Лабораторная работа № 4. «Компиляция объектно-ориентированного языка»

3 июня 2024 г.

Сергей Виленский, ИУ9-62Б

Цель работы

Целью данной работы является ознакомление с компиляцией средств объектно-ориентированного программирования.

Индивидуальный вариант

НУИЯП++ с проверкой типов, реализованной через односвязные списки виртуальных таблиц.

Реализация и тестирование

src/OOP/OOP_BoolExpr.ref

```
*$FROM src/OOP/OOP_Expr.ref
$EXTERN OOP_Expr;

/**
  <OOP_BoolExpr
      TRUE | FALSE
    | (t.Expr s.RelOp t.Expr)
    | (not t.BoolExpr)
    | (t.BoolExpr and t.BoolExpr)
    | (t.BoolExpr or t.BoolExpr)
    | (isinstance e.ObjectPtr s.Name)
  > == t.BoolExpr
*/
$ENTRY OOP_BoolExpr {
  (not t.BoolExpr) = (not <OOP_BoolExpr t.BoolExpr>);
  (t.BoolExprL and t.BoolExprR) = (<OOP_BoolExpr t.BoolExprL> and <OOP_BoolExpr t.BoolExprR>);
  (t.BoolExprL or t.BoolExprR) = (<OOP_BoolExpr t.BoolExprL> or <OOP_BoolExpr t.BoolExprR>);
  (isinstance t.ObjectPtr s.Name)
    = (
      (let ((vtable_let__ 1) (res_let__ 1))
        (vtable_let__ "=" (L <OOP_Expr t.ObjectPtr>))
        (while (
          ((L vtable_let__) "<>" 0) and
       ((L vtable_let__) "<>" <Implode_Ext <Explode s.Name> '_vtbl__'>)
        )
          (vtable_let__ "=" (L (L vtable_let__)))
        )
        (if ((L vtable_let__) "<>" 0)
          (res_let__ "=" 1)
        else
          (res_let__ "=" 0)
        )
        (L res_let__)
      ) "==" 1);

 (t.ExprL s.RelOp t.ExprR) = (<OOP_Expr t.ExprL> s.RelOp <OOP_Expr t.ExprR>);
  e.Other = e.Other;
}
```

