s);break;
  }
}
void abstractAsmVar_decl_list_List (sVar_decl_list *s)
{
  abstractAsmVar_decl_list(s->val.eList.var_decl_list);
  abstractAsmVar_type(s->val.eList.var_type);
}
void abstractAsmVar_decl_list_Var (sVar_decl_list *s)
{
  abstractAsmVar_type(s->val.eVar.var_type);
}
/*-----------------------------------------------------------------*/
void abstractAsmVar_type (sVar_type *s)
```

```
{
  int offset;
  if (abstractAsmOffsetFortegn == 1)
    offset = abstractAsmOffset
        +(s->dataOffset*abstractAsmOffsetFortegn);
  else
    offset = abstractAsmOffset
        +(s->dataOffset*abstractAsmOffsetFortegn)
        -s->val.eVar.type->dataLength;
  if (s->scopeLevel == abstractAsm_ScopeLevel)
    sprintf(abstractText,"%s lokal adr. på ebp",
      type_Kind_Txt(getSlutType(s->val.eVar.type)));
  else if (s->scopeLevel < 0)
    sprintf(abstractText,"%s indirekte adresse",
      type_Kind_Txt(getSlutType(s->val.eVar.type)));
  else
    sprintf(abstractText,"%s lokal adr. fra andet scope",
      type_Kind_Txt(getSlutType(s->val.eVar.type)));
  addAsmElm(asMakeOffset(s->val.eVar.id, s->labelnr,offset,
      s->linienr,abstractText));

  abstractAsmType(s->val.eVar.type);
}
/*------------------------------------------------------------------*/
void abstractAsmDataDecl_list (sDecl_list *s)
{
  switch(s->kind)
  {
    case kDecl_list_List:  abstractAsmDataDecl_list_List (s);break;
    case kDecl_list_Empty: break;                  break;
  }
}
void abstractAsmDataDecl_list_List (sDecl_list *s)
{
    abstractAsmDataDecl_list(s->val.eList.decl_list);
    abstractAsmDataDeclaration(s->val.eList.declaration);
}
void abstractAsmFunctionDecl_list (sDecl_list *s)
{
  switch(s->kind)
  {
    case kDecl_list_List:  abstractAsmFunctionDecl_list_List(s);break;
    case kDecl_list_Empty: break;                    break;
  }
}
void abstractAsmFunctionDecl_list_List (sDecl_list *s)
{
  abstractAsmFunctionDecl_list(s->val.eList.decl_list);
  abstractAsmFunctionDeclaration(s->val.eList.declaration);
}
/*------------------------------------------------------------------*/
void abstractAsmDataDeclaration (sDeclaration *s)
{
  switch(s->kind)
  {
    case kDeclaration_Type:  abstractAsmDeclaration_Type (s);break;
    case kDeclaration_Function: break;
    case kDeclaration_Var:  abstractAsmDeclaration_Var  (s);break;
  }
}
void abstractAsmFunctionDeclaration (sDeclaration *s)
{
  switch(s->kind)
```

```
  {
    case kDeclaration_Type:  break;
    case kDeclaration_Function: abstractAsmDeclaration_Function (s);break;
    case kDeclaration_Var:  break;
  }
}
void abstractAsmDeclaration_Type (sDeclaration *s)
{
  sType *type;
  if (!s->val.eType.aktiv) return;
  type = s->val.eType.type;
  while (type->kind == kType_Array)
    type = type->val.eArray.type;
  if (type->kind != kType_Record) return;

  sprintf(abstractText,"type %s", s->val.eType.id);
  addAsmElm(asMakeComment(s->linienr,abstractText));
  abstractAsmType(s->val.eType.type);
  sprintf(abstractText,"end type %s", s->val.eType.id);
  addAsmElm(asMakeComment(s->linienr,abstractText));
}
void abstractAsmDeclaration_Function (sDeclaration *s)
{
  abstractAsmFunction(s->val.eFunction.function);
}
void abstractAsmDeclaration_Var (sDeclaration *s)
{
  abstractAsmVar_decl_list(s->val.eVar.var_decl_list);
}
/*-------------------------------------------------------------------*/
void abstractAsmStatement_list (sStatement_list *s)
{
  switch(s->kind)
  {
    case kStatement_list_List:      abstractAsmStatement_list_List     (s);
            break;
    case kStatement_list_Statement: abstractAsmStatement_list_Statement(s);
            break;
  }
}
void abstractAsmStatement_list_List (sStatement_list *s)
{
  abstractAsmStatement_list(s->val.eList.statement_list);
  abstractAsmStatement(s->val.eList.statement);
}
void abstractAsmStatement_list_Statement (sStatement_list *s)
{
  abstractAsmStatement(s->val.eList.statement);
}
/*-------------------------------------------------------------------*/
void abstractAsmStatement (sStatement *s)
{
  if (!s->aktiv) return;
  switch(s->kind)
  {
    case kStatement_Return:    abstractAsmStatement_Return    (s);break;
    case kStatement_Write:     abstractAsmStatement_Write     (s);break;
    case kStatement_NewLength: abstractAsmStatement_NewLength (s);break;
    case kStatement_New:       abstractAsmStatement_New       (s);break;
    case kStatement_Assign:    abstractAsmStatement_Assign    (s);break;
    case kStatement_IfElse:    abstractAsmStatement_IfElse    (s);break;
    case kStatement_If:        abstractAsmStatement_If        (s);break;
    case kStatement_While:     abstractAsmStatement_While     (s);break;
```

```
    case kStatement_Compound:  abstractAsmStatement_Compound  (s);break;
    case kStatement_ForTo:     abstractAsmStatement_ForTo     (s);break;
    case kStatement_ForDownto: abstractAsmStatement_ForTo     (s);break;
  }
}
void abstractAsmStatement_Return (sStatement *s)
{
  addAsmElm(asMakeComment(s->linienr,"return"));
  abstractAsmExpression(s->val.eReturn.expression);
  addAsmElm(asMakeJump(akJmp,labFexit,0,"return med vaerdi i eax"));
  addAsmElm(asMakeComment(0,""));
}
void abstractAsmStatement_Write (sStatement *s)
{
  bool test = false;
  if (debugWrite) if (s->val.eWrite.expression->kind == kExp_Term)
      if (s->val.eWrite.expression->val.eTerm.term->kind == kTerm_Null)
        test = true;
  if (test) {
    if (debugWrite) dumpRegister(0);
    if (debugWrite) dumpHeap ();
    return;
  }

  addAsmElm(asMakeComment(s->linienr,"write"));
  abstractAsmExpression(s->val.eWrite.expression);
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(eax),0,""));
  addAsmElm(asMakeInstr1(akPush,asMakeAdrLabelAdr(labIntFormat),0,""));
  addAsmElm(asMakeCall(labPrintf,0,""));
  addAsmElm(asMakeInstr2(akAdd,asMakeAdrImm(8),asMakeAdrReg(esp),
            0,"Fjern parametre igen"));
  addAsmElm(asMakeComment(0,""));
}
void abstractAsmStatement_NewLength (sStatement *s)
{
  sType *arrayType = getSlutType(s->val.eNewLength.variable->type);
  sType *arrayElmType = getSlutType(arrayType->val.eArray.type);
  asElm *labNewLength1;
  asElm *labNewLength2;

  addAsmElm(asMakeComment(s->linienr,"new length"));

  abstractAsmExpression(s->val.eNewLength.expression);

  ++jmplabelnr;
  labNewLength1 = asMakeLabel2("_newLength",jmplabelnr,"1",0,"");
  labNewLength2 = asMakeLabel2("_newLength",jmplabelnr,"2",0,"");

  addAsmElm(asMakeInstr2(akOr,asMakeAdrReg(eax),asMakeAdrReg(eax),
            0,"test positivt antal elementer"));
  addAsmElm(asMakeJump(akJg,labNewLength1,0,""));
  addAsmElm(asMakeInstr2(akMove,asMakeAdrImm(s->linienr),asMakeAdrReg(eax),
            0,"linienr på fejl"));
  addAsmElm(asMakeJump(akJmp,labNonpositivelements,0,""));
  addAsmElm(labNewLength1);

  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(eax),0,"Save antal elm"));

  if (recordArray && arrayElmType->kind == kType_Record)
    addAsmElm(asMakeInstr2(akMove,
        asMakeAdrImm(arrayElmType->val.eRecord.elmLength),
              asMakeAdrReg(ebx),
              0,"ebx = elementlaengde"));
```

```
  else
    addAsmElm(asMakeInstr2(akMove,
        asMakeAdrImm(arrayType->val.eArray.elmLength),asMakeAdrReg(ebx),
              0,"ebx = elementlaengde"));

  addAsmElm(asMakeInstr1(akMul,asMakeAdrReg(ebx),
            0,"memory_size = elementer*elementlængde"));

  addAsmElm(asMakeInstr2(akAdd,asMakeAdrImm(4),asMakeAdrReg(eax),
            0,"plads til antals variabel"));

  addAsmElm(asMakeInstr2(akMove,
            asMakeAdrLabelValue(labMyheapfreeadr),asMakeAdrReg(esi),0,""));
  addAsmElm(asMakeInstr2(akMove,
            asMakeAdrReg(eax),asMakeAdrMemEx(0,esi,0,nreg),
            0,"gem allokeret mem size"));

  addAsmElm(asMakeInstr2(akAdd,asMakeAdrReg(esi),asMakeAdrReg(eax),
            0,"ny freeadr"));
  addAsmElm(asMakeInstr2(akAdd,asMakeAdrImm(4),asMakeAdrReg(eax),
            0,"beregnes"));
  addAsmElm(asMakeInstr2(akMove,
            asMakeAdrReg(eax),asMakeAdrLabelValue(labMyheapfreeadr),
            0,"og gemmes"));

  addAsmElm(asMakeInstr2(akCmp,
            asMakeAdrLabelAdr(labAftermyheapspace),asMakeAdrReg(eax),
            0,"test for out of memory"));
  addAsmElm(asMakeJump(akJng,labNewLength2,0,""));
  addAsmElm(asMakeInstr2(akMove,asMakeAdrImm(s->linienr),asMakeAdrReg(eax),
            0,"linienr på fejl"));
  addAsmElm(asMakeJump(akJmp,labOutofmemory,0,""));
  addAsmElm(labNewLength2);

  addAsmElm(asMakeInstr2(akLoadAdr,
            asMakeAdrMemEx(8,esi,0,nreg),asMakeAdrReg(eax),
            0,"adresse på array : offset -8 er mem.size og -4 antal elm."));

  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(ebx),0,""));
  addAsmElm(asMakeInstr2(akMove,
            asMakeAdrReg(ebx),
            asMakeAdrMemEx(-4,eax,0,nreg),
            0,"Gem antal elm. i array -4"));

  addAsmElm(asMakeComment(0,"store i variabel"));

  abstractAsmVariable(s->val.eNewLength.variable, -1);

  addAsmElm(asMakeComment(0,""));
}
void abstractAsmStatement_New (sStatement *s)
{
  sType *recordType = getSlutType(s->val.eNew.variable->type);
  asElm *labNew;

  addAsmElm(asMakeComment(s->linienr,"new"));

  ++jmplabelnr;
  labNew = asMakeLabel("_new",jmplabelnr,0,"");

  addAsmElm(asMakeInstr2(akMove,
            asMakeAdrImm(recordType->val.eRecord.elmLength),
            asMakeAdrReg(ebx),
```

```
            0,"edx = recordlaengde"));
  addAsmElm(asMakeInstr2(akMove,
            asMakeAdrLabelValue(labMyheapfreeadr),asMakeAdrReg(eax),
            0,""));
  addAsmElm(asMakeInstr2(akMove,
            asMakeAdrReg(ebx),asMakeAdrMemEx(0,eax,0,nreg),
            0,"gem allokeret mem size"));
  addAsmElm(asMakeInstr2(akAdd,asMakeAdrReg(ebx),asMakeAdrReg(eax),
            0,"ny freeadr"));
  addAsmElm(asMakeInstr2(akAdd,asMakeAdrImm(4),asMakeAdrReg(eax),
            0,"+ 4 byte til size"));
  addAsmElm(asMakeInstr2(akMove,
            asMakeAdrReg(eax),asMakeAdrLabelValue(labMyheapfreeadr),
            0,"gemmes"));
  addAsmElm(asMakeInstr2(akCmp,
            asMakeAdrLabelAdr(labAftermyheapspace),asMakeAdrReg(eax),
            0,"test for out of memory"));
  addAsmElm(asMakeJump(akJng,labNew,0,""));
  addAsmElm(asMakeInstr2(akMove,asMakeAdrImm(s->linienr),
            asMakeAdrReg(eax),
            0,"linienr på fejl"));
  addAsmElm(asMakeJump(akJmp,labOutofmemory,0,""));
  addAsmElm(labNew);

  addAsmElm(asMakeInstr2(akSub,
            asMakeAdrImm(recordType->val.eRecord.elmLength),
            asMakeAdrReg(eax),
            0,"adresse på record = freeadr - size"));
  addAsmElm(asMakeComment(0,"store i variabel"));

  abstractAsmVariable(s->val.eNew.variable, -1);

  addAsmElm(asMakeComment(0,""));
}
void abstractAsmStatement_Assign (sStatement *s)
{
  addAsmElm(asMakeComment(s->linienr,"assign"));
  abstractAsmExpression(s->val.eAssign.expression);
  abstractAsmVariable(s->val.eAssign.variable,-1);
  addAsmElm(asMakeComment(0,""));
}
void abstractAsmStatement_IfElse (sStatement *s)
{
  asElm *labIfElse;
  asElm *labIfEnd;

  ++jmplabelnr;
  labIfElse = asMakeLabel2("_if",jmplabelnr,"else",s->linienr,"");
  labIfEnd = asMakeLabel2("_if",jmplabelnr,"end",s->linienr,"");

  addAsmElm(asMakeComment(s->linienr,"if-then-else"));
  abstractAsmExpression(s->val.eIfElse.expression);

  addAsmElm(asMakeInstr2(akOr,asMakeAdrReg(eax),asMakeAdrReg(eax),
            0,"test for false"));
  addAsmElm(asMakeJump(akJe,labIfElse,0,"false -> hop"));
  addAsmElm(asMakeComment(0,""));

  abstractAsmStatement(s->val.eIfElse.ifStatement);

  addAsmElm(asMakeJump(akJmp,labIfEnd,0,"slut if"));

  addAsmElm(labIfElse);
```

```
  addAsmElm(asMakeComment(0,""));

  abstractAsmStatement(s->val.eIfElse.elseStatement);

  addAsmElm(labIfEnd);
  addAsmElm(asMakeComment(0,""));
}
void abstractAsmStatement_If (sStatement *s)
{
  asElm *labIfEnd;
  ++jmplabelnr;
  labIfEnd = asMakeLabel2("_if",jmplabelnr,"end",s->linienr,"");

  addAsmElm(asMakeComment(s->linienr,"if-then"));
  abstractAsmExpression(s->val.eIf.expression);

  addAsmElm(asMakeInstr2(akOr,asMakeAdrReg(eax),asMakeAdrReg(eax),
            0,"test for false"));
  addAsmElm(asMakeJump(akJe,labIfEnd,0,"false -> hop"));

  abstractAsmStatement(s->val.eIf.ifStatement);

  addAsmElm(labIfEnd);

  addAsmElm(asMakeComment(0,""));
}
void abstractAsmStatement_While (sStatement *s)
{
  asElm *labWhile;
  asElm *labWhileEnd;

