Лабораторная работа № 3. «Компиляция базового императивного языка»

23 мая 2024 г.

Сергей Виленский, ИУ9-62Б

Цель работы

Целью данной работы является ознакомление к компиляцией базовых конструкций императивного языка программирования: функциями, операторами, выражениями.

Индивидуальный вариант

1. Операция присваивания, т.е. `(t.Expr "=" t.Expr)` может встречаться в выражениях:

```
t.Expr ::= …
    | (t.Expr "=" t.Expr)
```

Значением операции присваивания должно быть присвоенное значение.

2. Оператор блока определяет локальные переменные:

```
t.Statement ::= …
  | (block t.LocalVars e.Code)
```

Область видимости локальных переменных ограничена этим блоком. При генерации кода текст для `e.Code` окружается инструкциями `PUSHN` и `DROPN`, резервирующими место локальных переменных на стеке.

3. Операция блока внутри выражения

```
t.Expr ::= …
  | (let t.LocalVars e.Code t.Expr)
```

Реализация

src/BinOp.ref

```
/**
 <BinOp "+" | "-" | "*" | "/" | "%" | "&" | "|" | "~"> == s.AsmCodeCmd
*/
$ENTRY BinOp {
  "+" = ADD;
  "-" = SUB;
  "*" = MUL;
  "/" = DIV;
  "%" = MOD;
  "&" = BITAND;
  "|" = BITOR;
  "~" = BITNOT;
}
```

src/BoolExpr.ref

```
*$FROM src/Expr.ref
*$FROM src/RelOp.ref
*$FROM src/Name.ref
$EXTERN Expr, RelOp, Name;

/**
  <BoolExpr
    t.Globals t.Locals t.TLabel t.FLabel
      TRUE | FALSE
    | (t.Expr s.RelOp t.Expr)
    | (not t.BoolExpr)
    | (t.BoolExpr and t.BoolExpr)
    | (t.BoolExpr or t.BoolExpr)
  > == t.Locals s.AsmCodeCmd*
*/
$ENTRY BoolExpr {
  t.Globals t.Locals (e.TLabel) (e.FLabel) TRUE
    = t.Locals
      e.TLabel JMP;

  t.Globals t.Locals (e.TLabel) (e.FLabel) FALSE
    = t.Locals
      e.FLabel JMP;

  t.Globals t.Locals (e.TLabel) (e.FLabel) (not t.BoolExpr)
    = <BoolExpr (e.FLabel) (e.TLabel) t.BoolExpr>;
  t.Globals (e.LocalsL (_bool_count s.BoolNum) e.LocalsR) (e.TLabel) (e.FLabel)
    (t.BoolExprL and t.BoolExprR)
   , (e.LocalsL (_bool_count <Add s.BoolNum 1>) e.LocalsR) : t.Locals1
    , <Name t.Locals1 _func_name> : SUCC s.FuncName
    , '_bool_' <itoa s.BoolNum> '_' s.FuncName : e.BoolName
   , <BoolExpr t.Globals t.Locals1 (e.BoolName) (e.FLabel) t.BoolExprL> : t.Locals2 e.L
   , <BoolExpr t.Globals t.Locals2 (e.TLabel) (e.FLabel) t.BoolExprR> : t.Locals3 e.R
    = t.Locals3
      e.L '\n'
      ':' e.BoolName '\n'
      e.R '\n';

  t.Globals (e.LocalsL (_bool_count s.BoolNum) e.LocalsR) (e.TLabel) (e.FLabel)
   (t.BoolExprL or t.BoolExprR)
   , (e.LocalsL (_bool_count <Add s.BoolNum 1>) e.LocalsR) : t.Locals1
    , <Name t.Locals1 _func_name> : SUCC s.FuncName
    , '_bool_' <itoa s.BoolNum> '_' s.FuncName : e.BoolName
   , <BoolExpr t.Globals t.Locals1 (e.TLabel) (e.BoolName) t.BoolExprL> : t.Locals2 e.L
   , <BoolExpr t.Globals t.Locals2 (e.TLabel) (e.FLabel) t.BoolExprR> : t.Locals3 e.R
    = t.Locals3
      e.L '\n'
      ':' e.BoolName '\n'
      e.R '\n';

  t.Globals t.Locals (e.TLabel) (e.FLabel) (t.ExprL s.RelOp t.ExprR)
    , <Expr t.Globals t.Locals t.ExprL> : t.Locals1 e.L
    , <Expr t.Globals t.Locals1 t.ExprR> : t.Locals2 e.R
    = t.Locals2
      e.L '\n'
      e.R '\n'
      CMP e.TLabel <RelOp s.RelOp> '\n'
      e.FLabel JMP;
}

* Symb
itoa {
  s.Int, <Compare s.Int 9> : {
      '+' = <itoa <Div s.Int 10>> <itoa <Mod s.Int 10>>;
      e._ = <Chr <Add 48 s.Int>>
    };
}
```