src/OOP/OOP_Class.ref

```
*$FROM src/OOP/OOP_Code.ref
*$FROM LibraryEx
$EXTERN OOP_Code, Map, Inc;

/**
  <OOP_Class
    t.Classes
    (class s.Name (s.Name?)
      (fields (s.Name t.ConstExpr)*)?
      (method s.Name (s.Name+) t.LocalVars? e.Code)*
  )> == t.Classes t.Function

  t.Classes ::= ((s.ClassName t.MethodList)*)
  t.MethodList ::= ((s.MethodOwner s.MethodName)*)
*/
$ENTRY OOP_Class {
  (e.Classes) (class s.Name () (fields e.Fields) e.Methods)
    = (e.Classes
        (s.Name
          (<Map
            { (method s.MethodName e._) = (s.Name s.MethodName); }
            e.Methods
          >)
        )
      )
      (struct s.Name
        ("-" 1)
        e.Fields
      )
      (struct <Implode_Ext <Explode s.Name> '_class__'>
        <Map
          { (method s.MethodName e._) = (s.MethodName 1); }
          e.Methods
        >
      )
      <Map {
        (method s.MethodName t.Params e.Body)
       = (function <Implode_Ext <Explode s.Name> '__' <Explode s.MethodName>>
            t.Params
            <OOP_Code e.Body>
          )
      } e.Methods>
      (var <Implode_Ext <Explode s.Name> '_vtbl__'>
        (<Implode_Ext <Explode s.Name> '_class__'> "+" 1) "="
        0
        <Map
          {
            (method s.MethodName e._)
         = <Implode_Ext <Explode s.Name> '__' <Explode s.MethodName>>;
          }
          e.Methods
        >
      );

  (e.Classes) (class s.Name (s.Base) (fields e.Fields) e.Methods)
    , e.Classes : e._ (s.Base (e.BaseMethods)) e._
    , <Map {
      (method s.NewMethod t.Params e.Body)
        , e.BaseMethods : e._ (s._ s.NewMethod) e._
        = /* пусто */;
      e.Method = e.Method;
    } e.Methods> : e.NewMethods
    = (e.Classes (s.Name
        (
          <Map {
            (s.BaseName s.BaseMethod)
              , e.Methods : e._ (method s.BaseMethod e._) e._
              = (s.Name s.BaseMethod);
            (s.BaseName s.BaseMethod) = (s.BaseName s.BaseMethod);
          } e.BaseMethods>
          <Map {
            (method s.NewMethod e._) = (s.Name s.NewMethod);
          } e.NewMethods>
        )
      ))
      (struct s.Name
        ("-" s.Base)
        e.Fields
      )
      (struct <Implode_Ext <Explode s.Name> '_class__'>
        ("-" <Implode_Ext <Explode s.Base> '_class__'>)
        <Map
          {
            (method s.MethodName e._) = (s.MethodName 1);
          }
          e.NewMethods
        >
      )
      <Map {
        (method s.MethodName t.Params e.Body)
       = (function <Implode_Ext <Explode s.Name> '__' <Explode s.MethodName>>
            t.Params
            <OOP_Code e.Body>
          )
      } e.Methods>
      (var <Implode_Ext <Explode s.Name> '_vtbl__'>
        (<Implode_Ext <Explode s.Name> '_class__'> "+" 1) "="
        <Implode_Ext <Explode s.Base> '_vtbl__'>
        <Map {
          /* переопределённый метод */
          (s.BaseName s.BaseMethod)
            , e.Methods : e._ (method s.BaseMethod e._) e._
        = <Implode_Ext <Explode s.Name> '__' <Explode s.BaseMethod>>;
          /* унаследованный метод */
          (s.BaseName s.BaseMethod)
        = <Implode_Ext <Explode s.BaseName> '__' <Explode s.BaseMethod>>;
        } e.BaseMethods>
        /* новый метод */
        <Map
          {
            (method s.MethodName e._)
         = <Implode_Ext <Explode s.Name> '__' <Explode s.MethodName>>;
          }
```

```
          e.NewMethods
        >
      );

  t.Classes (class s.Name t.Base e.Methods) = <OOP_Class t.Classes (class s.Name t.Base (fields)
}
```

src/OOP/OOP_Code.ref

```
*$FROM src/OOP/OOP_Statement.ref
$EXTERN OOP_Statement;

/**
  <OOP_Code t.Statement*> == e.Code
*/
$ENTRY OOP_Code {
  t.Statement e.Statements
    = <OOP_Statement t.Statement> <OOP_Code e.Statements>;
  /* пусто */ = /* пусто */;
}
```

src/OOP/OOP_Definition.ref

```
*$FROM src/OOP/OOP_Function.ref
*$FROM src/OOP/OOP_Class.ref
$EXTERN OOP_Function, OOP_Class;

/**
  <OOP_Definition
    t.Classes
      t.Struct
    | t.Const
    | t.GlobalVar
    | t.Function
    | t.Class> == t.Classes t.Definition
*/
$ENTRY OOP_Definition {
  t.Classes (function e.Function) = t.Classes <OOP_Function (function e.Function)>;
  t.Classes (class e.Class) = <OOP_Class t.Classes (class e.Class)>;
  t.Classes t.Other = t.Classes t.Other;
}
```