  ++jmplabelnr;
  labWhile = asMakeLabel("_while",jmplabelnr,s->linienr,"");
  labWhileEnd = asMakeLabel2("_while",jmplabelnr,"end",s->linienr,"");

  addAsmElm(asMakeComment(s->linienr,"while"));

  addAsmElm(labWhile);
  abstractAsmExpression(s->val.eWhile.expression);

  addAsmElm(asMakeInstr2(akOr,asMakeAdrReg(eax),asMakeAdrReg(eax),
            0,"test for false"));
  addAsmElm(asMakeJump(akJe,labWhileEnd,0,"false -> hop"));

  abstractAsmStatement(s->val.eWhile.statement);

  addAsmElm(asMakeJump(akJmp,labWhile,0,""));
  addAsmElm(labWhileEnd);

  addAsmElm(asMakeComment(0,""));
}
void abstractAsmStatement_Compound (sStatement *s)
{
  abstractAsmStatement_list(s->val.eCompound.statement_list);
}
void abstractAsmStatement_ForTo (sStatement *s)
{
  asElm *labForTest;
  asElm *labForEnd;

  ++jmplabelnr;
  labForTest = asMakeLabel2("_for",jmplabelnr,"test",s->linienr,"");
  labForEnd = asMakeLabel2("_for",jmplabelnr,"end",s->linienr,"");
```

```
  if (s->kind == kStatement_ForTo)
    addAsmElm(asMakeComment(s->linienr,"for-to"));
  else
    addAsmElm(asMakeComment(s->linienr,"for-downto"));

  abstractAsmExpression(s->val.eForTo.expressionInit);
  abstractAsmVariable(s->val.eForTo.variable,-1);
  /* beregn to-exp i ebx */
  abstractAsmExpressionRecursiv(s->val.eForTo.expressionTo,2);

  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(ebx),
            0,"gem stopvalue til at teste mod"));

  addAsmElm(labForTest);
  addAsmElm(asMakeInstr2(akCmp,asMakeAdrMemEx(0,esp,0,nreg),
            asMakeAdrReg(eax),
            0,"test mod TO/DOWNTO-vaerdi"));
  if (s->kind == kStatement_ForTo)
    addAsmElm(asMakeJump(akJg,labForEnd,0,"graense naaet -> slut"));
  else
    addAsmElm(asMakeJump(akJl,labForEnd,0,"graense naaet -> slut"));

  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(eax),
            0,"gem counter til næste gennemloeb"));

  abstractAsmStatement(s->val.eForTo.statement);

  addAsmElm(asMakeInstr1(akPop,asMakeAdrReg(eax),
            0,"reetabler og juster counter for næste gennemloeb"));
  if (s->kind == kStatement_ForTo)
    addAsmElm(asMakeInstr1(akInc,asMakeAdrReg(eax),
              0,"step +1"));
  else
    addAsmElm(asMakeInstr1(akDec,asMakeAdrReg(eax),
              0,"step -1"));

  abstractAsmVariable(s->val.eForTo.variable,-1); /* save ny vaerdi */

  addAsmElm(asMakeJump(akJmp,labForTest,0,""));
  addAsmElm(labForEnd);
  addAsmElm(asMakeInstr2(akAdd,asMakeAdrImm(4),asMakeAdrReg(esp),
            0,"Fjern TO/DOWNTO-vaerdi igen"));

  addAsmElm(asMakeComment(0,""));
}
/*-----------------------------------------------------------------*/
void abstractAsmVariable (sVariable *s, int operant)
{
  switch(s->kind)
  {
    case kVariable_Id:      abstractAsmVariable_Id  (s, operant);break;
    case kVariable_Indexed: abstractAsmVariable_Indexed (s, operant);break;
    case kVariable_Struct:  abstractAsmVariable_Struct  (s, operant);break;
  }
}
void abstractAsmVariable_Id (sVariable *s, int operant)
{
  asAdr *opAdr = NULL;
  switch (operant)
  {
    case 1: opAdr = asMakeAdrReg(eax); break;
    case 2: opAdr = asMakeAdrReg(ebx); break;
```

```
    default:break;
  }

  if (s->val.eId.var_type->scopeLevel == abstractAsm_ScopeLevel) {
    if (operant >= 1)
      addAsmElm(asMakeInstr2(akMove,
                asMakeAdrMem(asMakeOffset(s->val.eId.var_type->val.eVar.id,
                s->val.eId.var_type->labelnr,0,0,""),
                ebp,0,nreg),
                opAdr,
                s->linienr, "Load lokal variabel"));
    else
      addAsmElm(asMakeInstr2(akMove,
                asMakeAdrReg(eax),
                asMakeAdrMem(asMakeOffset(s->val.eId.var_type->val.eVar.id,
                s->val.eId.var_type->labelnr,0,0,""),
                ebp,0,nreg),
                s->linienr, "Store lokal variabel"));
  }
  else if (s->val.eId.var_type->scopeLevel >= 0) {
    if (operant >= 1) {
      addAsmElm(asMakeInstr1(akPush, asMakeAdrReg(esi),
                s->linienr, ""));
      addAsmElm(asMakeInstr2(akMove,
                asMakeAdrLabelValue(
                labScopeAdr[s->val.eId.var_type->scopeLevel]),
                asMakeAdrReg(esi),
                s->linienr, "hent adr. på data fra andet scope"));
      addAsmElm(asMakeInstr2(akMove,
                asMakeAdrMem(asMakeOffset(s->val.eId.var_type->val.eVar.id,
                              s->val.eId.var_type->labelnr,0,0,""),
                              esi,0,nreg),
                opAdr,
                s->linienr, "Load variabel i andet scope"));
      addAsmElm(asMakeInstr1(akPop, asMakeAdrReg(esi),
                s->linienr, ""));
    }
    else {
      addAsmElm(asMakeInstr1(akPush, asMakeAdrReg(edi),
                s->linienr, ""));
      addAsmElm(asMakeInstr2(akMove,
                asMakeAdrLabelValue(
                labScopeAdr[s->val.eId.var_type->scopeLevel]),
                asMakeAdrReg(edi),
                s->linienr, "hent adr. på data fra andet scope"));
      addAsmElm(asMakeInstr2(akMove,
                asMakeAdrReg(eax),
                asMakeAdrMem(asMakeOffset(s->val.eId.var_type->val.eVar.id,
                              s->val.eId.var_type->labelnr,0,0,""),
                              edi,0,nreg),
                s->linienr, "Store variabel i andet scope"));
      addAsmElm(asMakeInstr1(akPop, asMakeAdrReg(edi),
                s->linienr, ""));
    }
  }
  else
    fprintf(msgFile,"???????? alvorligFejl i adresseberegninger");
}
void abstractAsmVariable_Indexed (sVariable *s, int operant)
{
addAsmElm(asMakeComment(s->linienr,"***** abstractAsmVariable_Indexed"));
  if (operant < 1) {
    addAsmElm(asMakeComment(s->linienr,"indexed store"));
```

```
    addAsmElm(asMakeInstr1(akPush, asMakeAdrReg(eax),
        s->linienr, "save 1. op"));
  }
  else {
    addAsmElm(asMakeComment(s->linienr,"indexed load"));
    if (operant == 2)
      addAsmElm(asMakeInstr1(akPush, asMakeAdrReg(eax),
                s->linienr, "save 1. op"));
  }

  /* beregn base adresse */
  abstractAsmVariableAdresse(s, operant);/* skal levere base-adresse i eax */

  addAsmElm(asMakeComment(s->linienr,"indexed load/store"));
  if (operant == 1){      /* load */
    addAsmElm(asMakeInstr2(akMove,
              asMakeAdrMemEx(0,eax,0,nreg),
              asMakeAdrReg(eax),
              s->linienr, "load 1. op. variabel"));
  }
  else if (operant == 2){    /* load */
    addAsmElm(asMakeInstr2(akMove,
              asMakeAdrMemEx(0,eax,0,nreg),
              asMakeAdrReg(ebx),
              s->linienr, "load 2. op. variabel"));
    addAsmElm(asMakeInstr1(akPop, asMakeAdrReg(eax),
              s->linienr, "hent 1. op værdi igen"));
  }
  else{           /* store */
    addAsmElm(asMakeInstr1(akPop, asMakeAdrReg(edx),
              s->linienr, "hent værdi igen"));
    addAsmElm(asMakeInstr2(akMove,
              asMakeAdrReg(edx),
              asMakeAdrMemEx(0,eax,0,nreg),s->linienr, "store variabel"));
  }
}
void abstractAsmVariable_Struct (sVariable *s, int operant)
{
  if (operant < 1) {
    addAsmElm(asMakeComment(s->linienr,"struct store"));
    addAsmElm(asMakeInstr1(akPush, asMakeAdrReg(eax),
              s->linienr, "save 1. op"));
  }
  else {
    addAsmElm(asMakeComment(s->linienr,"struct load"));
    if (operant == 2)
      addAsmElm(asMakeInstr1(akPush, asMakeAdrReg(eax),
                s->linienr, "save 1. op"));
  }

  /* beregn base adresse */
  /* skal levere base-adresse i eax */
  abstractAsmVariableAdresse(s->val.eStruct.variable, operant);

  addAsmElm(asMakeComment(0,"struct load/store"));
  if (operant == 1){      /* load */
    addAsmElm(asMakeInstr2(akMove,
              asMakeAdrMem(asMakeOffset(s->val.eId.var_type->val.eVar.id,
                      s->val.eId.var_type->labelnr,0,0,""),
                      eax,0,nreg),
              asMakeAdrReg(eax),
              s->linienr,
              "load 1. op. fra base adr. på struktur-variabel i eax"));
```

```
  }
  else if (operant == 2){    /* load */
    addAsmElm(asMakeInstr2(akMove,
              asMakeAdrMem(asMakeOffset(s->val.eId.var_type->val.eVar.id,
                    s->val.eId.var_type->labelnr,0,0,""),
                    eax,0,nreg),
              asMakeAdrReg(ebx),
              s->linienr,
              "load 2. op. fra base adr. på struktur-variabel i eax"));
    addAsmElm(asMakeInstr1(akPop, asMakeAdrReg(eax),
              s->linienr, "hent 1. op værdi igen"));
  }
  else if (operant < 0){    /* store */
    addAsmElm(asMakeInstr1(akPop, asMakeAdrReg(edx),
              s->linienr, "hent værdi igen"));
    addAsmElm(asMakeInstr2(akMove,
              asMakeAdrReg(edx),
              asMakeAdrMem(asMakeOffset(s->val.eId.var_type->val.eVar.id,
                      s->val.eId.var_type->labelnr,0,0,""),
                      eax,0,nreg),
              s->linienr, "base adr. på struktur-variabel i eax"));
  }
  else    {            /* fejl */
    fprintf(msgFile,"Fejl: struct var operant %d ikke def",operant);
  }
}

void abstractAsmVariableAdresse (sVariable *s, int operant)
{
  switch(s->kind)
  {
    case kVariable_Id:      abstractAsmVariableAdresse_Id     (s, operant);
          break;
    case kVariable_Indexed: abstractAsmVariableAdresse_Indexed(s, operant);
          break;
    case kVariable_Struct:  abstractAsmVariableAdresse_Struct (s, operant);
          break;
  }
}
void abstractAsmVariableAdresse_Id (sVariable *s, int operant)
{
  /* adresse variabel */
  if (s->val.eId.var_type->scopeLevel == abstractAsm_ScopeLevel)
    addAsmElm(asMakeInstr2(akMove,
              asMakeAdrMem(asMakeOffset(s->val.eId.var_type->val.eVar.id,
                          s->val.eId.var_type->labelnr,0,0,""),
                          ebp,0,nreg),
              asMakeAdrReg(eax),
              s->linienr, "hent adr."));
  else if (s->val.eId.var_type->scopeLevel  >= 0) {
    addAsmElm(asMakeInstr2(akMove,
              asMakeAdrLabelValue(
                      labScopeAdr[s->val.eId.var_type->scopeLevel]),
              asMakeAdrReg(eax),
              s->linienr, "hent adr. på lokaldata fra andet scope"));
    addAsmElm(asMakeInstr2(akMove,
              asMakeAdrMem(asMakeOffset(s->val.eId.var_type->val.eVar.id,
                          s->val.eId.var_type->labelnr,0,0,""),
                          eax,0,nreg),
              asMakeAdrReg(eax),
              s->linienr, "variabel adressse fra andet scope i eax"));
  }
}
```

```
void abstractAsmVariableAdresse_Indexed (sVariable *s, int operant)
{
  asElm *lab1;
  asElm *lab2;
  sType *arrayType = getSlutType(s->val.eIndexed.variable->type);
  sType *arrayElmType = getSlutType(arrayType->val.eArray.type);

  addAsmElm(asMakeComment(s->linienr,"calc indexed adr."));
  addAsmElm(asMakeInstr1(akPush, asMakeAdrReg(ebx),
            s->linienr, "save"));

  /* skal levere base-adresse i eax */
  abstractAsmVariableAdresse(s->val.eIndexed.variable, 99);

  /* skal levere index i ebx */
  abstractAsmExpressionRecursiv(s->val.eIndexed.expression, 2);

  ++jmplabelnr;
  lab1 = asMakeLabel2("_index",jmplabelnr,"test2",s->linienr,"");
  lab2 = asMakeLabel2("_index",jmplabelnr,"ok",s->linienr,"");

  addAsmElm(asMakeInstr2(akOr,
            asMakeAdrReg(ebx), asMakeAdrReg(ebx),
            s->linienr, "test for index underflow"));
  addAsmElm(asMakeJump(akJge,lab1,0,""));
  addAsmElm(asMakeInstr2(akMove,
            asMakeAdrImm(s->linienr), asMakeAdrReg(eax),
            s->linienr, "linienr på fejl"));
  addAsmElm(asMakeJump(akJmp,labIndexunderflow,0,""));
  addAsmElm(lab1);
  addAsmElm(asMakeInstr2(akCmp,
            asMakeAdrReg(ebx), asMakeAdrMemEx(-4,eax,0,nreg),
            s->linienr, "test for index overflow = offset fejl"));
  addAsmElm(asMakeJump(akJg,lab2,0,""));
  addAsmElm(asMakeInstr2(akMove,
            asMakeAdrImm(s->linienr), asMakeAdrReg(eax),
            s->linienr, "linienr på fejl"));
  addAsmElm(asMakeJump(akJmp,labIndexoverflow,0,""));
  addAsmElm(lab2);

  addAsmElm(asMakeInstr1(akPush, asMakeAdrReg(eax),
            s->linienr, "save array-adr."));

  if (recordArray && arrayElmType->kind == kType_Record)
    addAsmElm(asMakeInstr2(akMove,
          asMakeAdrImm(arrayElmType->val.eRecord.elmLength),
          asMakeAdrReg(eax),
          s->linienr, "hent elm. længde eax"));
  else
    addAsmElm(asMakeInstr2(akMove,
          asMakeAdrImm(arrayType->val.eArray.elmLength),
          asMakeAdrReg(eax),
          s->linienr, "hent elm. længde eax"));

  addAsmElm(asMakeInstr1(akMul, asMakeAdrReg(ebx),
            s->linienr, "eax = elm. offset"));
  addAsmElm(asMakeInstr1(akPop, asMakeAdrReg(ebx),
            s->linienr, "restore array-adr. i ebx"));

  if (operant == 99) {  /* ikke 1. level -> returner adr. elm.  */
    if (recordArray && arrayElmType->kind == kType_Record)
      addAsmElm(asMakeInstr2(akLoadAdr,
                asMakeAdrMemEx(0,ebx,1,eax), asMakeAdrReg(eax),
```