src/Code.ref

```
*$FROM src/Statement.ref
```

```
$EXTERN Statement;

/**
  <Code t.Globals t.Locals t.Statement*> == t.Locals s.AsmCodeCmd*
*/
$ENTRY Code {
  t.Globals t.Locals t.Statement e.Statements
    , <Statement t.Globals t.Locals t.Statement> : t.Locals1 e.St
    , <Code t.Globals t.Locals1 e.Statements> : t.Locals2 e.Sts
    = t.Locals2
      e.St e.Sts;
  t.Globals t.Locals = t.Locals;
}
```

src/Const.ref

```
*$FROM src/ConstExpr.ref
$EXTERN ConstExpr;

/**
  <Const t.Globals (const s.Name "=" t.ConstExpr)> == t.Globals
*/
$ENTRY Const {
  (e.Globals) (const s.Name "=" t.ConstExpr)
    , <ConstExpr (e.Globals) t.ConstExpr> : s.ConstVal
    = (e.Globals (s.Name s.ConstVal));
}
```

src/ConstExpr.ref

```
*$FROM src/Name.ref
*$FROM src/ConstExpr.ref
$EXTERN Name, ConstExpr;

/**
  <ConstExpr
    t.Globals
      s.Name
    | s.NUMBER
    | ("-" t.ConstExpr)
    | (t.ConstExpr s.BinOp t.ConstExpr)
  > == s.ConstVal
*/
$ENTRY ConstExpr {
  t.Globals s.Name
    , <Name t.Globals s.Name> : SUCC e.Val
    = e.Val;

  t.Globals s.NUMBER = s.NUMBER;

  t.Globals ("-" t.ConstExpr) = <Sub 0 <ConstExpr t.Globals t.ConstExpr>>;

  t.Globals (t.ConstExprL s.BinOp t.ConstExprR)
    , <ConstExpr t.Globals t.ConstExprL> : s.LeftVal
    , <ConstExpr t.Globals t.ConstExprR> : s.RightVal
    , s.BinOp : {
      "+" = <Add s.LeftVal s.RightVal>;
      "-" = <Sub s.LeftVal s.RightVal>;
      "*" = <Mul s.LeftVal s.RightVal>;
      "/" = <Div s.LeftVal s.RightVal>;
      "%" = <Mod s.LeftVal s.RightVal>;
    };
}
```

src/Definition.ref

```
*$FROM src/Struct.ref
*$FROM src/Const.ref
*$FROM src/GlobalVar.ref
*$FROM src/Function.ref
$EXTERN Struct, Const, GlobalVar, Function;

/**
  <Definition t.Globals
      t.Struct
    | t.Const
    | t.GlobalVar
    | t.Function> == t.Globals s.AsmCodeCmd*
*/
$ENTRY Definition {
  t.Globals (struct e.Struct) = <Struct t.Globals (struct e.Struct)>;
  t.Globals (const e.Const) = <Const t.Globals (const e.Const)>;
  t.Globals (var e.GlobalVar) = <GlobalVar t.Globals (var e.GlobalVar)>;
  t.Globals (function e.Function) = <Function t.Globals (function e.Function)>;
}
```