src/OOP/OOP_Expr.ref

```
*$FROM src/OOP/OOP_Code.ref
*$FROM LibraryEx
```

```
$EXTERN OOP_Code, Map;

/**
  <OOP_Expr
      s.Name
    | s.NUMBER
    | (L t.Expr)
    | ("-" t.Expr)
    | (t.Expr s.BinOp t.Expr)
    | (call t.Expr t.Expr*)
    | (mcall t.Expr s.Name t.Expr*)
    | (asm s.ANY+)
    | (t.Expr "=" t.Expr)
    | (let t.LocalVars e.Code t.Expr)
  > == t.Expr
*/
$ENTRY OOP_Expr {
  (L t.Expr) = (L <OOP_Expr t.Expr>);
  ("-" t.Expr) = ("-" <OOP_Expr t.Expr>);
  (call t.Expr e.Exprs) = (call <Map OOP_Expr t.Expr e.Exprs>);
  (t.ExprL "=" t.ExprR) = (<OOP_Expr t.ExprL> "=" <OOP_Expr t.ExprR>);
  (let t.LocalVars e.Code t.Expr) = (let t.LocalVars <OOP_Code e.Code> <OOP_Expr t.Expr>);

  (mcall t.Object s.Method e.Args)
    = (let ((object_let__ 1))
        (object_let__ "=" <OOP_Expr t.Object>)
        (call
          (L (
            ((L (L object_let__)) "+" 1) "+" s.Method
          ))
          (L object_let__) e.Args
        )
      );

  (t.ExprL s.BinOp t.ExprR) = (<OOP_Expr t.ExprL> s.BinOp <OOP_Expr t.ExprR>);
  e.Other = e.Other;
}
```

src/OOP/OOP_Function.ref

```
*$FROM src/OOP/OOP_Code.ref
$EXTERN OOP_Code;

/**
  <OOP_Function (function s.Name (s.Name*) t.LocalVars? e.Code)> == t.Function
*/
```

```
$ENTRY OOP_Function {
  (function s.Name t.Params (var e.LocalVars) e.Code)
   = (function s.Name t.Params (var e.LocalVars) <OOP_Code e.Code>);

  (function s.Name t.Params e.Code)
    = (function s.Name t.Params <OOP_Code e.Code>);
}
```

src/OOP/OOP_Program.ref

```
*$FROM src/OOP/OOP_Definition.ref
$EXTERN OOP_Definition;

/**
  <OOP_Program t.Definition*> == e.Program
*/
$ENTRY OOP_Program {
    e.Definitions = <_OOP_Program () e.Definitions>;
}

_OOP_Program {
  t.Classes t.Definition e.Definitions
   , <OOP_Definition t.Classes t.Definition> : t.Classes1 e.Definition1
    = e.Definition1 <_OOP_Program t.Classes1 e.Definitions>;
  t.Classes = /* пусто */;
}
```

src/OOP/OOP_Statement.ref

```
*$FROM src/OOP/OOP_Expr.ref
*$FROM src/OOP/OOP_BoolExpr.ref
*$FROM src/OOP/OOP_Code.ref
*$FROM LibraryEx
$EXTERN OOP_Expr, OOP_BoolExpr, OOP_Code, Map;

/**
  <OOP_Statement
      (t.Expr "=" t.Expr)
    | (call t.Expr t.Expr*)
    | (mcall t.Expr s.Name 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)
    | (init t.ObjectPtr s.Name)
  > == t.Statement
*/
$ENTRY OOP_Statement {
  (t.ObjectPtr "=" t.Expr) = (<OOP_Expr t.ObjectPtr> "=" <OOP_Expr t.Expr>);
  (call t.Expr e.Args) = (call <Map OOP_Expr t.Expr e.Args>);
  (return t.Expr) = (return <OOP_Expr t.Expr>);
  (if t.BoolExpr e.CodeT else e.CodeF)
   = (if <OOP_BoolExpr t.BoolExpr> <OOP_Code e.CodeT> else <OOP_Code e.CodeF>);
  (if t.BoolExpr e.Code) = (if <OOP_BoolExpr t.BoolExpr> <OOP_Code e.Code>);
  (while t.BoolExpr e.Code) = (while <OOP_BoolExpr t.BoolExpr> <OOP_Code e.Code>);
  (block t.LocalVars e.Code) = (block t.LocalVars <OOP_Code e.Code>);

  (init t.ObjectPtr s.Name) = (<OOP_Expr t.ObjectPtr> "=" <Implode_Ext <Explode s.Name> '_vtbl__
  (mcall t.Object s.Method e.Args) = <OOP_Expr (mcall t.Object s.Method e.Args)>;

  e.Other = e.Other
}
```