```
                s->linienr, "adr. paa array elm. adresse i eax"));
    else
      addAsmElm(asMakeInstr2(akMove,
                asMakeAdrMemEx(0,ebx,1,eax), asMakeAdrReg(eax),
                s->linienr, "adr. elm. i eax"));
  }
  else
    addAsmElm(asMakeInstr2(akLoadAdr,
              asMakeAdrMemEx(0,ebx,1,eax), asMakeAdrReg(eax),
              s->linienr, "adr. paa array elm. adresse i eax"));

  addAsmElm(asMakeInstr1(akPop, asMakeAdrReg(ebx),
            s->linienr, "reetabler"));
}

void abstractAsmVariableAdresse_Struct (sVariable *s, int operant)
{
  addAsmElm(asMakeComment(s->linienr,"calc struct adr."));

  abstractAsmVariableAdresse(s->val.eStruct.variable, 99);

  addAsmElm(asMakeInstr2(akMove,
            asMakeAdrMem(asMakeOffset(s->val.eId.var_type->val.eVar.id,
                         s->val.eId.var_type->labelnr,0,0,""),
                         eax,0,nreg),
            asMakeAdrReg(eax),
            s->linienr, "struct adresse i eax"));
}
/*-------------------------------------------------------------------*/
void abstractAsmExpression (sExpression *s)
{
  abstractAsmExpressionRecursiv (s, 1);
}
void abstractAsmExpressionRecursiv (sExpression *s, int operant)
{
  switch(s->kind)
  {
    case kExp_Eq:   abstractAsmExpression_Dyadisk (s,"==",operant);break;
    case kExp_Neq:  abstractAsmExpression_Dyadisk (s,"!=",operant);break;
    case kExp_Lt:   abstractAsmExpression_Dyadisk (s,"<",operant);break;
    case kExp_Gt:   abstractAsmExpression_Dyadisk (s,">",operant);break;
    case kExp_Lteq: abstractAsmExpression_Dyadisk (s,"<=",operant);break;
    case kExp_Gteq: abstractAsmExpression_Dyadisk (s,">=",operant);break;
    case kExp_Add:  abstractAsmExpression_Dyadisk (s,"+",operant);break;
    case kExp_Sub:  abstractAsmExpression_Dyadisk (s,"-",operant);break;
    case kExp_Mul:  abstractAsmExpression_Dyadisk (s,"*",operant);break;
    case kExp_Div:  abstractAsmExpression_Dyadisk (s,"/",operant);break;
    case kExp_Or:   abstractAsmExpression_Dyadisk (s,"||",operant);break;
    case kExp_And:  abstractAsmExpression_Dyadisk (s,"&&",operant);break;
    case kExp_Term: abstractAsmExpression_Term    (s,operant);break;

  }
}
void abstractAsmDyadiskInstruction (sExpression *s)
{
  asElm *lab1;
  asElm *lab2;
  int jmp;
  switch(s->kind)
  {
    case kExp_Eq:
    case kExp_Neq:
    case kExp_Lt:
```

```
    case kExp_Gt:
    case kExp_Lteq:
    case kExp_Gteq:
        ++jmplabelnr;
        lab1 = asMakeLabel2("_equation",jmplabelnr,"true",s->linienr,"");
        lab2 = asMakeLabel2("_equation",jmplabelnr,"end",s->linienr,"");
        addAsmElm(asMakeInstr2(akCmp,
                  asMakeAdrReg(ebx), asMakeAdrReg(eax),
                  s->linienr, "sammenlignings exp"));
        switch(s->kind)
        {
          case kExp_Eq:   jmp = akJe;  break;
          case kExp_Neq:  jmp = akJne; break;
          case kExp_Lt:   jmp = akJl;  break;
          case kExp_Gt:   jmp = akJg;  break;
          case kExp_Lteq: jmp = akJle; break;
          case kExp_Gteq: jmp = akJge; break;
        }
        addAsmElm(asMakeJump(jmp,lab1,0,""));
        addAsmElm(asMakeInstr2(akXor,
                  asMakeAdrReg(eax), asMakeAdrReg(eax),
                  s->linienr, "exp false"));
        addAsmElm(asMakeJump(akJmp,lab2,0,""));
        addAsmElm(lab1);
        addAsmElm(asMakeInstr2(akMove,
                  asMakeAdrImm(1), asMakeAdrReg(eax),
                  s->linienr, "exp true"));
        addAsmElm(lab2);
        break;
    case kExp_Add:
        addAsmElm(asMakeInstr2(akAdd,
                  asMakeAdrReg(ebx), asMakeAdrReg(eax),
                  s->linienr, "+"));
        break;
    case kExp_Sub:
        addAsmElm(asMakeInstr2(akSub,
                  asMakeAdrReg(ebx), asMakeAdrReg(eax),
                  s->linienr, "-"));
        break;
    case kExp_Mul:
        addAsmElm(asMakeInstr1(akMul,
                  asMakeAdrReg(ebx),
                  s->linienr, "*"));
        break;
    case kExp_Div:
        ++jmplabelnr;
        lab1 = asMakeLabel2("_divide",jmplabelnr,"",s->linienr,"");
        addAsmElm(asMakeInstr2(akOr,
                  asMakeAdrReg(ebx), asMakeAdrReg(ebx),
                  s->linienr, "test for zero-divide"));
        addAsmElm(asMakeJump(akJne,lab1,0,""));
        addAsmElm(asMakeInstr2(akMove,
                  asMakeAdrImm(s->linienr), asMakeAdrReg(eax),
                  s->linienr, "linienr på fejl"));
        addAsmElm(asMakeJump(akJmp,labZerodivide,0,""));
        addAsmElm(lab1);
        addAsmElm(asMakeCltd(0,"Klargoer for division"));
        addAsmElm(asMakeInstr1(akDiv,
                  asMakeAdrReg(ebx),
                  s->linienr, "/"));
        break;
    case kExp_Or:
        addAsmElm(asMakeInstr2(akOr,
```

```
                    asMakeAdrReg(ebx), asMakeAdrReg(eax),
                    s->linienr, "||"));
          break;
      case kExp_And:
          addAsmElm(asMakeInstr2(akAnd,
                    asMakeAdrReg(ebx), asMakeAdrReg(eax),
                    s->linienr, "&&"));
      break;
    }
}
void abstractAsmExpression_Dyadisk (sExpression *s, char const *operation,
              int operant)
{
  if (operant!=1){
    addAsmElm(asMakeInstr1(akPush,
        asMakeAdrReg(eax), s->linienr, "gem 1. operant eax"));
  }
  /* beregn 1. op i eax */
  abstractAsmExpressionRecursiv(s->val.eLeftRight.expressionLeft,1);
  /* beregn 2. op i ebx */
  abstractAsmExpressionRecursiv(s->val.eLeftRight.expressionRight,2);
  abstractAsmDyadiskInstruction (s);
  if (operant!=1){
    addAsmElm(asMakeInstr2(akMove,
              asMakeAdrReg(eax), asMakeAdrReg(ebx),
              s->linienr, "aflever i 2. operant ebx"));
    addAsmElm(asMakeInstr1(akPop,
              asMakeAdrReg(eax), s->linienr, "reetabeler 1. operant eax"));
  }
}
void abstractAsmExpression_Term (sExpression *s, int operant)
{
  abstractAsmTerm(s->val.eTerm.term, operant);
}
/*-----------------------------------------------------------------*/
void abstractAsmTerm (sTerm *s, int operant)
{
  switch(s->kind)
  {
    case kTerm_Variable: abstractAsmTerm_Variable(s,operant);break;
    case kTerm_Call:   abstractAsmTerm_Call   (s,operant);break;
    case kTerm_Parantes: abstractAsmTerm_Parantes(s,operant);break;
    case kTerm_Not:   abstractAsmTerm_Not   (s,operant);break;
    case kTerm_Numeric:  abstractAsmTerm_Numeric (s,operant);break;
    case kTerm_Num:   abstractAsmTerm_Num   (s,operant);break;
    case kTerm_True:   abstractAsmTerm_True   (s,operant);break;
    case kTerm_False:   abstractAsmTerm_False   (s,operant);break;
    case kTerm_Null:   abstractAsmTerm_Null   (s,operant);break;
  }
}
void abstractAsmTerm_Variable (sTerm *s, int operant)
{
  abstractAsmVariable(s->val.eVariable.variable, operant);
}
void abstractAsmTerm_Call (sTerm *s, int operant)
{
  int parmBlokSize = s->val.eCall.head->val.eHead.par_decl_list->dataLength;

  if (operant!=1){
    addAsmElm(asMakeInstr1(akPush,
        asMakeAdrReg(eax), s->linienr, "gem 1. operant eax"));
  }
  if (parmBlokSize != 0) {
```

```
    addAsmElm(asMakeInstr1(akPush,
              asMakeAdrReg(edi), s->linienr, "save edi"));
    addAsmElm(asMakeInstr2(akSub,
              asMakeAdrImm(parmBlokSize), asMakeAdrReg(esp),
              s->linienr, "reserver memory for parameterblok"));

    addAsmElm(asMakeInstr2(akMove,
              asMakeAdrReg(esp), asMakeAdrReg(edi),
              s->linienr, "etabler base-reference til parametre"));
    addAsmElm(asMakeInstr2(akSub,
              asMakeAdrImm(8), asMakeAdrReg(edi),
              s->linienr, "juster som ebp efter kald"));
  }

  abstractAsmAct_list(s->val.eCall.act_list);

  addAsmElm(asMakeCall(s->val.eCall.head->labelEntry,
            s->linienr,"aktiver funktion"));

  if (parmBlokSize != 0) {
    addAsmElm(asMakeInstr2(akAdd,
              asMakeAdrImm(parmBlokSize), asMakeAdrReg(esp),
              s->linienr, "frigiv parameterblok"));
    addAsmElm(asMakeInstr1(akPop,
              asMakeAdrReg(edi), s->linienr, "restore edi"));
  }
  if (operant!=1){
    addAsmElm(asMakeInstr2(akMove, asMakeAdrReg(eax), asMakeAdrReg(ebx),
              s->linienr, "aflever i 2. operant ebx"));
    addAsmElm(asMakeInstr1(akPop,
              asMakeAdrReg(eax), s->linienr, "reetabeler 1. operant eax"));
  }
}
void abstractAsmTerm_Parantes (sTerm *s, int operant)
{
  abstractAsmExpressionRecursiv(s->val.eParantes.expression, operant);
}
void abstractAsmTerm_Not (sTerm *s, int operant)
{
  asAdr *toAdr = NULL;
  switch (operant)
  {
    case 1: toAdr = asMakeAdrReg(eax); break;
    case 2: toAdr = asMakeAdrReg(ebx); break;
    default:
      fprintf(msgFile,"Fejl: NOT operant %d ikke def",operant);
  }
  abstractAsmTerm(s->val.eNot.term, operant);
  addAsmElm(asMakeInstr2(akXor,
            asMakeAdrImm(1), toAdr,s->linienr, ">0 ?"));
}
void abstractAsmTerm_Numeric (sTerm *s, int operant)
{
  asElm *labNumeric;
  asAdr *rAdr = NULL;
  int rReg;
  switch (operant)
  {
    case 1: rReg = eax; break;
    case 2: rReg = ebx; break;
    default:
      fprintf(msgFile,"Fejl: NUMERIC operant %d ikke def",operant);
  }
```

```
  rAdr = asMakeAdrReg(rReg);

  addAsmElm(asMakeComment(s->linienr,"Numeric"));
  abstractAsmExpressionRecursiv(s->val.eNumeric.expression, operant);

  if (s->val.eNumeric.expression->type->slutType->kind == kType_Int) {
    ++jmplabelnr;
    labNumeric = asMakeLabel("_positiv",jmplabelnr,s->linienr,"");
    addAsmElm(asMakeInstr2(akOr, rAdr, rAdr,s->linienr, ">0 ?"));
    addAsmElm(asMakeJump(akJge,labNumeric,0,""));
    addAsmElm(asMakeInstr1(akNeg, rAdr, s->linienr, "vend fortegn"));
    addAsmElm(labNumeric);
  }
  else if (s->val.eNumeric.expression->type->slutType->kind == kType_Array) {
    addAsmElm(asMakeInstr2(akMove,
              asMakeAdrMemEx (-4,rReg,0,nreg), rAdr,
              s->linienr, "hent længde på array"));
  }
}
void abstractAsmTerm_Num (sTerm *s, int operant)
{
  asAdr *toAdr = NULL;
  switch (operant)
  {
    case 1: toAdr = asMakeAdrReg(eax); break;
    case 2: toAdr = asMakeAdrReg(ebx); break;
    default:
      fprintf(msgFile,"Fejl: TRUE operant %d ikke def",operant);
  }
  addAsmElm(asMakeInstr2(akMove,
          asMakeAdrImm(s->val.eNum.num), toAdr,s->linienr, "num"));
}
void abstractAsmTerm_True (sTerm *s, int operant)
{
  asAdr *toAdr = NULL;
  switch (operant)
  {
    case 1: toAdr = asMakeAdrReg(eax); break;
    case 2: toAdr = asMakeAdrReg(ebx); break;
    default:
      fprintf(msgFile,"Fejl: TRUE operant %d ikke def",operant);
  }
  addAsmElm(asMakeInstr2(akMove,
            asMakeAdrImm(1), toAdr,s->linienr, "true"));
}
void abstractAsmTerm_False (sTerm *s, int operant)
{
  asAdr *toAdr = NULL;
  switch (operant)
  {
    case 1: toAdr = asMakeAdrReg(eax); break;
    case 2: toAdr = asMakeAdrReg(ebx); break;
    default:
      fprintf(msgFile,"Fejl: FALSE operant %d ikke def",operant);
  }
  addAsmElm(asMakeInstr2(akXor,
            toAdr, toAdr, s->linienr, "false"));
}
void abstractAsmTerm_Null (sTerm *s, int operant)
{
  asAdr *toAdr = NULL;
  switch (operant)
  {
```

```
    case 1: toAdr = asMakeAdrReg(eax); break;
    case 2: toAdr = asMakeAdrReg(ebx); break;
    default:
      fprintf(msgFile,"Fejl: NULL operant %d ikke def",operant);
  }
  addAsmElm(asMakeInstr2(akMove, asMakeAdrImm(-1), toAdr,
            s->linienr,"null"));
}
/*------------------------------------------------------------------*/
void abstractAsmAct_list (sAct_list *s)
{
  switch(s->kind)
  {
    case kAct_list_List:  abstractAsmAct_list_List  (s);break;
    case kAct_list_Empty: abstractAsmAct_list_Empty (s);break;
  }
}
void abstractAsmAct_list_List (sAct_list *s)
{
  abstractAsmExp_list(s->val.eList.exp_list);
}
void abstractAsmAct_list_Empty (sAct_list *s)
{
}
/*------------------------------------------------------------------*/
void abstractAsmExp_list (sExp_list *s)
{
  switch(s->kind)
  {
    case kExp_list_List:       abstractAsmExp_list_List      (s);break;
    case kExp_list_Expression: abstractAsmExp_list_Expression(s);break;
  }
}
void abstractAsmExp_list_List (sExp_list *s)
{
  abstractAsmExp_list(s->val.eList.exp_list);
  abstractAsmExpressionRecursiv(s->val.eList.expression, 1);
  addAsmElm(asMakeInstr2(akMove,
            asMakeAdrReg(eax),
            asMakeAdrMem(asMakeOffset(s->val.eList.var_type->val.eVar.id,
                    s->val.eList.var_type->labelnr,0, 0,""),edi,0,nreg),
            s->linienr,"gem parameter paa stak"));
}
void abstractAsmExp_list_Expression (sExp_list *s)
{
  abstractAsmExpressionRecursiv(s->val.eExpression.expression, 1);
  addAsmElm(asMakeInstr2(akMove,
            asMakeAdrReg(eax),
            asMakeAdrMem(asMakeOffset(
                        s->val.eExpression.var_type->val.eVar.id,
                        s->val.eExpression.var_type->labelnr,0, 0,""),
                        edi,0,nreg),
            s->linienr,"gem parameter paa stak"));
}
```

Optimize fasen

Funktioner til peep-hole optimering

tonyAbstractAsmPeepHole.c

```
/* ------------------------------------------------------------------
  Realisering af funktioner til peep hole optimering
  Programmør: Bjørk Busch
  Historik:   2005.05.10
  Historik:   2005.05.13 (optimizenr tælles ned fra 0 istedet)
------------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include "bjbu_std.h"
#include "symbol.h"
#include "tonyAbstractAsm.h"

extern char *getRegName (int reg);
extern char *getInstrName (asElm *e);


/* head på abstract assembler liste */
extern struct asElm *asmListeHead;
/* tail på abstract assembler liste */
extern struct asElm *asmListeTail;

extern FILE *msgFile;
extern bool optimizeLog;