src/Expr.ref

```
*$FROM src/Name.ref
*$FROM src/BinOp.ref
*$FROM src/LocalVars.ref
*$FROM src/Code.ref
```

```
*$FROM LibraryEx
$EXTERN Name, BinOp, LocalVars, Code, Map;

/**
  <Expr
    t.Globals
    t.Locals
      s.Name
    | s.NUMBER
    | (L t.Expr)
    | ("-" t.Expr)
    | (t.Expr s.BinOp t.Expr)
    | (call t.Expr t.Expr*)
    | (asm s.ANY+)
    | (t.Expr "=" t.Expr)
    | (let t.LocalVars e.Code t.Expr)
  > == t.Locals s.AsmCodeCmd*
*/
$ENTRY Expr {
  t.Globals t.Locals (t.ExprL "=" t.ExprR)
    , <Expr t.Globals t.Locals t.ExprL> : t.Locals1 e.L
    , <Expr t.Globals t.Locals1 t.ExprR> : t.Locals2 e.R
    = t.Locals2
      e.L e.R SWAP OVER SAVE;

  t.Globals (e.Locals) (let (e.LocalVars) e.Code t.Expr)
    , <LocalVars e.LocalVars> : {
    e.LocalVars1_ '|' s.AllocateSize_ PUSHN '\n' = e.LocalVars1_ s.AllocateSize_;
      '|' = 0;
    } : e.LocalVars1 s.AllocateSize
    , <Map {
      (s.Name e.Val (e.FPWay)) = (s.Name e.Val (e.FPWay LOAD));
      e.Other = e.Other;
    } e.Locals> e.LocalVars1 : e.Locals1
    , <Code t.Globals (e.Locals1) e.Code> : (e.Locals2) e.AsmCode
    , <Expr t.Globals (e.Locals2) t.Expr> : (e.Locals3) e.ExprCode
    , <Map {
      (s.Name e.Val (e.FPWay LOAD)) = (s.Name e.Val (e.FPWay));
      (s.Name e.Val (GETFP)) = /* пусто */;
      e.Other = e.Other;
    } e.Locals3> : e.Locals4
    = (e.Locals4)
      GETFP GETSP SETFP '\n'
      s.AllocateSize PUSHN '\n'
      e.AsmCode
      e.ExprCode
      SETRV
      GETFP SETSP SETFP '\n'
      GETRV;

  t.Globals t.Locals s.Number
    , <Type s.Number> : 'N' e._
    = t.Locals s.Number;

  t.Globals t.Locals s.Name
    , <Name t.Globals s.Name> : SUCC e.Val
    = t.Locals e.Val;

  t.Globals t.Locals s.Name
    , <Name t.Locals s.Name> : ERR
    = t.Locals '_' s.Name;

  t.Globals t.Locals s.Name
    , <Name t.Locals s.Name> : {
      SUCC '-' s.Val (e.FPWay) = s.Val e.FPWay SUB;
      SUCC s.Val (e.FPWay) = s.Val e.FPWay ADD;
    } : s.Val e.FPWay s.Op
    = t.Locals e.FPWay s.Val s.Op;

  t.Globals t.Locals (L t.Expr) = <Expr t.Globals t.Locals t.Expr> LOAD;

  t.Globals t.Locals ("-" t.Expr) = <Expr t.Globals t.Locals t.Expr> NEG;

  t.Globals t.Locals (call t.Expr e.Exprs)
    , <CompileArgs t.Globals t.Locals e.Exprs> : t.Locals1 e.Args
    , <Expr t.Globals t.Locals1 t.Expr> : t.Locals2 e.Func
    = t.Locals2
      e.Args e.Func
      CALL GETRV;

  t.Globals t.Locals (asm e.Code) = t.Locals e.Code;

  t.Globals t.Locals (t.ExprL s.BinOp t.ExprR)
    , <Expr t.Globals t.Locals t.ExprL> : t.Locals1 e.L
    , <Expr t.Globals t.Locals1 t.ExprR> : t.Locals2 e.R
    = t.Locals2
      e.L e.R
      <BinOp s.BinOp>;
}

CompileArgs {
  t.Globals t.Locals e.Args t.Arg
```

```
    , <Expr t.Globals t.Locals t.Arg> : t.Locals1 e.CmpArg
    , <CompileArgs t.Globals t.Locals1 e.Args> : t.Locals2 e.CmpArgs
    = t.Locals2
      e.CmpArg e.CmpArgs;
  t.Globals t.Locals = t.Locals;
}
```