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 t.Globals t.Locals (e.FLabel) (e.TLabel) t.BoolExpr>;

 t.Globals (e.LocalsL (_bool_count s.BoolNum) e.LocalsR) (e.TLabel) (e.FLabel) (t.BoolExprL and
   , (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
   , (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.Name
    , <Type s.Name> : 'W' e._
    = <Implode_Ext '_' <Explode s.Name>>;

  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 t.Globals 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 (t.Globals 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 t.Globals 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 {
  t.Globals = '|';
  t.Globals e.LocalVars = <_LocalVars t.Globals 0 e.LocalVars> PUSHN '\n';
}

_LocalVars {
  t.Globals s.MemShift = '|' s.MemShift;
  t.Globals s.MemShift (s.VarName s.VarSize) e.LocalVars
    , <Add s.MemShift <ConstExpr t.Globals s.VarSize>> : s.VarShift
    = (s.VarName <Sub 0 s.VarShift> (GETFP))
      <_LocalVars t.Globals 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 t.Globals 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.Localst (let (e.LocalVars) e.Code t.Expr)
   = <Expr t.Globals t.Localst (let (e.LocalVars) e.Code t.Expr)> 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>;
}
```

a.asm

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

:_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

:_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
```

```
:_Animal__setName
GETFP GETSP SETFP
GETFP 2 ADD LOAD 1 ADD GETFP 3 ADD LOAD SAVE
:__Animal__setName
GETFP SETSP SETFP
2 RETN

:_Animal__getName
GETFP GETSP SETFP
GETFP 2 ADD LOAD 1 ADD LOAD SETRV __Animal__getName JMP
:__Animal__getName
GETFP SETSP SETFP
1 RETN

:_Animal__getType
GETFP GETSP SETFP
1 SETRV __Animal__getType JMP
:__Animal__getType
GETFP SETSP SETFP
1 RETN

:_Animal_vtbl__
0 _Animal__setName _Animal__getName _Animal__getType

:_Cat__getType
GETFP GETSP SETFP
2 SETRV __Cat__getType JMP
:__Cat__getType
GETFP SETSP SETFP
1 RETN

:_Cat__setBreed
GETFP GETSP SETFP
GETFP 2 ADD LOAD 2 ADD GETFP 3 ADD LOAD SAVE
:__Cat__setBreed
GETFP SETSP SETFP
2 RETN

:_Cat__getBreed
GETFP GETSP SETFP
GETFP 2 ADD LOAD 2 ADD LOAD SETRV __Cat__getBreed JMP
:__Cat__getBreed
GETFP SETSP SETFP
1 RETN

:_Cat_vtbl__
```

```
_Animal_vtbl__ _Animal__setName _Animal__getName _Cat__getType _Cat__setBreed _Cat__getBreed

:_B__ff
GETFP GETSP SETFP
GETFP 2 ADD SETRV __B__ff JMP
:__B__ff
GETFP SETSP SETFP
1 RETN

:_B__gg
GETFP GETSP SETFP
GETFP 2 ADD SETRV __B__gg JMP
:__B__gg
GETFP SETSP SETFP
1 RETN

:_B__hh
GETFP GETSP SETFP
GETFP 2 ADD SETRV __B__hh JMP
:__B__hh
GETFP SETSP SETFP