/* precond: altid mindst en kommentar i start og slut */
struct asElm *focusInstr;
struct asElm *currentInstr;   /* sidste - der indsættes efter*/
bool optimer = true;
int antalSlettede = 0;
int antalSlettedeMul = 0;
int antalSlettedeJump = 0;
int optimizenr = -1;

asElm *insertAsmElm (asElm *elm); /* insættes efter current */
void   deleteInstr (asElm *elm);  /* sletmarker elm */
asElm *focusNextInstr ();          /* flyt til næste instruktion */
asElm *nextInstr (asElm *elm);    /* hent næste instruktion */
/*------------------------------------------------------------------
Funktioner for peep-hole optimering af Abstract assembler-Strukturer
------------------------------------------------------------------*/
void moveToFocusInstr(){
    currentInstr = focusInstr;
}
asElm *insertAsmElm (asElm *elm) { /* insættes efter current */

  elm->next = currentInstr->next;
  currentInstr->next = elm;
  currentInstr = elm;
  --antalSlettede;
}
void   deleteInstr (asElm *elm) {  /* sletmarker elm */
  elm->slettet = true;
  elm->linienr = optimizenr;
  ++antalSlettede;
}
asElm *focusNextInstr () {         /* flyt current til næste instruktion */
```

```
  while (true)
  {
    if (focusInstr == NULL) return asmListeTail;
    if (focusInstr->next == NULL) return asmListeTail;
    focusInstr = focusInstr->next;
    currentInstr = focusInstr;
    if (!focusInstr->slettet) {
        switch (focusInstr->kind){
        case akComment: break;
        case akDef:     break;
        case akOffset:  break;
        case akLabel:   break;
        default:        return focusInstr;
                        break;
      }
    }
  }
  return asmListeTail;
}
asElm *nextInstr (asElm *elm) { /* hent næste instruktion */
  while (true)
  {
    if (elm == NULL) return asmListeTail;
    if (elm->next == NULL) return asmListeTail;
    elm = elm->next;
    currentInstr = elm;
    if (!elm->slettet) {
      switch (elm->kind){
        case akComment: break;
        case akDef:     break;
        case akOffset:  break;
        default:        return elm;
                        break;
      }
    }
  }
  return asmListeTail;
}

bool equalsAdr (asAdr *adr1, asAdr *adr2)
{
  if (adr1->kind == adr2->kind)
    switch (adr1->kind)
    {
      case aaReg:
        if (adr1->val.eReg.reg != adr2->val.eReg.reg)
          return false;
        return true;
      case aaMem:
        if (adr1->val.eMem.offset->val.eOffset.nr !=    /* nr er unique */
            adr2->val.eMem.offset->val.eOffset.nr) return false;
        if (adr1->val.eMem.base != adr2->val.eMem.base) return false;
        if (adr1->val.eMem.faktor != adr2->val.eMem.faktor) return false;
        if (adr1->val.eMem.disp != adr2->val.eMem.disp) return false;
        return true;
      case aaMemEx:
        if (adr1->val.eMemEx.offset != adr2->val.eMemEx.offset) return false;
        if (adr1->val.eMemEx.base != adr2->val.eMemEx.base) return false;
        if (adr1->val.eMemEx.faktor != adr2->val.eMemEx.faktor) return false;
        if (adr1->val.eMemEx.disp != adr2->val.eMemEx.disp) return false;
        return true;
      case aaImm:
        if (adr1->val.eImm.imm != adr2->val.eImm.imm)
```

```
          return false;
        return true;
      case aaLabelValue:
        if (adr1->val.eLabel.label != adr2->val.eLabel.label)
          return false;
        return true;
      case aaLabelAdr:
        if (adr1->val.eLabel.label != adr2->val.eLabel.label)
          return false;
        return true;
    }
  return false;
}

void peepHoleOptimizeAddSub2() /* add/sub 2. operant */
{
  asElm *elm2;
  int kind;
  if (focusInstr->slettet) return;
  /* mønster:   (tilsvarede for subl) (regx != regy)
       movl adr,regx
       addl regx,regy
     hvis adr = 1 eller - 1 reduceres til:
       incl regy  eller decl regy
     ellers reduceres til:
       addl adr,regy
  */
  if (focusInstr->kind != akMove) return;
  if (focusInstr->val.eInstr2.tAdr->kind != aaReg) return;
  elm2 = nextInstr(focusInstr);
  if (elm2->kind != akAdd && elm2->kind != akSub) return;
  if (!equalsAdr(focusInstr->val.eInstr2.tAdr,elm2->val.eInstr2.fAdr))
    return;
  if (equalsAdr(elm2->val.eInstr2.fAdr,elm2->val.eInstr2.tAdr))
    return;
  kind = elm2->kind;
  if (focusInstr->val.eInstr2.fAdr->kind == aaImm) {
    if (focusInstr->val.eInstr2.fAdr->val.eImm.imm == 1)
      if (elm2->kind == akAdd) kind = akInc;
      else                     kind = akDec;
    else if (focusInstr->val.eInstr2.fAdr->val.eImm.imm == -1)
      if (elm2->kind == akAdd) kind = akDec;
      else                     kind = akInc;
  }
  if (kind == akInc || kind == akDec)
      insertAsmElm(asMakeInstr1(kind,elm2->val.eInstr2.tAdr,
                   optimizenr,"optimized add/sub")); /*NB samme adr. elm !!*/
  else
    insertAsmElm(asMakeInstr2(kind,
                 focusInstr->val.eInstr2.fAdr,
                 elm2->val.eInstr2.tAdr,
                 optimizenr,"optimized add/sub")); /*NB samme adr. elm !!*/
  deleteInstr (focusInstr);
  deleteInstr (elm2);
  focusNextInstr(); /* flyt til nyindsatte*/
  optimer = true;
  if (optimizeLog)
    fprintf(msgFile, "add/sub optimize: %d\r\n",optimizenr--);
}

void peepHoleOptimizeAddConst1() /* add const 1. operant */
{
  asElm *elm2;
```

```
  asElm *elm3;
  int kind;
  if (focusInstr->slettet) return;
  /* mønster:   (regx != regy && adr != regx)
       movl const,regx
       movl adr,regy
       addl regy,regx
     hvis const = 1 eller - 1 reduceres til:
       movl adr,regx
       incl regx  eller decl regx
     ellers reduceres til:
       movl adr,regx
       addl const,regx
  */
  if (focusInstr->kind != akMove) return;
  if (focusInstr->val.eInstr2.fAdr->kind != aaImm) return;
  if (focusInstr->val.eInstr2.tAdr->kind != aaReg) return;
  elm2 = nextInstr(focusInstr);
  if (elm2->kind != akMove) return;
  if (equalsAdr(focusInstr->val.eInstr2.tAdr,elm2->val.eInstr2.tAdr))
    return;
  if (equalsAdr(focusInstr->val.eInstr2.tAdr,elm2->val.eInstr2.fAdr))
    return;
  if (elm2->val.eInstr2.tAdr->kind != aaReg) return;
  elm3 = nextInstr(elm2);
  if (elm3->kind != akAdd) return;
  if (focusInstr->val.eInstr2.fAdr->val.eImm.imm == 1)
    kind = akInc;
  else if (focusInstr->val.eInstr2.fAdr->val.eImm.imm == -1)
    kind = akDec;
  else
    kind = akAdd;

  if (kind == akInc || kind == akDec) {
    insertAsmElm(asMakeInstr2(akMove,
                elm2->val.eInstr2.fAdr,
                elm3->val.eInstr2.tAdr,
                optimizenr,"optimized add/sub const 1. op"));
                /*NB samme adr. elm !!*/
    insertAsmElm(asMakeInstr1(kind,elm3->val.eInstr2.tAdr,
                 optimizenr,"optimized add/sub const 1. op"));
                 /*NB samme adr. elm !!*/
  }
  else {
    insertAsmElm(asMakeInstr2(akMove,
                 elm2->val.eInstr2.fAdr,
                 elm3->val.eInstr2.tAdr,
                 optimizenr,"optimized add/sub const 1. op"));
                 /*NB samme adr. elm !!*/
    insertAsmElm(asMakeInstr2(akAdd,
                 focusInstr->val.eInstr2.fAdr,
                 elm3->val.eInstr2.tAdr,
                 optimizenr,"optimized add/sub const 1. op"));
                 /*NB samme adr. elm !!*/
  }
  deleteInstr (focusInstr);
  deleteInstr (elm2);
  deleteInstr (elm3);
  focusNextInstr(); /* flyt til nyindsatte*/
  optimer = true;
  if (optimizeLog)
    fprintf(msgFile, "add/sub const 1. op optimize: %d\r\n",optimizenr--);
}
```

```
void peepHoleOptimizeAddSubConstTwize()
{
  asElm *elm2;
  int kind;
  int const3;
  if (focusInstr->slettet) return;
  /* mønster:
       addl const1,adr2   eller subl const1,adr2
       addl const2,adr2   eller subl const1,adr2
     reduceres til:
       addl const3,adr2
  */
  /* mønster:
       movl const1,adr2
       addl const2,adr2   eller subl const1,adr2
     reduceres til:
       movl const3,adr2
  */
  if (focusInstr->kind != akAdd && focusInstr->kind != akSub &&
      focusInstr->kind != akMove) return;
  if (focusInstr->val.eInstr2.fAdr->kind != aaImm) return;
  elm2 = nextInstr(focusInstr);
  if (elm2->kind != akAdd && elm2->kind != akSub) return;
  if (elm2->val.eInstr2.fAdr->kind != aaImm) return;
  if (!equalsAdr(focusInstr->val.eInstr2.tAdr,elm2->val.eInstr2.tAdr))
    return;
  if (focusInstr->kind == akSub)
      const3 = -focusInstr->val.eInstr2.fAdr->val.eImm.imm;
  else
      const3 = focusInstr->val.eInstr2.fAdr->val.eImm.imm;
  if (elm2->kind == akSub)
      const3 -= elm2->val.eInstr2.fAdr->val.eImm.imm;
  else
      const3 += elm2->val.eInstr2.fAdr->val.eImm.imm;

  if (focusInstr->kind == akMove) kind = akMove;
  else                            kind = akAdd;
  insertAsmElm(asMakeInstr2(kind,
              asMakeAdrImm (const3),
              elm2->val.eInstr2.tAdr,
              optimizenr,"optimized twize add/sub const"));
  deleteInstr (focusInstr);
  deleteInstr (elm2);
  focusNextInstr(); /* flyt til nyindsatte*/
  optimer = true;
  if (optimizeLog)
    fprintf(msgFile, "Add/sub const twize optimize: %d\r\n",optimizenr--);
}

void peepHoleOptimizeCompare()
{
  asElm *elm2;
  bool const0 = false;
  if (focusInstr->slettet) return;
  /* mønster:
       movl adr,regx
       cmpl regx,regy
     reduceres til:
       cmpl adr,regy
  */
  if (focusInstr->kind != akMove) return;
  if (focusInstr->val.eInstr2.tAdr->kind != aaReg) return;
```

```
  elm2 = nextInstr(focusInstr);
  if (elm2->kind != akCmp) return;
  if (!equalsAdr(focusInstr->val.eInstr2.tAdr,elm2->val.eInstr2.fAdr))
    return;
/*
  if (focusInstr->val.eInstr2.fAdr->kind == aaImm)
    if (focusInstr->val.eInstr2.fAdr->val.eImm.imm == 0)
      const0 = true;
*/
  if (const0)
    insertAsmElm(asMakeInstr2(akOr,
                 elm2->val.eInstr2.tAdr,
                 elm2->val.eInstr2.tAdr,
                 optimizenr,"optimized cmp")); /*NB samme adr. elm !!*/
  else
    insertAsmElm(asMakeInstr2(akCmp,
                 focusInstr->val.eInstr2.fAdr,
                 elm2->val.eInstr2.tAdr,
                 optimizenr,"optimized cmp")); /*NB samme adr. elm !!*/

  deleteInstr (focusInstr);
  deleteInstr (elm2);
  focusNextInstr(); /* flyt til nyindsatte*/
  optimer = true;
  if (optimizeLog)
    fprintf(msgFile, "Cmp optimize: %d\r\n",optimizenr--);
}

void peepHoleOptimizeStoreLoad()
{
  asElm *elm2;
  if (focusInstr->slettet) return;
  /* mønster:
       movl reg,adr
       movl adr,reg
     reduceres til:
       movl reg,adr
  */
  if (focusInstr->kind != akMove) return;
  if (focusInstr->val.eInstr2.fAdr->kind != aaReg) return;
  elm2 = nextInstr(focusInstr);
  if (elm2->kind != akMove) return;
  if (!equalsAdr(focusInstr->val.eInstr2.fAdr,elm2->val.eInstr2.tAdr))
    return;
  if (!equalsAdr(focusInstr->val.eInstr2.tAdr,elm2->val.eInstr2.fAdr))
    return;
  deleteInstr (elm2);
  optimer = true;
  if (optimizeLog)
    fprintf(msgFile, "Load/store optimize: %d\r\n",optimizenr--);
}

void peepHoleOptimizeMoveTwize()
{
  asElm *elm2;
  if (focusInstr->slettet) return;
  /* mønster: (tilsvarende for leal)
    (hvis adr1 aaMem eller aaMemEx må base og disp ikke være adr2)
       movl adr1,adr2
         * antal push adrx eller pop adry
       movl adr1,adr2
     reduceres til:
       movl adr1,adr2
```

```
        * antal push adrx eller pop adry

      push adrx eller pop adry skal gælde
       adry != adr1 og adry != adr2
       hvis adr1 aaMem eller aaMemEx må base og disp ikke være adry
  */
  if (focusInstr->kind != akMove && focusInstr->kind != akLoadAdr) return;
  elm2 = nextInstr(focusInstr);
  while (elm2->kind == akPush ||elm2->kind == akPop) {
    if (elm2->kind == akPop) {
      if (equalsAdr(focusInstr->val.eInstr2.fAdr,elm2->val.eInstr1.adr))
        return;
      if (equalsAdr(focusInstr->val.eInstr2.tAdr,elm2->val.eInstr1.adr))
        return;
      if (focusInstr->val.eInstr2.fAdr->kind == aaMemEx)
        if (elm2->val.eInstr1.adr->kind == aaReg)
          if (focusInstr->val.eInstr2.fAdr->val.eMemEx.base ==
              elm2->val.eInstr1.adr->val.eReg.reg
              ||focusInstr->val.eInstr2.fAdr->val.eMemEx.disp ==
                elm2->val.eInstr1.adr->val.eReg.reg)
        return;
      if (focusInstr->val.eInstr2.fAdr->kind == aaMem)
        if (elm2->val.eInstr1.adr->kind == aaReg)
          if (focusInstr->val.eInstr2.fAdr->val.eMem.base ==
              elm2->val.eInstr1.adr->val.eReg.reg
              ||focusInstr->val.eInstr2.fAdr->val.eMem.disp ==
                elm2->val.eInstr1.adr->val.eReg.reg)
        return;
    }
    elm2 = nextInstr(elm2);
  }
  if (elm2->kind != focusInstr->kind) return;
  if (!equalsAdr(focusInstr->val.eInstr2.fAdr,elm2->val.eInstr2.fAdr))
    return;
  if (!equalsAdr(focusInstr->val.eInstr2.tAdr,elm2->val.eInstr2.tAdr))
    return;
  if (focusInstr->val.eInstr2.fAdr->kind == aaMemEx)
    if (focusInstr->val.eInstr2.tAdr->kind == aaReg)
      if (focusInstr->val.eInstr2.fAdr->val.eMemEx.base ==
          focusInstr->val.eInstr2.tAdr->val.eReg.reg
          ||focusInstr->val.eInstr2.fAdr->val.eMemEx.disp ==
            focusInstr->val.eInstr2.tAdr->val.eReg.reg)
    return;
  if (focusInstr->val.eInstr2.fAdr->kind == aaMem)
    if (focusInstr->val.eInstr2.tAdr->kind == aaReg)
      if (focusInstr->val.eInstr2.fAdr->val.eMem.base ==
          focusInstr->val.eInstr2.tAdr->val.eReg.reg
          ||focusInstr->val.eInstr2.fAdr->val.eMem.disp ==
            focusInstr->val.eInstr2.tAdr->val.eReg.reg)
    return;
  deleteInstr (elm2);

  if (optimizeLog)
    fprintf(msgFile, "Move twize optimize: %d\r\n",optimizenr--);

  optimer = true;
}

void peepHoleOptimizeLoadLoadImm()
{
  asElm *elm2;
  if (focusInstr->slettet) return;
  /* mønster:
```