src/Function.ref

```
*$FROM src/LocalVars.ref
*$FROM src/Code.ref
$EXTERN LocalVars, Code;

/**
 <Function (function s.Name (s.Name*) t.LocalVars? e.Code)> == t.Globals s.AsmCodeCmd*
*/
$ENTRY Function {
  t.Globals (function s.Name (e.Params) (var e.LocalVars) e.Code)
    , <CompileParams 2 0 e.Params> : e.LocalParams s.ParamCount
    , <Compare s.ParamCount 0> : {
      '+' = s.ParamCount RETN;
      e._ = JMP;
    } : e.EpilogCode
    , <LocalVars e.LocalVars> : e.Locals1 '|' e.AllocateCode
    , e.LocalParams e.Locals1
      (_func_name s.Name)
      (_if_count 0)
      (_while_count 0)
      (_bool_count 0) : e.Locals2
    , <Code t.Globals (e.Locals2) e.Code> : t.Locals3 e.AsmCode
    = t.Globals
      ':_' s.Name '\n'
      GETFP GETSP SETFP '\n'
      e.AllocateCode
      e.AsmCode
      ':__' s.Name '\n'
      GETFP SETSP SETFP '\n'
      e.EpilogCode '\n'
      '\n';

  t.Globals (function s.Name t.Params e.Code)
    = <Function t.Globals (function s.Name t.Params (var) e.Code)>;
}

CompileParams {
  s.MemShift s.ParamCount = s.ParamCount;
  s.MemShift s.ParamCount s.ParamName e.Params
    = (s.ParamName s.MemShift (GETFP))
      <CompileParams
        <Add s.MemShift 1>
        <Add s.ParamCount 1>
        e.Params>;
}
```

src/GlobalVar.ref

```
*$FROM src/ConstExpr.ref
*$FROM src/Init.ref
$EXTERN ConstExpr, Init;

/**
  <GlobalVar t.Globals (var s.Name t.ConstExpr e.Init?)> == t.Globals s.AsmCodeCmd*
*/
$ENTRY GlobalVar {
  t.Globals (var s.Name t.ConstExpr) = <GlobalVar t.Globals (var s.Name t.ConstExpr "=")>;
  t.Globals (var s.Name t.ConstExpr e.Init)
    , <ConstExpr t.Globals t.ConstExpr> : s.Size
    = t.Globals ':_' s.Name '\n'
      <Init t.Globals s.Size e.Init> '\n\n';
}
```

src/Init.ref

```
*$FROM src/ConstExpr.ref
$EXTERN ConstExpr;

/**
  <Init t.Globals s.Size "=" t.ConstExpr*> == s.AsmCodeCmd*
*/
$ENTRY Init {
  t.Globals 0 "=" = /* пусто */;
  t.Globals s.Size "=" = 0 <Init t.Globals <Sub s.Size 1> "=">;
  t.Globals s.Size "=" t.ConstExpr e.ConstExprs
    , <ConstExpr t.Globals t.ConstExpr> : s.ConstVal
    = s.ConstVal <Init t.Globals <Sub s.Size 1> "=" e.ConstExprs>;
}
```

src/LocalVars.ref

```
*$FROM src/Name.ref
*$FROM src/ConstExpr.ref
$EXTERN Name, ConstExpr;
```

```
/**
  <LocalVars (var (s.Name t.ConstExpr)*)> == t.Locals e.AllocateCode
*/
$ENTRY LocalVars {
  /* пусто */ = '|';
  e.LocalVars = <_LocalVars 0 e.LocalVars> PUSHN '\n';
}

_LocalVars {
  s.MemShift = '|' s.MemShift;
  s.MemShift (s.VarName s.VarSize) e.LocalVars
    , <Add s.MemShift s.VarSize> : s.VarShift
    = (s.VarName <Sub 0 s.VarShift> (GETFP))
      <_LocalVars s.VarShift e.LocalVars>;
}
```