1 RETN

:_B__kk
GETFP GETSP SETFP
GETFP 2 ADD SETRV __B__kk JMP
:__B__kk
GETFP SETSP SETFP
1 RETN

:_B_vtbl__
0 _B__ff _B__gg _B__hh _B__kk

:_D__gg
GETFP GETSP SETFP
GETFP 2 ADD SETRV __D__gg JMP
:__D__gg
GETFP SETSP SETFP
1 RETN

:_D__kk
GETFP GETSP SETFP
GETFP 2 ADD SETRV __D__kk JMP
:__D__kk
GETFP SETSP SETFP
1 RETN
```

```
:_D_vtbl__
_B_vtbl__ _B__ff _D__gg _B__hh _D__kk

:_E__gg
GETFP GETSP SETFP
GETFP 2 ADD SETRV __E__gg JMP
:__E__gg
GETFP SETSP SETFP
1 RETN

:_E__ff
GETFP GETSP SETFP
GETFP 2 ADD SETRV __E__ff JMP
:__E__ff
GETFP SETSP SETFP
1 RETN

:_E_vtbl__
_D_vtbl__ _E__ff _E__gg _B__hh _D__kk

:_main
GETFP GETSP SETFP
GETFP GETSP SETFP
5 PUSHN
GETFP 2 SUB _Animal_vtbl__ SAVE
GETFP 5 SUB _Cat_vtbl__ SAVE
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB GETFP LOAD 2 SUB SAVE
1501 GETFP 1 SUB LOAD GETFP 1 SUB LOAD LOAD 1 ADD 0 ADD LOAD CALL GETRV SETRV GETFP SETSP SETFP
GETRV DROP
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB GETFP LOAD 2 SUB SAVE
GETFP 1 SUB LOAD GETFP 1 SUB LOAD LOAD 1 ADD 1 ADD LOAD CALL GETRV SETRV GETFP SETSP SETFP
GETRV _printInt CALL GETRV DROP
_newLine CALL GETRV DROP
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB GETFP LOAD 2 SUB SAVE
GETFP 1 SUB LOAD GETFP 1 SUB LOAD LOAD 1 ADD 2 ADD LOAD CALL GETRV SETRV GETFP SETSP SETFP
GETRV _printInt CALL GETRV DROP
_newLine CALL GETRV DROP
GETFP GETSP SETFP
1 PUSHN
```

```
GETFP 1 SUB GETFP LOAD 5 SUB SAVE
100500 GETFP 1 SUB LOAD GETFP 1 SUB LOAD LOAD 1 ADD 0 ADD LOAD CALL GETRV SETRV GETFP SETSP SETFP
GETRV DROP
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB GETFP LOAD 5 SUB SAVE
1007 GETFP 1 SUB LOAD GETFP 1 SUB LOAD LOAD 1 ADD 3 ADD LOAD CALL GETRV SETRV GETFP SETSP SETFP
GETRV DROP
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB GETFP LOAD 5 SUB SAVE
GETFP 1 SUB LOAD GETFP 1 SUB LOAD LOAD 1 ADD 1 ADD LOAD CALL GETRV SETRV GETFP SETSP SETFP
GETRV _printInt CALL GETRV DROP
_newLine CALL GETRV DROP
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB GETFP LOAD 5 SUB SAVE
GETFP 1 SUB LOAD GETFP 1 SUB LOAD LOAD 1 ADD 4 ADD LOAD CALL GETRV SETRV GETFP SETSP SETFP
GETRV _printInt CALL GETRV DROP
_newLine CALL GETRV DROP
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB GETFP LOAD 5 SUB SAVE
GETFP 1 SUB LOAD GETFP 1 SUB LOAD LOAD 1 ADD 2 ADD LOAD CALL GETRV SETRV GETFP SETSP SETFP
GETRV _printInt CALL GETRV DROP
_newLine CALL GETRV DROP
GETFP GETSP SETFP
2 PUSHN
GETFP 1 SUB GETFP LOAD 2 SUB LOAD SAVE
:_loop_while_0_main
GETFP 1 SUB LOAD
0
CMP _bool_0_main JNE
_exit_while_0_main JMP
:_bool_0_main
GETFP 1 SUB LOAD
_Animal_vtbl__
CMP _true_while_0_main JNE
_exit_while_0_main JMP

:_true_while_0_main