```
        movl imm,regx
        movl regx,regy
      reduceres til:
        movl imm,regy
  */
  if (focusInstr->kind != akMove) return;
  if (focusInstr->val.eInstr2.fAdr->kind != aaImm) return;
  if (focusInstr->val.eInstr2.tAdr->kind != aaReg) return;
  elm2 = nextInstr(focusInstr);
  if (elm2->kind != akMove) return;
  if (elm2->val.eInstr2.tAdr->kind != aaReg) return;
  if (!equalsAdr(focusInstr->val.eInstr2.tAdr,elm2->val.eInstr2.fAdr))
    return;
  insertAsmElm(asMakeInstr2(akMove,
              focusInstr->val.eInstr2.fAdr,
              elm2->val.eInstr2.tAdr,
              optimizenr,"optimized load-load-const")); /*NB samme adr. elm
!!*/
  deleteInstr (focusInstr);
  deleteInstr (elm2);

  if (optimizeLog)
    fprintf(msgFile, "Load-load-const optimize: %d\r\n",optimizenr--);

  focusNextInstr(); /* flyt til nyindsatte*/
  optimer = true;
}

void peepHoleOptimizeMulConst2() /* 2. operant const */
{
  asElm *elm2;
  int imm;
  if (focusInstr->slettet) return;
  /*
     mønster: (regx != eax)
       movl 0,regx
       imul regx
     reduceres til:
       xorl eax
     mønster: (regx != eax)
       movl 1,regx
       imul regx
     reduceres til:
       ingenting
     mønster: (regx != eax)
       movl -1,regx
       imul regx
     reduceres til:
       negl eax
     mønster: (regx != eax)
       movl 2,regx
       imul regx
     reduceres til:
       addl eax,eax
     mønster: (regx != eax)
       movl -2,regx
       imul regx
     reduceres til: (regx != eax)
       addl eax,eax
       negl eax
  */
  if (focusInstr->kind != akMove) return;
  if (focusInstr->val.eInstr2.fAdr->kind != aaImm) return;
```

```
  imm = focusInstr->val.eInstr2.fAdr->val.eImm.imm;
  if (imm != 1 && imm != -1 && imm != 2 && imm != -2 && imm != 0) return;
  if (focusInstr->val.eInstr2.tAdr->kind != aaReg) return;
  if (focusInstr->val.eInstr2.tAdr->val.eReg.reg == eax) return;
  elm2 = nextInstr(focusInstr);
  if (elm2->kind != akMul) return;
  if (!equalsAdr(focusInstr->val.eInstr2.tAdr,elm2->val.eInstr1.adr))
    return;
  if (imm == 0)
    insertAsmElm(asMakeInstr2(akXor,
                 asMakeAdrReg(eax),
                 asMakeAdrReg(eax),
                 optimizenr,"optimized mul 2.op er 0"));
  else if (imm == -1)
    insertAsmElm(asMakeInstr1(akNeg,
                 asMakeAdrReg(eax),
                 optimizenr,"optimized mul 2.op er -1"));
  else if (imm == 2)
    insertAsmElm(asMakeInstr2(akAdd,
                 asMakeAdrReg(eax),
                 asMakeAdrReg(eax),
                 optimizenr,"optimized mul 2.op er 2"));
  else if (imm == -2) {
    insertAsmElm(asMakeInstr2(akAdd,
                 asMakeAdrReg(eax),
                 asMakeAdrReg(eax),
                 optimizenr,"optimized mul 2.op er -2"));
    insertAsmElm(asMakeInstr1(akNeg,
                 asMakeAdrReg(eax),
                 optimizenr,"optimized mul -2"));
  }
  ++antalSlettedeMul;
  deleteInstr (focusInstr);
  deleteInstr (elm2);

  if (optimizeLog)
    fprintf(msgFile, "Mul const 2.op optimize: %d\r\n",optimizenr--);

  focusNextInstr(); /* flyt til nyindsatte*/
  optimer = true;
}

void peepHoleOptimizeMulConst1() /* 1. operant constant */
{
  asElm *elm2;
  asElm *elm3;
  int imm;
  if (focusInstr->slettet) return;
  /* mønster: (regx != eax og adr != eax)
       movl 0,eax
       movl adr,regx
       imul regx
     reduceres til:
       xor eax,eax
     mønster: (regx != eax og adr != eax)
       movl 1,eax
       movl adr,regx
       imul regx
     reduceres til:
       movl adr,eax
     mønster: (regx != eax og adr != eax)
       movl -1,eax
       movl adr,regx
```

```
      imul regx
    reduceres til:
      movl adr,eax
      negl eax
    mønster: (regx != eax og adr != eax)
      movl 2,eax
      movl adr,regx
      imul regx
    reduceres til:
      movl adr,eax
      movl eax,eax
    mønster: (regx != eax og adr != eax)
      movl -2,eax
      movl adr,regx
      imul regx
    reduceres til:
      movl adr,eax
      movl eax,eax
      negl eax
*/
if (focusInstr->kind != akMove) return;
if (focusInstr->val.eInstr2.fAdr->kind != aaImm) return;
imm = focusInstr->val.eInstr2.fAdr->val.eImm.imm;
if (imm != 1 && imm != -1 && imm != 2 && imm != -2 && imm != 0) return;
if (focusInstr->val.eInstr2.tAdr->kind != aaReg) return;
if (focusInstr->val.eInstr2.tAdr->val.eReg.reg != eax) return;
elm2 = nextInstr(focusInstr);
if (elm2->kind != akMove) return;
if (elm2->val.eInstr2.fAdr->kind == aaReg)
  if (elm2->val.eInstr2.fAdr->val.eReg.reg == eax) return;
if (elm2->val.eInstr2.tAdr->kind == aaReg)
  if (elm2->val.eInstr2.tAdr->val.eReg.reg == eax) return;
elm3 = nextInstr(elm2);
if (elm3->kind != akMul) return;
if (!equalsAdr(elm2->val.eInstr2.tAdr,elm3->val.eInstr1.adr)) return;
if (imm == 0)
  insertAsmElm(asMakeInstr2(akXor,
               asMakeAdrReg(eax),
               asMakeAdrReg(eax),
               optimizenr,"optimized mul 1.op er 0"));
else if (imm == 1)
  insertAsmElm(asMakeInstr2(akMove,
               elm2->val.eInstr2.fAdr,  /*NB samme adr. elm !!*/
               asMakeAdrReg(eax),
               optimizenr,"optimized mul 1.op er 1"));
else if (imm == -1) {
  insertAsmElm(asMakeInstr2(akMove,
               elm2->val.eInstr2.fAdr,  /*NB samme adr. elm !!*/
               asMakeAdrReg(eax),
               optimizenr,"optimized mul 1.op er -1"));
  insertAsmElm(asMakeInstr1(akNeg,
               asMakeAdrReg(eax),
               0,"optimized mul -1"));
}
else if (imm == 2) {
  insertAsmElm(asMakeInstr2(akMove,
               elm2->val.eInstr2.fAdr,  /*NB samme adr. elm !!*/
               asMakeAdrReg(eax),
               optimizenr,"optimized mul 1.op er 2"));
  insertAsmElm(asMakeInstr2(akAdd,
               asMakeAdrReg(eax),
               asMakeAdrReg(eax),
               optimizenr,"optimized mul 1.op er 2"));
```

```
  }
  else if (imm == -2) {
    insertAsmElm(asMakeInstr2(akMove,
                elm2->val.eInstr2.fAdr,  /*NB samme adr. elm !!*/
                asMakeAdrReg(eax),
                optimizenr,"optimized mul 1.op er -2"));
    insertAsmElm(asMakeInstr2(akAdd,
                asMakeAdrReg(eax),
                asMakeAdrReg(eax),
                optimizenr,"optimized mul 1.op er -2"));
    insertAsmElm(asMakeInstr1(akNeg,
                asMakeAdrReg(eax),
                optimizenr,"optimized mul 1.op er -2"));
  }
  ++antalSlettedeMul;
  deleteInstr (focusInstr);
  deleteInstr (elm2);
  deleteInstr (elm3);

  if (optimizeLog)
    fprintf(msgFile, "Mul const 1.op optimize: %d\r\n",optimizenr--);

  focusNextInstr(); /* flyt til nyindsatte*/
  optimer = true;
}

void peepHoleOptimizeJumpTwize()
{
  asElm *elm2;
  if (focusInstr->slettet) return;
  /* mønster: (lab1 != lab2)
       jxx1 lab1
       lab1:
       jxx2 lab2
     reduceres til:
       jxx1 lab2
       lab1:
       jxx2 lab2
  */
  if (focusInstr->kind != akJump) return;
  elm2 = nextInstr(focusInstr->val.eJump.label);
  while (elm2->kind == akLabel)  /* find 1. instr. efter label*/
    elm2 = nextInstr(elm2);
  if (elm2->kind != akJump) return;
  if (focusInstr->val.eJump.label == elm2->val.eJump.label) return;
  moveToFocusInstr();
  insertAsmElm(asMakeJump(focusInstr->val.eJump.jump,
              elm2->val.eJump.label,
              optimizenr,"optimized jump twize"));

  deleteInstr (focusInstr);
  ++antalSlettedeJump;

  if (optimizeLog)
    fprintf(msgFile, "Jump twize optimize: %d\r\n",optimizenr--);

  focusNextInstr(); /* flyt til nyindsatte*/
  optimer = true;
}

void peepHoleOptimizeIncDecFollow()
{
  asElm *elm2;
```

```
  asElm *elm3;
  int const3;
  if (focusInstr->slettet) return;
  /* mønster:
       incl adr
       incl adr
       incl adr
     reduceres til:
       addl 3,adr
    mønster:
       decl adr
       decl adr
       decl adr
     reduceres til:
       addl -3,adr
  */
  if (focusInstr->kind != akInc ||focusInstr->kind != akInc) return;
  elm2 = nextInstr(focusInstr);
  if (elm2->kind != focusInstr->kind) return;
  if (!equalsAdr(focusInstr->val.eInstr1.adr,elm2->val.eInstr1.adr))
    return;
  elm3 = nextInstr(elm2);
  if (elm3->kind != focusInstr->kind) return;
  if (focusInstr->kind == akInc) const3 = 3;
  else                           const3 = -3;
  insertAsmElm(asMakeInstr2(akAdd,
               asMakeAdrImm(const3),
               asMakeAdrReg(eax),
               optimizenr,"optimized 3 gange inc/dec"));
  deleteInstr (focusInstr);
  deleteInstr (elm2);
  deleteInstr (elm3);

  focusNextInstr(); /* flyt til nyindsatte*/
  optimer = true;
  if (optimizeLog)
    fprintf(msgFile, "inc/dec optimize: %d\r\n",optimizenr--);
}

void peepHoleOptimizeTwizeCmpOrAnd()
{
  asElm *elm2;
  if (focusInstr->slettet) return;
  /* mønster: op2 (cmpl, orl, andl, xorl)
       op2 adr1,adr2
       op2 adr1,adr2
     reduceres til:
       op2 adr1,adr2
  */
  if (focusInstr->kind != akCmp &&
      focusInstr->kind != akOr && focusInstr->kind != akAnd)
   return;
  elm2 = nextInstr(focusInstr);
  if (elm2->kind != focusInstr->kind) return;
  if (!equalsAdr(focusInstr->val.eInstr2.fAdr,elm2->val.eInstr2.fAdr))
    return;
  if (!equalsAdr(focusInstr->val.eInstr2.tAdr,elm2->val.eInstr2.tAdr))
    return;
  deleteInstr (focusInstr);

  if (optimizeLog)
    fprintf(msgFile, "Cmp/Or/And twize optimize: %d\r\n",optimizenr--);
```

```
  focusNextInstr(); /* flyt til næste ikke slettede*/
  optimer = true;
}

void peepHoleOptimizeContra()
{
  asElm *elm2;
  if (focusInstr->slettet) return;
  /* mønster:
       negl adr
       negl adr
     eller
       incl adr
       decl adr
     eller
       decl adr
       incl adr
     eller
       pushl adr
       popl  adr
     reduceres til:
       ingenting   (NB: flag bruges ikke)
  */
  if (focusInstr->kind == akNeg) {
    elm2 = nextInstr(focusInstr);
    if (elm2->kind != akNeg) return;
    if (!equalsAdr(focusInstr->val.eInstr1.adr,elm2->val.eInstr1.adr))
      return;
  }
  else if (focusInstr->kind == akInc) {
    elm2 = nextInstr(focusInstr);
    if (elm2->kind != akDec) return;
    if (!equalsAdr(focusInstr->val.eInstr1.adr,elm2->val.eInstr1.adr))
      return;
  }
  else if (focusInstr->kind == akDec) {
    elm2 = nextInstr(focusInstr);
    if (elm2->kind != akInc) return;
    if (!equalsAdr(focusInstr->val.eInstr1.adr,elm2->val.eInstr1.adr))
      return;
  }
  else if (focusInstr->kind == akPush) {
    elm2 = nextInstr(focusInstr);
    if (elm2->kind != akPop) return;
    if (!equalsAdr(focusInstr->val.eInstr1.adr,elm2->val.eInstr1.adr))
      return;
  }
  else return;
  deleteInstr (focusInstr);
  deleteInstr (elm2);

  if (optimizeLog)
    fprintf(msgFile, "Contra optimize: %d\r\n",optimizenr--);

  focusNextInstr(); /* flyt til næste ikke slettede*/
  optimer = true;
}

void peepHoleOptimize ()
{
  asElm *elm;
  int antLoops = 0;
  optimer = true;
```

```
  antalSlettede = 0;
  antalSlettedeMul = 0;

  if (!optimizeLog)optimizenr = 0;

  fprintf(msgFile, "TONY peep-hole optimimering start\r\n");
  while (optimer) {
    ++antLoops;
    optimer = false;
    focusInstr = asmListeHead;
    elm = focusNextInstr();
    while (elm != asmListeTail)
    {
      peepHoleOptimizeContra();
      peepHoleOptimizeAddConst1();
      peepHoleOptimizeAddSub2();
      peepHoleOptimizeCompare();
      peepHoleOptimizeAddSubConstTwize();
      peepHoleOptimizeStoreLoad();
      peepHoleOptimizeMoveTwize();
      peepHoleOptimizeTwizeCmpOrAnd();
      peepHoleOptimizeLoadLoadImm();
      peepHoleOptimizeMulConst2();
      peepHoleOptimizeMulConst1();
      peepHoleOptimizeJumpTwize();
      peepHoleOptimizeIncDecFollow();
      elm = focusNextInstr();
    }
  }
  fprintf(msgFile,"TONY peep-hole optimimering slut:\r\n");
  fprintf(msgFile,"    %d gennemloeb og reduceret med:\r\n", antLoops);
  fprintf(msgFile,"    %d instruktioner\r\n", antalSlettede);
  fprintf(msgFile,"    %d multiplikationer\r\n", antalSlettedeMul);
  fprintf(msgFile,"    %d jumps\r\n", antalSlettedeJump);
}
```

Emit fasen

Funktioner til gennerering af assemblerkode ud fra abstakt assembler

tonyAbstractAsmPrint.c

```
/* ------------------------------------------------------------------
  Realisering af funktioner til kodegennerering
  Programmør: Bjørk Busch
  Historik:   2005.04.30
  Historik:   2005.05.10  kosmetik i source
------------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include "bjbu_std.h"
#include "symbol.h"
#include "tonyAbstractAsm.h"

extern FILE *msgFile;
/* head på abstract assembler liste */
extern struct asElm *asmListeHead;
/* tail på abstract assembler liste */
extern struct asElm *asmListeTail;

extern bool printComment;