src/Name.ref

```
/**
  <Name t.Names s.WORD> == s.VAL
*/
$ENTRY Name {
  (e._ (s.WORD e.VAL) e._) s.WORD = SUCC e.VAL;
  e._ = ERR;
}
```

src/Program.ref

```
*$FROM src/Definition.ref
$EXTERN Definition;

/**
  <Program t.Definition*> == s.AsmCodeCmd*
*/
$ENTRY Program {
  e.Definitions
    = GETSP _MEMORY_SIZE SWAP SAVE '\n'
      _main CALL '\n'
      GETRV HALT '\n'
      ':'_MEMORY_SIZE 0 '\n'
      ':'_PROGRAM_SIZE PROGRAM_SIZE '\n\n'
      <_Program () e.Definitions>;
}

/**
```

```
  <_Program t.Globals t.Definition*> == s.AsmCodeCmd*
*/
_Program {
  t.Globals = /* пусто */;
  t.Globals t.Definition e.Definitions
    , <Definition t.Globals t.Definition> : t.DefGlobals e.Code
    = e.Code <_Program t.DefGlobals e.Definitions>;
}
```

src/RelOp.ref

```
/**
  <RelOp "<" | ">" | "==" | "<>" | ">=" | "<="> == s.AsmCodeCmd
*/
$ENTRY RelOp {
  "<" = JLT;
  ">" = JGT;
  "==" = JEQ;
  "<>" = JNE;
  ">=" = JGE;
  "<=" = JLE;
}
```