GETFP 1 SUB GETFP 1 SUB LOAD LOAD SAVE
_loop_while_0_main JMP
:_exit_while_0_main
GETFP 1 SUB LOAD
0
```

```
CMP _true_if_1_main JNE
_false_if_1_main JMP
:_true_if_1_main
GETFP 2 SUB 1 SAVE
_exit_if_1_main JMP
:_false_if_1_main
GETFP 2 SUB 0 SAVE
:_exit_if_1_main
GETFP 2 SUB LOAD SETRV GETFP SETSP SETFP
GETRV
1
CMP _exit_if_0_main JEQ
_true_if_0_main JMP
:_true_if_0_main
6661 _printInt CALL GETRV DROP
_newLine CALL GETRV DROP
:_exit_if_0_main
GETFP GETSP SETFP
2 PUSHN
GETFP 1 SUB GETFP LOAD 5 SUB LOAD SAVE
:_loop_while_1_main
GETFP 1 SUB LOAD
0
CMP _bool_1_main JNE
_exit_while_1_main JMP
:_bool_1_main
GETFP 1 SUB LOAD
_Animal_vtbl__
CMP _true_while_1_main JNE
_exit_while_1_main JMP

:_true_while_1_main
GETFP 1 SUB GETFP 1 SUB LOAD LOAD SAVE
_loop_while_1_main JMP
:_exit_while_1_main
GETFP 1 SUB LOAD
0
CMP _true_if_3_main JNE
_false_if_3_main JMP
:_true_if_3_main
GETFP 2 SUB 1 SAVE
_exit_if_3_main JMP
:_false_if_3_main
GETFP 2 SUB 0 SAVE
:_exit_if_3_main
GETFP 2 SUB LOAD SETRV GETFP SETSP SETFP
```

```
GETRV
1
CMP _exit_if_2_main JEQ
_true_if_2_main JMP
:_true_if_2_main
6662 _printInt CALL GETRV DROP
_newLine CALL GETRV DROP
:_exit_if_2_main
GETFP GETSP SETFP
2 PUSHN
GETFP 1 SUB GETFP LOAD 2 SUB LOAD SAVE
:_loop_while_2_main
GETFP 1 SUB LOAD
0
CMP _bool_2_main JNE
_exit_while_2_main JMP
:_bool_2_main
GETFP 1 SUB LOAD
_Cat_vtbl__
CMP _true_while_2_main JNE
_exit_while_2_main JMP

:_true_while_2_main
GETFP 1 SUB GETFP 1 SUB LOAD LOAD SAVE
_loop_while_2_main JMP
:_exit_while_2_main
GETFP 1 SUB LOAD
0
CMP _true_if_5_main JNE
_false_if_5_main JMP
:_true_if_5_main
GETFP 2 SUB 1 SAVE
_exit_if_5_main JMP
:_false_if_5_main
GETFP 2 SUB 0 SAVE
:_exit_if_5_main
GETFP 2 SUB LOAD SETRV GETFP SETSP SETFP
GETRV
1
CMP _true_if_4_main JEQ
_exit_if_4_main JMP
:_true_if_4_main
6663 _printInt CALL GETRV DROP
_newLine CALL GETRV DROP
:_exit_if_4_main
GETFP GETSP SETFP
```

```
2 PUSHN
GETFP 1 SUB GETFP LOAD 5 SUB LOAD SAVE
:_loop_while_3_main
GETFP 1 SUB LOAD
0
CMP _bool_3_main JNE
_exit_while_3_main JMP
:_bool_3_main
GETFP 1 SUB LOAD
_Cat_vtbl__
CMP _true_while_3_main JNE
_exit_while_3_main JMP

:_true_while_3_main
GETFP 1 SUB GETFP 1 SUB LOAD LOAD SAVE
_loop_while_3_main JMP
:_exit_while_3_main
GETFP 1 SUB LOAD
0
CMP _true_if_7_main JNE
_false_if_7_main JMP
:_true_if_7_main
GETFP 2 SUB 1 SAVE
_exit_if_7_main JMP
:_false_if_7_main
GETFP 2 SUB 0 SAVE
:_exit_if_7_main
GETFP 2 SUB LOAD SETRV GETFP SETSP SETFP
GETRV
1
CMP _exit_if_6_main JEQ
_true_if_6_main JMP
:_true_if_6_main
6664 _printInt CALL GETRV DROP
_newLine CALL GETRV DROP
:_exit_if_6_main
GETFP SETSP SETFP
0 SETRV __main JMP
:__main
GETFP SETSP SETFP
JMP
```