#define printAsmFile_FILENAME "stdout"
FILE *printAsmFile;
extern FILE *msgFile;

void openprintAsmFileTxt();


/*-----------------------------------------------------------------
Funktioner for udskrift af Abstract assembler-Strukturer
-----------------------------------------------------------------*/
char *getRegName (int reg)
{
  switch (reg)
  {
    case nreg:      return "???";
    case eax:       return "%eax";
    case ebx:       return "%ebx";
    case ecx:       return "%ecx";
    case edx:       return "%edx";
    case esi:       return "%esi";
    case edi:       return "%edi";
    case ebp:       return "%ebp";
    case esp:       return "%esp";
    default:
            fprintf(msgFile, "Fejl: registerdef. mangler\r\n");
            return "?reg?";
  }
}
char *getInstrName (asElm *e)
{
  switch (e->kind)
  {
    case akPush:    return "pushl";
    case akPop:     return "popl";
    case akMul:     return "imull";
    case akDiv:     return "idivl";
```

```
    case akInc:     return "incl";
    case akDec:     return "decl";
    case akNeg:     return "negl";

    case akMove:    return "movl";
    case akLoadAdr: return "leal";
    case akAdd:     return "addl";
    case akSub:     return "subl";
    case akAnd:     return "andl";
    case akOr:      return "orl";
    case akXor:     return "xorl";
    case akCmp:     return "cmpl";

    case akJump:
          switch (e->val.eJump.jump)
          {
            case akJmp:  return "jmp";
            case akJe:   return "je";
            case akJne:  return "jne";
            case akJl:   return "jl";
            case akJg:   return "jg";
            case akJle:  return "jle";
            case akJge:  return "jge";
            case akJnl:  return "jnl";
            case akJng:  return "jng";
          }
    case akCall:    return "call";
    case akReturn:  return "ret";
    case akCltd:    return "cltd";
    default:
            fprintf(msgFile, "Fejl: Instruktion def. mangler\r\n");
            return "?instr?";
  }
}
/*-----------------------------------------------------------------
Funktioner for udskrift af Abstract assembler-Strukturer adresserdel
-----------------------------------------------------------------*/
char *printAdrReg (asAdr *a)
{
  fprintf(printAsmFile,"%s",getRegName(a->val.eReg.reg));
}
char *printAdrMem (asAdr *a)
{
  asElm *o = a->val.eMem.offset;
  if (o != NULL) {
    if (o->val.eOffset.name[0] != 0) {
      fprintf(printAsmFile,"%s",o->val.eOffset.name);
      if (o->val.eOffset.nr >= 0)
        fprintf(printAsmFile,"_%d",o->val.eOffset.nr);
    }
    else
      fprintf(printAsmFile,"%d",o->val.eOffset.offset);
  }
  else
    fprintf(printAsmFile,"%d",a->val.eMem.offset);

  fprintf(printAsmFile,"(");
  if (a->val.eMem.disp <= nreg )
    fprintf(printAsmFile,"%s",getRegName(a->val.eMem.base));
  else {
    fprintf(printAsmFile,"%s,",getRegName(a->val.eMem.base));
    if ( a->val.eMem.faktor > 1)
      fprintf(printAsmFile,"%d,",a->val.eMem.faktor);
```

```
    fprintf(printAsmFile,"%s",getRegName(a->val.eMem.disp));
  }
  fprintf(printAsmFile,")");
}
char *printAdrMemEx (asAdr *a)
{
  fprintf(printAsmFile,"%d(",a->val.eMemEx.offset);
  if (a->val.eMemEx.disp <= nreg)
    fprintf(printAsmFile,"%s",getRegName(a->val.eMemEx.base));
  else {
    fprintf(printAsmFile,"%s,",getRegName(a->val.eMemEx.base));
    if ( a->val.eMemEx.faktor > 1)
      fprintf(printAsmFile,"%d,",a->val.eMemEx.faktor);
    fprintf(printAsmFile,"%s",getRegName(a->val.eMemEx.disp));
  }
  fprintf(printAsmFile,")");
}
char *printAdrImm (asAdr *a)
{
  fprintf(printAsmFile,"$%d",a->val.eImm.imm);
}
char *printAdrLabelValue (asAdr *a)
{
  asElm *l = a->val.eLabel.label;
  fprintf(printAsmFile,"%s",l->val.eLabel.name);
  if (l->val.eLabel.nr >= 0)
    fprintf(printAsmFile,"_%d",l->val.eLabel.nr);
  if (l->val.eLabel.suffix[0] != 0)
    fprintf(printAsmFile,"_%s",l->val.eLabel.suffix);
}
char *printAdrLabelAdr (asAdr *a)
{
  asElm *l = a->val.eLabel.label;
  fprintf(printAsmFile,"$%s",l->val.eLabel.name);
  if (l->val.eLabel.nr >= 0)
    fprintf(printAsmFile,"_%d",l->val.eLabel.nr);
  if (l->val.eLabel.suffix[0] != 0)
    fprintf(printAsmFile,"_%s",l->val.eLabel.suffix);
}
char *printAdr (asAdr *a)
{
  switch (a->kind)
  {
    case aaReg:        printAdrReg(a);        break;
    case aaMem:        printAdrMem(a);        break;
    case aaMemEx:      printAdrMemEx(a);      break;
    case aaImm:        printAdrImm(a);        break;
    case aaLabelValue: printAdrLabelValue(a); break;
    case aaLabelAdr:   printAdrLabelAdr(a);   break;
  }
}
/*-------------------------------------------------------------
Funktioner for udskrift af Abstract assembler-Strukturer elm
-------------------------------------------------------------*/
void printAsmComment (asElm *e)
{
  if (printComment) {
    if (e->comment != NULL) if (e->comment[0] != 0) {
      if (e->kind != akComment)
        fprintf(printAsmFile,"  ");
      if (e->linienr > 0)
        fprintf(printAsmFile,"#%d:%s",e->linienr,e->comment);
      else
```

```
        fprintf(printAsmFile,"#%s",e->comment);
    }
  }
  fprintf(printAsmFile,"\n");
}
void printAsmLabel (asElm *e)
{
  if (e->val.eLabel.nr < 0)
    fprintf(printAsmFile,"%s:",e->val.eLabel.name);
  else if (e->val.eLabel.suffix[0] == 0)
    fprintf(printAsmFile,"%s_%d:",e->val.eLabel.name,e->val.eLabel.nr);
  else
    fprintf(printAsmFile,"%s_%d_%s:",e->val.eLabel.name,
            e->val.eLabel.nr, e->val.eLabel.suffix);
  printAsmComment (e);
}

void printAsmDef (asElm *e)
{
  fprintf(printAsmFile,"%s",e->val.eDef.def);
  printAsmComment (e);
}
void printAsmOffset (asElm *e)
{
  if (e->val.eOffset.nr < 0)
    fprintf(printAsmFile,".equ  %s,%d",e->val.eOffset.name,
                    e->val.eOffset.offset);
  else
    fprintf(printAsmFile,".equ  %s_%d,%d",e->val.eOffset.name,
                    e->val.eOffset.nr, e->val.eOffset.offset);
  printAsmComment (e);
}
void printAsmInstr1 (asElm *e)
{
  fprintf(printAsmFile,"  %s  ",getInstrName(e));
  printAdr (e->val.eInstr1.adr);
  printAsmComment (e);
}
void printAsmInstr2 (asElm *e)
{
  fprintf(printAsmFile,"  %s  ",getInstrName(e));
  printAdr (e->val.eInstr2.fAdr);
  fprintf(printAsmFile,",");
  printAdr (e->val.eInstr2.tAdr);
  printAsmComment (e);
}
void printAsmJump (asElm *e)
{
  fprintf(printAsmFile,"  %s  %s",getInstrName(e),
          e->val.eJump.label->val.eLabel.name);
  if (e->val.eJump.label->val.eLabel.nr >= 0)
          fprintf(printAsmFile,"_%d",e->val.eJump.label->val.eLabel.nr);
  if (e->val.eJump.label->val.eLabel.suffix[0] != 0)
          fprintf(printAsmFile,"_%s",e->val.eJump.label->val.eLabel.suffix);
  printAsmComment (e);
}
void printAsmCall (asElm *e)
{
  fprintf(printAsmFile,"  %s  %s",getInstrName(e),
          e->val.eCall.label->val.eLabel.name);
  if (e->val.eCall.label->val.eLabel.nr >= 0)
          fprintf(printAsmFile,"_%d",e->val.eCall.label->val.eLabel.nr);
  if (e->val.eCall.label->val.eLabel.suffix[0] != 0)
```

```
          fprintf(printAsmFile,"_%s",e->val.eCall.label->val.eLabel.suffix);
  printAsmComment (e);
}
void printAsmReturn (asElm *e)
{
  fprintf(printAsmFile,"  %s",getInstrName(e));
  printAsmComment (e);
}
void printAsmCltd (asElm *e)
{
  fprintf(printAsmFile,"  %s",getInstrName(e));
  printAsmComment (e);
}

void printAsmElm (asElm *e)
{
  if (!printComment && e->slettet) return;
  if (e->slettet)
    fprintf(printAsmFile,"###");
  switch (e->kind)
  {
    case akComment: if (printComment) printAsmComment(e);  break;
    case akDef:     printAsmDef(e);                 break;
    case akOffset:  printAsmOffset(e);              break;
    case akLabel:   printAsmLabel(e);               break;

    case akPush:    printAsmInstr1(e);              break;
    case akPop:     printAsmInstr1(e);              break;
    case akMul:     printAsmInstr1(e);              break;
    case akDiv:     printAsmInstr1(e);              break;
    case akInc:     printAsmInstr1(e);              break;
    case akDec:     printAsmInstr1(e);              break;
    case akNeg:     printAsmInstr1(e);              break;

    case akMove:    printAsmInstr2(e);              break;
    case akLoadAdr: printAsmInstr2(e);              break;
    case akAdd:     printAsmInstr2(e);              break;
    case akSub:     printAsmInstr2(e);              break;
    case akAnd:     printAsmInstr2(e);              break;
    case akOr:      printAsmInstr2(e);              break;
    case akXor:     printAsmInstr2(e);              break;
    case akCmp:     printAsmInstr2(e);              break;

    case akJump:    printAsmJump(e);                break;
    case akCall:    printAsmCall(e);                break;
    case akReturn:  printAsmReturn(e);              break;
    case akCltd:    printAsmCltd(e);                break;
    default:
            fprintf(msgFile, "Fejl: Assembler element mangler def.\r\n");
  }
}


void openprintAsmFile()
{

  printAsmFile = stdout;
/*
  printAsmFile = fopen(printAsmFile_FILENAME,"w");
  if (printAsmFile==NULL){
    fprintf(msgFile, "alvorligFejl ved open af xmlfile <%s>\n",
    printAsmFile_FILENAME);
    exit(1);
```

```
  }
*/
}

void printAsmFileProgram ()
{
  int i = 0;
  asElm *e = asmListeHead;
  openprintAsmFile();
  fprintf(msgFile,
          "TONY assembler udskrives til <%s> start\n",
          printAsmFile_FILENAME);

  fprintf(printAsmFile,"# Oversat fra TONY\n# Compiler skrevet af %s\n\n",
      "Bjørk Boye Busch - bjbu@tietgen.dk");

  e = asmListeHead;
  while (e != NULL) {
    printAsmElm (e);
    e = e->next;
    ++i;
  }
  fprintf(msgFile,
          "TONY assembler udskrives til <%s> slut\n",
          printAsmFile_FILENAME);
  fclose(printAsmFile);
}

void printAsmFileProgramTestData ()
{
  int i;
  char* wtekst;
  asAdr *ad;
  asElm *a;
  asElm *labMainStartWin   = asMakeLabel("_main",-1,0,"entry for windows");
  asElm *labMainStartLinux = asMakeLabel("main",-1,0,"entry for linux");
  asElm *labMainEnd =     asMakeLabel("main_end",-1,0,"");
  asElm *labIntFormat =   asMakeLabel("_int_format",-1,0,"");
  asElm *labScopeAdr[100];        /* max. 100 scopes */

  fprintf(msgFile, "Test assembler-code\n");

  wtekst = (char*)malloc(100);
  asmListeHead = NULL;            /* head på abstract assembler liste */
  asmListeTail = NULL;            /* tail på abstract assembler liste */

  addAsmElm(asMakeComment(0,"Oversat fra TONY"));
  addAsmElm(asMakeComment(0,
           "Compiler skrevet af Bjørk Boye Busch - bjbu@tietgen.dk"));
  addAsmElm(asMakeDef(".data",0,""));
  addAsmElm(labIntFormat);
  addAsmElm(asMakeDef(".ascii \"%d\\n\\0\"",0,""));
  addAsmElm(labMainStartWin);


/*for (i=0;i<=s->maxScopeLevel;++i) { */
  for (i=0;i<=4;++i) {
    labScopeAdr[i] = asMakeLabel("_scopeadr",i,
                                 0,"frame-adr. til til scopelevel");
    addAsmElm(labScopeAdr[i]);
  }
```

```
  addAsmElm(asMakeInstr1(akPush,asMakeAdrReg(ebp),0,"save ebp"));
  addAsmElm(asMakeInstr2(akMove,asMakeAdrReg(esp),
            asMakeAdrReg(ebp),0,"set frame ptr"));

  a = asMakeOffset("tal",1,-20,0,"");
  addAsmElm(a);
  addAsmElm(asMakeInstr2(akAdd,
            asMakeAdrMem(a,ebp,0,0),asMakeAdrReg(eax),0,"Add"));
  addAsmElm(asMakeInstr2(akAdd,
            asMakeAdrMem(a,ebp,0,esi),asMakeAdrReg(ebx),0,"Add"));
  addAsmElm(asMakeInstr2(akAdd,
            asMakeAdrMem(a,ebp,2,esi),asMakeAdrReg(ecx),0,"Add"));
  addAsmElm(asMakeInstr2(akSub,
            asMakeAdrMemEx(-4,ebp,0,0),asMakeAdrReg(ebx),0,"Sub"));
  addAsmElm(asMakeInstr2(akSub,
            asMakeAdrImm(100),asMakeAdrMemEx(-4,ebp,0,0),0,"Sub"));

  addAsmElm(asMakeInstr2(akMove,
            asMakeAdrImm(100),asMakeAdrReg(eax),0,"Load"));
  addAsmElm(asMakeJump(akJmp,labMainEnd,0,"hop"));
  addAsmElm(asMakeJump(akJne,labMainEnd,0,"hop"));
  addAsmElm(asMakeCall(labMainEnd,0,"hop"));
  addAsmElm(asMakeReturn(0,"hop"));

}
```

Sammenkædning til compileren

Main programmet

main.c

```
/* ------------------------------------------------------------
  Realisering af hovedrutine til opbygning af TONY
  af abstract syntax tree, weed-fase, type-check og
  udskrift af AST med typer, samt symbol-tabeller med
  reference til hvor de er defineret og bruges
  Programmør: Bjørk Busch
  Historik:  2005.03.26
  Historik: 2005.05.10  Flags og kosmetik i source
------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include "bjbu_std.h"
#include "symbol.h"
#include "tonyAbstractAsm.h"
#include "tonyAST.h"
FILE *msgFile;