src/Statement.ref

```
*$FROM src/Expr.ref
*$FROM src/BoolExpr.ref
*$FROM src/Code.ref
*$FROM src/Name.ref
*$FROM src/LocalVars.ref
*$FROM LibraryEx
$EXTERN Expr, BoolExpr, Code, Name, LocalVars, Map;

/**
  <Statement
    t.Globals t.Locals
      (t.Expr "=" t.Expr)
    | (call t.Expr t.Expr*)
    | (return t.Expr)
    | (if t.BoolExpr e.Code)
    | (if t.BoolExpr e.Code else e.Code)
    | (while t.BoolExpr e.Code)
    | (asm s.ANY+)
    | (block t.LocalVars e.Code)
  > == t.Locals s.AsmCodeCmd*
*/
```

```
$ENTRY Statement {
  t.Globals (e.Locals) (block (e.LocalVars) e.Code)
    , <LocalVars e.LocalVars> : {
    e.LocalVars1_ '|' s.AllocateSize_ PUSHN '\n' = e.LocalVars1_ s.AllocateSize_;
      '|' = 0;
    } : e.LocalVars1 s.AllocateSize
    , <Map {
      (s.Name e.Val (e.FPWay)) = (s.Name e.Val (e.FPWay LOAD));
      e.Other = e.Other;
    } e.Locals> e.LocalVars1 : e.Locals1
    , <Code t.Globals (e.Locals1) e.Code> : (e.Locals2) e.AsmCode
    , <Map {
      (s.Name e.Val (e.FPWay LOAD)) = (s.Name e.Val (e.FPWay));
      (s.Name e.Val (GETFP)) = /* пусто */;
      e.Other = e.Other;
    } e.Locals2> : e.Locals3
    = (e.Locals3)
      GETFP GETSP SETFP '\n'
      s.AllocateSize PUSHN '\n'
      e.AsmCode
      GETFP SETSP SETFP '\n';

  t.Globals t.Locals (t.ExprL "=" t.ExprR)
    , <Expr t.Globals t.Locals t.ExprL> : t.Locals1 e.ExprL
    , <Expr t.Globals t.Locals1 t.ExprR> : t.Locals2 e.ExprR
    = t.Locals2
      e.ExprL e.ExprR SAVE '\n';

  t.Globals t.Locals (call t.Expr e.Exprs)
    = <Expr t.Globals t.Locals (call t.Expr e.Exprs)> DROP '\n';

  t.Globals t.Locals (return t.Expr)
    , <Name t.Locals _func_name> : SUCC s.FuncName
    = <Expr t.Globals t.Locals t.Expr>
      SETRV
      '__' s.FuncName JMP '\n';

  t.Globals (e.LocalsL (_if_count s.IfNum) e.LocalsR) (if t.BoolExpr e.CodeT else e.CodeF)
    , (e.LocalsL (_if_count <Add s.IfNum 1>) e.LocalsR) : t.Locals1
    , <Name t.Locals1 _func_name> : SUCC s.FuncName
    , '_if_' <itoa s.IfNum> '_' s.FuncName : e.IfName
    , e.CodeF : {
      /* пусто */ = t.Locals1 ('_exit');
      e._
        , <Code t.Globals t.Locals1 e.CodeF> : t.Locals2 e.FCode
        = t.Locals2
```

```
          '_exit' e.IfName JMP '\n'
          ':_false' e.IfName '\n'
          e.FCode
          ('_false');
    } : t.Locals3 e.ElseCode (e.FalseAlt)
   , <BoolExpr t.Globals t.Locals3 ('_true' e.IfName) (e.FalseAlt e.IfName) t.BoolExpr> :
        t.Locals4 e.BoolCode
    , <Code t.Globals t.Locals4 e.CodeT> : t.Locals5 e.TrueCode
    = t.Locals5
      e.BoolCode '\n'
      ':_true' e.IfName '\n'
      e.TrueCode
      e.ElseCode
      ':_exit' e.IfName '\n';

  t.Globals t.Locals (if t.BoolExpr e.Code)
    = <Statement t.Globals t.Locals (if t.BoolExpr e.Code else)>;

  t.Globals (e.LocalsL (_while_count s.WhileNum) e.LocalsR) (while t.BoolExpr e.InnerCode)
   , (e.LocalsL (_while_count <Add s.WhileNum 1>) e.LocalsR) : t.Locals1
    , <Name t.Locals1 _func_name> : SUCC s.FuncName
    , '_while_' <itoa s.WhileNum> '_' s.FuncName : e.WhileName
   , <BoolExpr t.Globals t.Locals1 ('_true' e.WhileName) ('_exit' e.WhileName) t.BoolExpr> :
        t.Locals2 e.BoolCode
    , <Code t.Globals t.Locals2 e.InnerCode> : t.Locals3 e.Code
    = t.Locals3
      ':_loop' e.WhileName '\n'
      e.BoolCode '\n'
      ':_true' e.WhileName '\n'
      e.Code
      '_loop' e.WhileName JMP '\n'
      ':_exit' e.WhileName '\n';

  t.Globals t.Locals (asm e.ANYS)
    = t.Locals
      e.ANYS '\n';
}

itoa {
  s.Int, <Compare s.Int 9> : {
      '+' = <itoa <Div s.Int 10>> <itoa <Mod s.Int 10>>;
      e._ = <Chr <Add 48 s.Int>>
    };
}
```

src/Struct.ref

```
*$FROM src/Name.ref
*$FROM src/ConstExpr.ref
$EXTERN Name, ConstExpr;

/**
  <Struct t.Globals (struct s.Name (s.Name t.ConstExpr)*)> == t.Globals
*/
$ENTRY Struct {
  (e.Globals) (struct s.Name e.Fields) = (e.Globals <_Struct (e.Globals) 0 s.Name e.Fields>);
}

_Struct {
  t.Globals s.Size s.Name = (s.Name s.Size);
  t.Globals s.Size s.Name ("-" t.ConstExpr) e.Fields
    , <ConstExpr t.Globals t.ConstExpr> : s.FieldSize
    = <_Struct t.Globals <Add s.Size s.FieldSize> s.Name e.Fields>;
  t.Globals s.Size s.Name (s.FieldName t.ConstExpr) e.Fields
    , <ConstExpr t.Globals t.ConstExpr> : s.FieldSize
    = (s.FieldName s.Size)
      <_Struct t.Globals <Add s.Size s.FieldSize> s.Name e.Fields>;
}
```

main.ref

```
*$FROM LibraryEx
*$FROM src/Program.ref
$EXTERN LoadExpr, SaveFile, Program;

$ENTRY Go {
  /* пусто */ = <SaveFile (<Arg 2>) (<Program <LoadExpr <Arg 1>>>)>;
}
```