cmp.sh

```
rlmake.bat main.ref -o a.exe
```

```
./a.exe source.txt a.asm
rm a.exe
rm *.rasl
```

int.sh

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

int.v

```
// константы
`define MAX_LEXEM_COUNT 2048
`define MAX_LEXEM_SIZE  32
`define MAX_LABELS      512
`define MEMMORY_LIMIT   1_000_000

module main;
    // имена файлов
    localparam targerProgramFile    = "a.asm";
    localparam stdinFile            = "stdin.txt";

    // лексемы
   reg [7:0]  lexemsArray[0:`MAX_LEXEM_COUNT-1][0:`MAX_LEXEM_SIZE-
1];
    integer     lexemsSizesArray[0:`MAX_LEXEM_COUNT-1];
    integer     lexemsCount = 0;

    // метки
    reg [7:0]      labelsArray[0:`MAX_LABELS-1][0:`MAX_LEXEM_SIZE-
1];
    integer         labelsSizesArray[0:`MAX_LABELS-1];
    integer signed  labelsValuesArray[0:`MAX_LABELS-1];
    integer         labelsCount = 0;

    // фактическая память машины
    integer signed  memmory[0:`MEMMORY_LIMIT-1];    // память машины
    integer        program_size;  // размер программы в памяти машины
   integer      reg_IP;    // указатель инструкций (instruction pointer)
    integer         reg_SP;     // указатель стека (stack pointer)
    integer signed  reg_FP;     // указатель базы (frame pointer)
   integer signed  reg_RV;    // возвращаемое значение (return value)
```

```
    // вспомогательные переменные при работе со стеком
    integer signed x, y, z, N, a, v;

    // ошибка, выдаваемая при недостаточном размере
    //  стека при выпонении некоторой операции
    localparam STACK_ERROR =
        "RuntimeError: line~%d: %s: In stack must \
be at least %d element, but %d found.\n";

    // ASCII-коды нужных сиволов
    localparam CHAR_TAB         = 9;
    localparam CHAR_NEWLINE     = 10;
    localparam CHAR_SPACE       = 32;
    localparam CHAR_PLUS        = 43;
    localparam CHAR_MINUS       = 45;
    localparam CHAR_HYPHEN      = 45;
    localparam CHAR_0           = 48;
    localparam CHAR_9           = 57;
    localparam CHAR_COLON       = 58;
    localparam CHAR_SEMICOLON   = 59;
    localparam CHAR_A           = 65;
    localparam CHAR_B           = 66;
    localparam CHAR_C           = 67;
    localparam CHAR_D           = 68;
    localparam CHAR_E           = 69;
    localparam CHAR_F           = 70;
    localparam CHAR_G           = 71;
    localparam CHAR_H           = 72;
    localparam CHAR_I           = 73;
    localparam CHAR_J           = 74;
    localparam CHAR_K           = 75;
    localparam CHAR_L           = 76;
    localparam CHAR_M           = 77;
    localparam CHAR_N           = 78;
    localparam CHAR_O           = 79;
    localparam CHAR_P           = 80;
    localparam CHAR_Q           = 81;
    localparam CHAR_R           = 82;
    localparam CHAR_S           = 83;
    localparam CHAR_T           = 84;
    localparam CHAR_U           = 85;
    localparam CHAR_V           = 86;
    localparam CHAR_W           = 87;
    localparam CHAR_X           = 88;
    localparam CHAR_Y           = 89;
    localparam CHAR_Z           = 90;
    localparam CHAR_UNDERSCORE  = 95;
    localparam CHAR_a           = 97;
    localparam CHAR_z           = 122;

    // коды встроенных команд
    localparam CMD_ADD          = -1;
    localparam CMD_SUB          = -2;
    localparam CMD_MUL          = -3;
    localparam CMD_DIV          = -4;
    localparam CMD_MOD          = -5;
    localparam CMD_NEG          = -6;
    localparam CMD_BITAND       = -7;
    localparam CMD_BITOR        = -8;
    localparam CMD_BITNOT       = -9;