/*------------------------------------------------------------
Fælles data på tværs af moduler
------------------------------------------------------------*/
extern int alvorligFejl;
extern int fejl;
extern int advarsel;
extern sProgram *theProgram;
int tonyHeapSize = 10000000; /* Størrelse på heap */
bool restriktivCheck = false;
bool functionTopdown = true; /* styrer rækkefølge af main og funktioner */
bool printComment = true;
bool optimizeLog = false;
bool recordArray = true;     /* true->record array ikke ref. til record */

bool debugWrite = false;     /* debug facilitet*/
bool debugRegister = false;  /* debug facilitet*/
bool debugHeap = false;      /* debug facilitet*/
int  debugHeapSize = 50;    /* debug facilitet */
/*------------------------------------------------------------
Compiler options
------------------------------------------------------------*/
bool make_xref = false;
bool make_prettytxt = false;
bool make_prettyxml = false;
bool make_directasmcode = false;
bool make_abstasmcode = true;
bool make_abstasmcodeTest = false;
bool make_printAsm = true;
bool make_optimize = true;
char *msgFileName = NULL;


extern void yyparse();
extern void weedProgram (sProgram *s);
extern void symbolsBuildProgram (sProgram *s);
extern void symbolsCheckProgram (sProgram *s);
extern void symbolsXrefProgram (sProgram *s);
extern void prettyXmlProgram (sProgram *s);
extern void prettyTxtProgram (sProgram *s);
```

```
extern void codeOffsetProgram (sProgram *s);
extern void abstractAsmProgram (sProgram *s);
extern void printAsmFileProgram ();
extern void  printAsmFileProgramTestData ();
extern void peepHoleOptimize ();

void setFlags (int argc, char *argv[]) {
  int i = 1;

  while (i < argc) {
    if (strcmp(argv[i],"--help")==0){
    }
    else if (strcmp(argv[i],"-test")==0) {
      make_abstasmcodeTest = true;
      make_directasmcode = false;
      make_abstasmcode = false;
    }
    else if (strcmp(argv[i],"-restriktiv")==0) {
      restriktivCheck = true;
    }
    else if (strcmp(argv[i],"-heapSize")==0) {
      ++i;
      tonyHeapSize = atoi(argv[i]);
    }
    else if (strcmp(argv[i],"-xref")==0) {
      make_xref = true;
    }
    else if (strcmp(argv[i],"-prettyTxt")==0) {
      make_prettytxt = true;
    }
    else if (strcmp(argv[i],"-prettyXml")==0) {
      make_prettyxml = true;
    }
    else if (strcmp(argv[i],"-S")==0) {
      make_printAsm = false;
    }
    else if (strcmp(argv[i],"-direct")==0) {
      make_directasmcode = true;
      make_abstasmcode = false;
    }
    else if (strcmp(argv[i],"-msg")==0) {
      ++i;
      msgFileName = argv[i];
    }
    else if (strcmp(argv[i],"-noOptimize")==0) {
      make_optimize = false;
    }
    else if (strcmp(argv[i],"-optimizeLog")==0) {
      optimizeLog = true;
    }
    else if (strcmp(argv[i],"-noComment")==0) {
      printComment = false;
    }
    else if (strcmp(argv[i],"-noRecordArray")==0) {
      recordArray = false;
    }
    else if (strcmp(argv[i],"-debugWrite")==0) {
      ++i;
      debugWrite = true;      /* debug facilitet*/
      debugRegister = true;   /* debug facilitet*/
      debugHeap = true;       /* debug facilitet*/
      debugHeapSize = atoi(argv[i]);
    }
```

```
    else {
      fprintf(stderr, "Ukendt compiler diriktiv <%s>\r\n",argv[i]);
      exit(100);
    }
    ++i;
  }
}
void visFlags (int argc, char *argv[]) {
  int i = 1;
  if (argc > 1)
    fprintf(msgFile,"Options:\r\n");
  while (i < argc) {
    if (strcmp(argv[i],"--help")==0){
      fprintf(msgFile,"Usage: %s [option]\r\n",argv[0]);
      fprintf(msgFile,"Options:\r\n");
      fprintf(msgFile,"  --help         Vis denne information\r\n");
      fprintf(msgFile,"  -test          dan test assembler\r\n");
      fprintf(msgFile,"  -restriktiv    ristriktiv typecheck\r\n");
      fprintf(msgFile,"  -heapSize x    saet heapsize til x\r\n");
      fprintf(msgFile,"  -xref          udskriv x-ref\r\n");
      fprintf(msgFile,"  -prettyTxt     udskriv pretty-tekst fil\r\n");
      fprintf(msgFile,"  -prettyXml     udskriv pretty-xml fil\r\n");
      fprintf(msgFile,"  -S             oversaet kun - ingen assembler\r\n");
      fprintf(msgFile,"  -direct        oversaet direkte\r\n");
      fprintf(msgFile,"  -noComment     udskriv ikke kommentarer\r\n");
      fprintf(msgFile,"  -msg file      send meddelelser til file\r\n");
      fprintf(msgFile,"  -noOptimize    brug ikke optimizer\r\n");
      fprintf(msgFile,"  -optimizeLog   udskriv optimizer log\r\n");
      fprintf(msgFile,"  -debugWrite x  debug med heapsize x\r\n");
      fprintf(msgFile,"                 write null giver dump\r\n");
      fprintf(msgFile,"  -noRecordArray array uden record direkte\r\n");
      fprintf(msgFile,"                 istedet anvendes adresser og\r\n");
      fprintf(msgFile,"                 record skal allokeres med new\r\n");
    }
    else if (strcmp(argv[i],"-test")==0) {
      fprintf(msgFile, "-test er sat\r\n");
    }
    else if (strcmp(argv[i],"-restriktiv")==0) {
      fprintf(msgFile, "-restriktiv er sat\r\n");
    }
    else if (strcmp(argv[i],"-heapSize")==0) {
      ++i;
      fprintf(msgFile, "-heapSize sat til %d\r\n",tonyHeapSize);
    }
    else if (strcmp(argv[i],"-xref")==0) {
      fprintf(msgFile, "-xref er sat\r\n");
    }
    else if (strcmp(argv[i],"-prettyTxt")==0) {
      fprintf(msgFile, "-prettyTxt er sat\r\n");
    }
    else if (strcmp(argv[i],"-prettyXml")==0) {
      fprintf(msgFile, "-prettyXml er sat\r\n");
    }
    else if (strcmp(argv[i],"-S")==0) {
      fprintf(msgFile, "-S er sat\r\n");
    }
    else if (strcmp(argv[i],"-direct")==0) {
      fprintf(msgFile, "-direct er sat\r\n");
    }
    else if (strcmp(argv[i],"-msg")==0) {
      ++i;
      fprintf(msgFile, "-msg sat til %s\r\n",msgFileName);
    }
```

```
    else if (strcmp(argv[i],"-noOptimize")==0) {
      fprintf(msgFile, "-noOptimize er sat\r\n");
    }
    else if (strcmp(argv[i],"-noComment")==0) {
      fprintf(msgFile, "-noComment er sat\r\n");
    }
    else if (strcmp(argv[i],"-optimizeLog")==0) {
      fprintf(msgFile, "-optimizeLog er sat\r\n");
    }
    else if (strcmp(argv[i],"-noRecordArray")==0) {
      fprintf(msgFile, "-noRecordArray er sat\r\n");
    }
    else if (strcmp(argv[i],"-debugWrite")==0) {
      ++i;
      fprintf(msgFile, "-debugWrite sat til med heapsize %d\r\n",
              debugHeapSize);
    }
    else{
      fprintf(msgFile, "%s\r\n",argv[i]);
    }
    ++i;
  }
}

int main (int argc, char *argv[]) {
  setFlags(argc, argv);
  if (msgFileName != NULL) {
    msgFile = fopen(msgFileName,"w");
    if (msgFile==NULL){
      fprintf(stderr,"alvorligFejl ved open af msgFile <%s>\n",msgFileName);
      exit(100);
    }
    fprintf(stderr,"Meddelelser udskrives til <%s>\n",msgFileName);
  }
  else
    msgFile = stderr;
  visFlags(argc, argv);

  yyparse();
  if (alvorligFejl+fejl == 0)
    fprintf(msgFile, "TONY Parser afsluttet normal\r\n");
  else
    fprintf(msgFile, "TONY Parser afsluttet med %d alvorligFejl\r\n",
            alvorligFejl);

  if (theProgram != NULL)  weedProgram (theProgram);
  if (theProgram != NULL)  symbolsBuildProgram (theProgram);
  if (theProgram != NULL)  symbolsCheckProgram (theProgram);

  if (theProgram != NULL && make_xref)       symbolsXrefProgram (theProgram);
  if (theProgram != NULL && make_prettyxml)  prettyXmlProgram(theProgram);
  if (theProgram != NULL && make_prettytxt)  prettyTxtProgram(theProgram);

  if (alvorligFejl+fejl==0) {
    if (theProgram != NULL && make_printAsm) {
      codeOffsetProgram(theProgram);
      if (make_directasmcode)
        codeAsmProgram(theProgram);
      else if (make_abstasmcode) {
        abstractAsmProgram (theProgram);
        if (make_optimize)
          peepHoleOptimize ();
        printAsmFileProgram ();
```

```
      }
    }

  }
  fprintf(msgFile, "\r\n");
  fprintf(msgFile, "Antal alvorligFejl: %d\r\n",alvorligFejl);
  fprintf(msgFile, "Antal fejl: %d\r\n",fejl);
  fprintf(msgFile, "Antal advarsler: %d\r\n",advarsel);

  close(msgFile);
  if (alvorligFejl+fejl>0) {
    return 1;
  }

  return 0;
}
```

Utility funktioner

bjbu_std.h

```
/* ------------------------------------------------------------------
  Definition af diverse gennerelle hjælperutiner og macroer
  Programmør: Bjørk Busch
  Historik:  2005.03.08  Udgave til rapport 2 færdig
  Historik: 2005.05.10  kosmetik i source
------------------------------------------------------------------*/

/* oprindelig pladseret i symbol.h */
#define NEW(type) (type *)malloc(sizeof(type))
void *malloc(unsigned n);

/* bool type tilføjet */
#define bool int
#define false 0
#define true 1

/* size: antal tegn excl. terminator */
char* NewString(unsigned size);

/* str: streng med initieringsværdi */
char* NewStringCopy(char* str);

/* str: streng med initieringsværdi */
/* minSize: mindste antal tegn excl. terminator */
/* returnerer streng med indhold fra str og plads til mindst minSize tegn */
char* NewStringCopySized(char* str, unsigned minSize);

/* str1: foreste streng */
/* str2: bagerste streng */
char* NewStringCat(char* str1, char* str2);
```

bjbu_std.c

```
/* ------------------------------------------------------------------
  Realisering af diverse gennerelle hjælperutiner
  Programmør: Bjørk Busch
  Historik:  2005.03.08  Udgave til rapport 2 færdig
  Historik: 2005.05.10  kosmetik i source
------------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
```

```
#include "bjbu_std.h"

char* NewString(unsigned size) {
  char* nyStr;
  nyStr = (char*)malloc(size+1);
  nyStr[0] = 0;
  return nyStr;
}
char* NewStringCopy(char* str) {
  char* nyStr;
  nyStr = (char*)malloc(strlen(str)+1);
  strcpy(nyStr,str);
  return nyStr;
}
char* NewStringCopySized(char* str, unsigned minSize) {
  char* nyStr;
  if (minSize < strlen(str))
    nyStr = (char*)malloc(strlen(str)+1);
  else
    nyStr = (char*)malloc(minSize+1);
  strcpy(nyStr,str);
  return nyStr;
}

char* NewStringCat(char* str1, char* str2) {
  char* nyStr;
  nyStr = (char*)malloc(strlen(str1)+strlen(str2)+1);
  strcpy(nyStr,str1);
  strcat(nyStr,str2);
  return nyStr;
}
```

Make file for build af compileren

p4_Build.make

```
CC      = gcc
RM      = rm -f
CCFLAGS   = -ansi -pedantic -c
LINK    = gcc

.PHONY:   all
all:    build

.PHONY:   clean
clean:
        $(RM) *.o
        $(RM) tony
        $(RM) lex.yy.c
        $(RM) tony.tab.*

.PHONY:   build
build:    clean P3

P3:       main.o bjbu_std.o tonyAST_Make.o tonyCodeOffset.o tonyCodeAsm.o \
tonyAbstractAsm.o tonyAbstractAsmPrint.o tonyAbstractAsmCode.o \
tonyAbstractAsmPeepHole.o tonyAST_Pretty_XML.o tonyAST_Pretty_TXT.o \
tonyWeed.o tonySymbols_Check.o tonySymbols_Build.o tonySymbols_Xref.o \
symbol.o lex.yy.o tony.tab.o
        $(LINK) *.o -o tony

main.o:   main.c bjbu_std.h symbol.h tonyAbstractAsm.h tonyAST.h
```

```
	$(CC) $(CCFLAGS) main.c -o main.o

bjbu_std.o: bjbu_std.c bjbu_std.h
	$(CC) $(CCFLAGS) bjbu_std.c

tonyAST_Make.o:  tonyAST_Make.c bjbu_std.h tonyAST.h tonyAST_Make.h symbol.h
	$(CC) $(CCFLAGS) tonyAST_Make.c

tonyAST_Pretty_XML.o: tonyAST_Pretty_XML.c  bjbu_std.h tonyAST.h symbol.h \
tonyAST_Pretty_XML.h
	$(CC) $(CCFLAGS) tonyAST_Pretty_XML.c

tonyAST_Pretty_TXT.o: tonyAST_Pretty_TXT.c bjbu_std.h tonyAST.h symbol.h \
tonyAST_Pretty_TXT.h
	$(CC) $(CCFLAGS) tonyAST_Pretty_TXT.c

tonyCodeOffset.o: tonyCodeOffset.c bjbu_std.h tonyAST.h symbol.h
	$(CC) $(CCFLAGS) tonyCodeOffset.c

tonyCodeAsm.o: tonyCodeAsm.c bjbu_std.h tonyAST.h symbol.h
	$(CC) $(CCFLAGS) tonyCodeAsm.c

tonyAbstractAsm.o: tonyAbstractAsm.c tonyAbstractAsm.h bjbu_std.h tonyAST.h \
symbol.h
	$(CC) $(CCFLAGS) tonyAbstractAsm.c

tonyAbstractAsmCode.o: tonyAbstractAsmCode.c tonyAbstractAsm.h bjbu_std.h \
tonyAST.h symbol.h
	$(CC) $(CCFLAGS) tonyAbstractAsmCode.c

tonyAbstractAsmPrint.o: tonyAbstractAsmPrint.c tonyAbstractAsm.h bjbu_std.h \
tonyAST.h symbol.h
	$(CC) $(CCFLAGS) tonyAbstractAsmPrint.c

tonyAbstractAsmPeepHole.o: tonyAbstractAsmPeepHole.c tonyAbstractAsm.h \
bjbu_std.h symbol.h
	$(CC) $(CCFLAGS) tonyAbstractAsmPeepHole.c


tonySymbols_Build.o: tonySymbols_Build.c bjbu_std.h symbol.h tonyAST.h
	$(CC) $(CCFLAGS)  tonySymbols_Build.c

tonySymbols_Check.o: tonySymbols_Check.c bjbu_std.h symbol.h tonyAST.h
	$(CC) $(CCFLAGS)  tonySymbols_Check.c

tonySymbols_Xref.o: tonySymbols_Xref.c bjbu_std.h symbol.h tonyAST.h
	$(CC) $(CCFLAGS)  tonySymbols_Xref.c

tonyWeed.o: tonyWeed.c bjbu_std.h tonyAST.h
	$(CC) $(CCFLAGS)  tonyWeed.c

symbol.o: symbol.c bjbu_std.h symbol.h
	$(CC) $(CCFLAGS) symbol.c

lex.yy.o: lex.yy.c bjbu_std.h tony.tab.h
	$(CC) $(CCFLAGS) lex.yy.c

tony.tab.o: tony.tab.c tony.tab.hbjbu_std.h
	$(CC) $(CCFLAGS) tony.tab.c

tony.tab.h: tony.y bjbu_std.h
	bison  --verbose --defines tony.y
```

```
tony.tab.c: tony.y bjbu_std.h
	bison  --verbose --defines tony.y

lex.yy.c: tony.lex tony.tab.h bjbu_std.h
	flex tony.lex
```

Aftestning

Hjælpeværktøjer

Afvikling af windows bat-filer på linux

winBat2linuxCmd.lex