Тестирование

run.sh

```
sh cmp.sh $1 a.asm
sh int.sh
```

cmp.sh

```
rlmake.bat main.ref -o a.exe
./a.exe source.txt a.asm
rm a.exe
```

int.sh

```
iverilog -o output int.v
if [ $? -eq 0 ]; then
    vvp output
    rm output
fi
```

source.txt

```
(function in ()
    (return (asm IN))
)

(function out (char)
    (asm GETFP 2 ADD LOAD OUT)
)

(function getFP ()
    (return (asm GETFP))
)

(function halt (code)
    (asm GETFP 2 ADD LOAD HALT)
)

(function printInt (int)
    (if ((L int) ">=" 10)
        (call printInt ((L int) "/" 10))
    )
    (call out (((L int) "%" 10) "+" 48))
)

(function newLine ()
    (call out 10)
)

(function nearSquare (x)
    (if ((L x) "<=" 3)
        (x "=" 0)
    else
        (x "=" ((L x) "+" 1))
    )

    (return ((L x) "*" (L x)))
)
```

```
(function gcd (x y)
    (var (res 1))

    (block ((ram 4) (rem 1))
        (while ((L y) "<>" 0)
            (rem "=" ((L x) "%" (x "=" (L y))))
            (y "=" (L rem))
        )
    )
    (res "=" (L x))

    (return (L res))
)

(function main ()
    (call printInt (call nearSquare 3))
    (call newLine)
    (call printInt (call nearSquare 4))
    (call newLine)
    (call printInt (call gcd 4
        (let ((x 1) (y 1))
            (x "=" 1)
            (y "=" (let ((z 1) (w 1))
                (z "=" 49)
                (w "=" 0)
                (while ((L z) ">" (call nearSquare (L w)))
                    (w "=" ((L w) "+" (L x)))
                )
                (call printInt (L w))
                (call newLine)
                (L w)
            ))
            (while ((L x) "<" (L y))
                (x "=" ((L x) "+" 1))
            )
            (call printInt (L x))
            (call newLine)
            (L x)
        )
    ))
    (call newLine)
    (return 0)
)
```