```
%option noyywrap
%{
/* ------------------------------------------------------------------------
   Konvertering af windows bat-filer til linuxformat
   Programmør: Bjørk Busch
   Historik: 2005.03.09
------------------------------------------------------------------------*/
#include <stdio.h>

int i;

%}

%x PARAMETER

%%
"\r\n"         {printf("\n");} /* Windows */
"\n"         {printf("\n");} /* linux */
[ \t]+         {printf("\t");} /* ignorer */

[rR][eE][mM][^\n]+"\n"       {}/* ignorer REM */
[pP][aA][uU][sS][eE][^\n]+"\n" {}/* ignorer PAUSE */

[tT][yY][pP][eE][ \t]{  /* TYPE */
                 printf("cat ");
                 BEGIN(PARAMETER);
               }
[rR][eE][nN][ \t]  {  /* REN */
              printf("mv ");
              BEGIN(PARAMETER);
            }
[cC][oO][pP][yY][ \t]{  /* COPY */
              printf("cp ");
              BEGIN(PARAMETER);
            }
[dD][eE][lL][ \t]  {  /* win del */
              printf("rm -f ");
              BEGIN(PARAMETER);
            }
[cC][dD][.]      {  /* cd.  indsæt blank */
              printf("cd .");
              BEGIN(PARAMETER);
            }
[a-zA-Z_][a-zA-Z0-9_\\]*[ \t]  {  /* andre komandoer */
                     for (i=0; yytext[i] != 0;++i)
                     {
                       if (yytext[i] == '\\')
                          yytext[i] == '/';
                     }
                     printf(yytext);
                     printf(" ");
                     BEGIN(PARAMETER);
                   }
```

```
<PARAMETER>"\r\n"  {  /* windows */
              printf("\n");
              BEGIN(INITIAL);
            }
<PARAMETER>"\n"    {  /* linux */
              printf("\n");
              BEGIN(INITIAL);
            }
<PARAMETER>"\\"    {  /* windows */
              printf("/");
            }
<PARAMETER>"%"   {  /* windows */
              printf("$");
            }
%%
int main () {
  yylex();
  return 0;
}
```

runbat

```
rm  -f tmp.cmd
winBat2linuxCmd.exe  < $1 > tmp.cmd
chmod 700 tmp.cmd
tmp.cmd  $2 $3 $4 $5 $6 $7 $8 $9
```

Påsætning at linienr på tekstfiler

printLnrCmd.lex

```
%option noyywrap
%{
/* ------------------------------------------------------------------------
   påsætning at linienr på tekstfiler
   Programmør: Bjørk Busch
   Historik: 2005.03.27
------------------------------------------------------------------------*/
#include <stdio.h>

int lnr = 1;
int i;
int t;

%}

%%

[^\n]+"\n"         {
                 printf("%d", lnr/100);
                 printf("%d", (lnr%100)/10);
                 printf("%d", (lnr%10));
                 for (i=0;yytext[i]!=0; ++i) {
                    if (yytext[i] == '\r' || yytext[i] == '\n')
                       yytext[i] = 0;
                 }
                 printf(": %s\n",yytext);
                 ++lnr;
               }
"\n"           {
```

```
                printf("%d", lnr/100);
                printf("%d", (lnr%100)/10);
                printf("%d", (lnr%10));
                for (i=0;yytext[i]!=0; ++i) {
                  if (yytext[i] == '\r' || yytext[i] == '\n')
                    yytext[i] = 0;
                }
                printf(": %s\n",yytext);
                ++lnr;
              }
.             {
                printf("%d", lnr/100);
                printf("%d", (lnr%100)/10);
                printf("%d", (lnr%10));
                for (i=0;yytext[i]!=0; ++i) {
                  if (yytext[i] == '\r' || yytext[i] == '\n')
                    yytext[i] = 0;
                }
                printf(": %s\n",yytext);
                ++lnr;
              }
[^\n]+      {
              printf("%d", lnr/100);
              printf("%d", (lnr%100)/10);
              printf("%d", (lnr%10));
              for (i=0;yytext[i]!=0; ++i) {
                if (yytext[i] == '\r' || yytext[i] == '\n')
                  yytext[i] = 0;
              }
              printf(": %s\n",yytext);
              ++lnr;
            }%%
int main () {
  yylex();
  return 0;

}
```

Gennering af samlet testkørsel

test_p4.make

```
CC     = gcc
RM     = rm -f
CCFLAGS  = -ansi -pedantic -c
LINK   = gcc
TCHECK = -S -restriktiv -prettyTxt -xref
TCODE =

.PHONY:  all
all:   code tonycode sducode com decl stm preced term weed stmtype \
exptype scopetype expand

.PHONY:  code
code:    clean tonycode

.PHONY:  sdu
sdu:   clean sducode

.PHONY:  check
check:   clean com decl stm preced term weed stmtype exptype scopetype \
expand
```

```
.PHONY:  weed_type_check
weed_type_check: weed stmtype exptype scopetype

.PHONY:  syntax
syntax:  com decl stm preced term weed

.PHONY:  clean
clean:
   $(RM) test_*.txt
   $(RM) test_*.xml
   $(RM) test_*.*.txt
   $(RM) test_*.*.xml
   $(RM) test_*.s
   $(RM) test_*.o
   $(RM) test_*.exe
   $(RM) test_total.txt

.PHONY:  tonycode
tonycode:
   runbat code_tony.bat test_code_exp1 test_total $(TCODE)
   runbat code_tony.bat test_code_exp2 test_total $(TCODE)
   runbat code_tony.bat test_code_assign1 test_total $(TCODE)
   runbat code_tony.bat test_code_suker test_total $(TCODE)
   runbat code_tony.bat test_code_if test_total $(TCODE)
   runbat code_tony.bat test_code_while1 test_total $(TCODE)
   runbat code_tony.bat test_code_while2 test_total $(TCODE)
   runbat code_tony.bat test_code_for1 test_total $(TCODE)
   runbat code_tony.bat test_code_array1 test_total $(TCODE)
   runbat code_tony.bat test_code_array2 test_total $(TCODE)
   runbat code_tony.bat test_code_array2Global test_total $(TCODE)
   runbat code_tony.bat test_code_recordListe test_total $(TCODE)
   runbat code_tony.bat test_code_arrayRecord1a test_total $(TCODE)
   runbat code_tony.bat test_code_arrayRecord1b test_total $(TCODE) \
-noRecordArray
   runbat code_tony.bat test_code_recordArray1a test_total $(TCODE)
   runbat code_tony.bat test_code_recordArray1b test_total $(TCODE) \
-noRecordArray
   runbat code_tony.bat test_code_recordArray2a test_total $(TCODE)
   runbat code_tony.bat test_code_recordArrayCyklisk test_total $(TCODE)
   runbat code_tony.bat test_code_call test_total $(TCODE)

   runbat code_tony.bat test_code_indexHigh test_total $(TCODE)
   runbat code_tony.bat test_code_indexLow test_total $(TCODE)
   runbat code_tony.bat test_code_heapend test_total $(TCODE) -heapSize 20
   runbat code_tony.bat test_code_zerodiv test_total $(TCODE)
   runbat code_tony.bat test_code_nonposelm test_total $(TCODE)


.PHONY:  sducode
sducode:
   runbat code_tony.bat test_code_sdu_AbsTest test_total $(TCODE)
   runbat code_tony.bat test_code_sdu_ArrayIndex test_total $(TCODE)
   runbat code_tony.bat test_code_sdu_Assoc test_total $(TCODE)
   runbat code_tony.bat test_code_sdu_ErrOutOfBounds1 test_total $(TCODE)
   runbat code_tony.bat test_code_sdu_ErrOutOfBounds2 test_total $(TCODE)
   runbat code_tony.bat test_code_sdu_Factorial test_total $(TCODE)
   runbat code_tony.bat test_code_sdu_Function $(TCODE)
   runbat code_tony.bat test_code_sdu_IfThen test_total $(TCODE)
   runbat code_tony.bat test_code_sdu_Knapsack test_total $(TCODE)
   runbat code_tony.bat test_code_sdu_SimpleRecord test_total $(TCODE)
   runbat code_tony.bat ttest_code_sdu_StaticLink test_total $(TCODE)
   runbat code_tony.bat test_code_sdu_WhileDo test_total $(TCODE)
```

```
.PHONY:  expand
expand:
   runbat check_tony.bat test_typeminus test_total $(TCHECK)
   runbat check_tony.bat test_typefor1 test_total $(TCHECK)

.PHONY:  weed
weed:
   runbat check_tony.bat test_weed1 test_total $(TCHECK)
   runbat check_tony.bat test_weed2 test_total $(TCHECK)
   runbat check_tony.bat test_weed3 test_total $(TCHECK)
   runbat check_tony.bat test_weed4 test_total $(TCHECK)

.PHONY:  stmtype
stmtype:
   runbat check_tony.bat test_typereturn1 test_total $(TCHECK)
   runbat check_tony.bat test_typereturn2 test_total $(TCHECK)
   runbat check_tony.bat test_typereturn3 test_total $(TCHECK)
   runbat check_tony.bat test_typecall1 test_total $(TCHECK)
   runbat check_tony.bat test_typecall2 test_total $(TCHECK)
   runbat check_tony.bat test_typeassign1 test_total $(TCHECK)
   runbat check_tony.bat test_typewrite1 test_total $(TCHECK)
   runbat check_tony.bat test_typenew1 test_total $(TCHECK)
   runbat check_tony.bat test_typeif1 test_total $(TCHECK)
   runbat check_tony.bat test_typewhile1 test_total $(TCHECK)

.PHONY:  exptype
exptype:
   runbat check_tony.bat test_typeexp1 test_total $(TCHECK)
   runbat check_tony.bat test_typeexp2 test_total $(TCHECK)
   runbat check_tony.bat test_typeexp3 test_total $(TCHECK)
   runbat check_tony.bat test_typeexp4 test_total $(TCHECK)
   runbat check_tony.bat test_typestruct1 test_total $(TCHECK)
   runbat check_tony.bat test_typearray1 test_total $(TCHECK)
   runbat check_tony.bat test_typearray2 test_total $(TCHECK)

.PHONY:  scopetype
scopetype:
   runbat check_tony.bat test_typescope1 test_total $(TCHECK)

.PHONY:  com
com:
   runbat check_tony.bat test_com1 test_total $(TCHECK)
   runbat check_tony.bat test_com2 test_total $(TCHECK)
   runbat check_tony.bat test_com3 test_total $(TCHECK)
   runbat check_tony.bat test_com4 test_total $(TCHECK)

.PHONY:  decl
decl:
   runbat check_tony.bat test_decl1 test_total $(TCHECK)
   runbat check_tony.bat test_decl1b test_total -S -prettyTxt -xref
   runbat check_tony.bat test_decl2 test_total $(TCHECK)
   runbat check_tony.bat test_decl3 test_total $(TCHECK)
   runbat check_tony.bat test_decl4 test_total $(TCHECK)

.PHONY:  stm
stm:
   runbat check_tony.bat test_stm1 test_total $(TCHECK)
   runbat check_tony.bat test_stm2 test_total $(TCHECK)
   runbat check_tony.bat test_stm3 test_total $(TCHECK)
   runbat check_tony.bat test_stm4 test_total $(TCHECK)
```

```
.PHONY:  preced
preced:
   runbat check_tony.bat test_precedens1 test_total $(TCHECK)
   runbat check_tony.bat test_precedens2 test_total $(TCHECK)
   runbat check_tony.bat test_precedens3 test_total $(TCHECK)
   runbat check_tony.bat test_precedens4 test_total $(TCHECK)

.PHONY:  term
term:
   runbat check_tony.bat test_term1 test_total $(TCHECK) -prettyXml
```

check_tony.bat

```
del %1.consol.txt
del %1.pretty.xml
del %1.pretty.txt
del %1.s
tony -msg %1.consol.txt %3 %4 %5 %6 %7 %8 %9 < %1.tony
echo : >> %2.txt
echo ------------------------------------------------------ >> %2.txt
echo : >> %2.txt
echo Start på test af %1.tony >> %2.txt
echo ------------------------------------------------------ >> %2.txt
echo %1.tony >> %2.txt
echo : >> %2.txt
printLnrCmd.exe < %1.tony >> %2.txt
echo ------------------------------------------------------ >> %2.txt
echo : >> %2.txt
echo tony.exe consol output >> %2.txt
echo : >> %2.txt
type %1.consol.txt>> %2.txt
echo ------------------------------------------------------ >> %2.txt
echo : >> %2.txt
echo tonyprettyTxt.txt >> %2.txt
echo : >> %2.txt
type tonyprettyTxt.txt >> %2.txt
echo ------------------------------------------------------ >> %2.txt
echo : >> %2.txt
echo tonyprettyXml.xml >> %2.txt
echo : >> %2.txt
type tonyprettyXml.xml >> %2.txt
echo ------------------------------------------------------ >> %2.txt
echo : >> %2.txt
echo tonysymbol_xref.txt >> %2.txt
echo : >> %2.txt
type tonysymbol_xref.txt >> %2.txt
echo ------------------------------------------------------ >> %2.txt
ren tonyprettyXml.xml %1.pretty.xml
ren tonyprettyTxt.txt %1.pretty.txt
ren tonysymbol_xref.txt %1.xref.txt
echo Slut på test af %1.tony >> %2.txt
echo : >> %2.txt
echo :::::::::::::::::::::::::::::::::::::::::::::::::::: >> %2.txt
echo : >> %2.txt
```

code_tony.bat

```
del %1.consol.txt
del %1.pretty.xml
del %1.pretty.txt
del %1.run.txt
```

```
del %1.s
del %1.o
del %1.exe
tony -msg %1.consol.txt %3 %4 %5 %6 %7 %8 %9 < %1.tony >> %1.s
echo : >> %2.txt
echo ---------------------------------------------------- >> %2.txt
echo : >> %2.txt
echo Start på test af %1.tony >> %2.txt
echo ---------------------------------------------------- >> %2.txt
echo %1.tony >> %2.txt
echo : >> %2.txt
printLnrCmd.exe < %1.tony >> %2.txt
echo ---------------------------------------------------- >> %2.txt
echo : >> %2.txt
echo tony.exe consol output >> %2.txt
echo : >> %2.txt
type %1.consol.txt>> %2.txt
echo ---------------------------------------------------- >> %2.txt
REM gcc -c -ansi -pedantic printf.s -o printf.o
gcc -c -ansi -pedantic %1.s -o %1.o
gcc printf.o %1.o -o %1.exe
del %1.o
REM %1.exe
REM pause
%1.exe > Asm_tonyCode.run.txt >> %1.run.txt
echo : >> %2.txt
echo consol output fra %1.exe >> %2.txt
echo : >> %2.txt
type %1.run.txt>> %2.txt
echo ---------------------------------------------------- >> %2.txt
echo : >> %2.txt
echo tonyprettyTxt.txt >> %2.txt
echo : >> %2.txt
type tonyprettyTxt.txt >> %2.txt
echo ---------------------------------------------------- >> %2.txt
echo : >> %2.txt
echo tonyprettyXml.xml >> %2.txt
echo : >> %2.txt
type tonyprettyXml.xml >> %2.txt
echo ---------------------------------------------------- >> %2.txt
echo : >> %2.txt
echo tonysymbol_xref.txt >> %2.txt
echo : >> %2.txt
type tonysymbol_xref.txt >> %2.txt
echo ---------------------------------------------------- >> %2.txt
echo : >> %2.txt
echo %1.s >> %2.txt
echo : >> %2.txt
type %1.s >> %2.txt
echo ---------------------------------------------------- >> %2.txt
ren tonyprettyXml.xml %1.pretty.xml
ren tonyprettyTxt.txt %1.pretty.txt
ren tonysymbol_xref.txt %1.xref.txt
echo Slut på test af %1.tony >> %2.txt
echo : >> %2.txt
echo :::::::::::::::::::::::::::::::::::::::::::::::::: >> %2.txt
echo : >> %2.txt
```