a.asm

```
GETSP _MEMORY_SIZE SWAP SAVE
_main CALL
GETRV HALT
:_MEMORY_SIZE 0
:_PROGRAM_SIZE PROGRAM_SIZE

:_in
GETFP GETSP SETFP
IN SETRV __in JMP
:__in
GETFP SETSP SETFP
JMP

:_out
GETFP GETSP SETFP
GETFP 2 ADD LOAD OUT
:__out
GETFP SETSP SETFP
1 RETN

:_getFP
GETFP GETSP SETFP
GETFP SETRV __getFP JMP
:__getFP
GETFP SETSP SETFP
JMP

:_halt
GETFP GETSP SETFP
GETFP 2 ADD LOAD HALT
:__halt
GETFP SETSP SETFP
1 RETN

:_printInt
GETFP GETSP SETFP
GETFP 2 ADD LOAD
10
CMP _true_if_0_printInt JGE
_exit_if_0_printInt JMP
:_true_if_0_printInt
GETFP 2 ADD LOAD 10 DIV _printInt CALL GETRV DROP
:_exit_if_0_printInt
GETFP 2 ADD LOAD 10 MOD 48 ADD _out CALL GETRV DROP
```

```
:__printInt
GETFP SETSP SETFP
1 RETN

:_newLine
GETFP GETSP SETFP
10 _out CALL GETRV DROP
:__newLine
GETFP SETSP SETFP
JMP

:_nearSquare
GETFP GETSP SETFP
GETFP 2 ADD LOAD
3
CMP _true_if_0_nearSquare JLE
_false_if_0_nearSquare JMP
:_true_if_0_nearSquare
GETFP 2 ADD 0 SAVE
_exit_if_0_nearSquare JMP
:_false_if_0_nearSquare
GETFP 2 ADD GETFP 2 ADD LOAD 1 ADD SAVE
:_exit_if_0_nearSquare
GETFP 2 ADD LOAD GETFP 2 ADD LOAD MUL SETRV __nearSquare JMP
:__nearSquare
GETFP SETSP SETFP
1 RETN

:_gcd
GETFP GETSP SETFP
1 PUSHN
GETFP GETSP SETFP
5 PUSHN
:_loop_while_0_gcd
GETFP LOAD 3 ADD LOAD
0
CMP _true_while_0_gcd JNE
_exit_while_0_gcd JMP
:_true_while_0_gcd
GETFP 5 SUB GETFP LOAD 2 ADD LOAD GETFP LOAD 2 ADD GETFP LOAD 3 ADD LOAD SWAP OVER SAVE MOD SAVE
GETFP LOAD 3 ADD GETFP 5 SUB LOAD SAVE
_loop_while_0_gcd JMP
:_exit_while_0_gcd
GETFP SETSP SETFP
GETFP 1 SUB GETFP 2 ADD LOAD SAVE
GETFP 1 SUB LOAD SETRV __gcd JMP
```

```
:__gcd
GETFP SETSP SETFP
2 RETN

:_main
GETFP GETSP SETFP
3 _nearSquare CALL GETRV _printInt CALL GETRV DROP
_newLine CALL GETRV DROP
4 _nearSquare CALL GETRV _printInt CALL GETRV DROP
_newLine CALL GETRV DROP
GETFP GETSP SETFP
2 PUSHN
GETFP 1 SUB 1 SAVE
GETFP 2 SUB GETFP GETSP SETFP
2 PUSHN
GETFP 1 SUB 49 SAVE
GETFP 2 SUB 0 SAVE
:_loop_while_0_main
GETFP 1 SUB LOAD
GETFP 2 SUB LOAD _nearSquare CALL GETRV
CMP _true_while_0_main JGT
_exit_while_0_main JMP
:_true_while_0_main
GETFP 2 SUB GETFP 2 SUB LOAD GETFP LOAD 1 SUB LOAD ADD SAVE
_loop_while_0_main JMP
:_exit_while_0_main
GETFP 2 SUB LOAD _printInt CALL GETRV DROP
_newLine CALL GETRV DROP
GETFP 2 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SAVE
:_loop_while_1_main
GETFP 1 SUB LOAD
GETFP 2 SUB LOAD
CMP _true_while_1_main JLT
_exit_while_1_main JMP
:_true_while_1_main
GETFP 1 SUB GETFP 1 SUB LOAD 1 ADD SAVE
_loop_while_1_main JMP
:_exit_while_1_main
GETFP 1 SUB LOAD _printInt CALL GETRV DROP
_newLine CALL GETRV DROP
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV 4 _gcd CALL GETRV _printInt CALL GETRV DROP
_newLine CALL GETRV DROP
0 SETRV __main JMP
:__main
```

```
GETFP SETSP SETFP
JMP
```

stdin.txt

stdout.txt

```
*Compiling main.ref:
+Linking C:\...\refal-5-lambda\lib\references\Library.rasl
+Linking C:\...\refal-5-lambda\lib\slim\exe\LibraryEx.rasl
*Compiling src/Program.ref:
*Compiling src/Definition.ref:
*Compiling src/Struct.ref:
*Compiling src/Name.ref:
*Compiling src/ConstExpr.ref:
*Compiling src/Const.ref:
*Compiling src/GlobalVar.ref:
*Compiling src/Init.ref:
*Compiling src/Function.ref:
*Compiling src/LocalVars.ref:
*Compiling src/Code.ref:
*Compiling src/Statement.ref:
*Compiling src/Expr.ref:
*Compiling src/BinOp.ref:
*Compiling src/BoolExpr.ref:
*Compiling src/RelOp.ref:
** Compilation succeeded **
0
25
6
6
2


Program terminated with code           0.
int.v:1348: $finish called at 0 (1s)
```

Вывод

В результате выполнения данной работы было произведено ознакомление с компиляцией базовых конструкций императивного языка программирования: функциями, операторами, выражениями.