    localparam CMD_LSHIFT       = -10;
    localparam CMD_RSHIFT       = -11;
    localparam CMD_DUP          = -12;
    localparam CMD_DROP         = -13;
    localparam CMD_SWAP         = -14;
    localparam CMD_ROT          = -15;
    localparam CMD_OVER         = -16;
    localparam CMD_DROPN        = -17;
    localparam CMD_PUSHN        = -18;
    localparam CMD_LOAD         = -19;
    localparam CMD_SAVE         = -20;
    localparam CMD_GETIP        = -21;
    localparam CMD_SETIP        = -22;
    localparam CMD_GETSP        = -23;
    localparam CMD_SETSP        = -24;
    localparam CMD_GETFP        = -25;
    localparam CMD_SETFP        = -26;
    localparam CMD_GETRV        = -27;
    localparam CMD_SETRV        = -28;
    localparam CMD_CMP          = -29;
    localparam CMD_JMP          = -22;
    localparam CMD_JLT          = -30;
    localparam CMD_JGT          = -31;
    localparam CMD_JEQ          = -32;
    localparam CMD_JLE          = -33;
    localparam CMD_JGE          = -34;
    localparam CMD_JNE          = -35;
    localparam CMD_CALL         = -36;
    localparam CMD_RETN         = -37;
    localparam CMD_IN           = -38;
    localparam CMD_OUT          = -39;
    localparam CMD_HALT         = -40;
```

```
    // вспомогательные переменные
    integer fd, i, j, k, current;
    reg [7:0] char;
    reg isComment, isEqual, isNegativ, wasFound;

    initial begin

        // <<<<<===== первый этап - парсинг =====>>>>>
        lexemsSizesArray[0] = 0;  // обнуляем размер первой лексемы

        // открытие файла программы
        fd = $fopen(targerProgramFile, "r");
        // проверка на существование файла
        if (!fd) begin
        $write("FileError: File '%s' unexists.\n", targerProgramFile);
            $finish(1);
        end

        // чтение файла
        while (!$feof(fd)) begin
            char = $fgetc(fd);

           if (isComment == 1) begin // игнорируем, если это коммент
            end else if (  // завершаем считывание прошлой лексемы
                char == CHAR_SEMICOLON  ||
                char == CHAR_NEWLINE    ||
                char == CHAR_TAB        ||
                char == CHAR_SPACE
            ) begin
                if (lexemsSizesArray[lexemsCount] != 0) begin
                    ++lexemsCount;
                    lexemsSizesArray[lexemsCount] = 0;
                end
            end else begin  // следующий символ лексемы

                // проверка на переполнение массива лексем
                if (lexemsCount >= `MAX_LEXEM_COUNT) begin
              $write("SyntaxError: Too many lexems. Max lexem count -
%d.\n",
                                                `MAX_LEXEM_COUNT);
                    $finish(1);
                end

                // проверка на превышение размера лексем
           if (lexemsSizesArray[lexemsCount] >= `MAX_LEXEM_SIZE) begin
```

```
                $write("SyntaxError: Too long lexem - ");
        for (i = 0; i != lexemsSizesArray[lexemsCount]; ++i) begin
                    $write("%s", lexemsArray[lexemsCount][i]);
                end
            $write(". Max lexem size - %d.\n", `MAX_LEXEM_SIZE);
                $finish(1);
            end

            // приписываем последней лексеме новый символ
        lexemsArray[lexemsCount][lexemsSizesArray[lexemsCount]] = char;
            ++lexemsSizesArray[lexemsCount];
        end

        if (char == CHAR_SEMICOLON) begin  // начало коммента
            isComment = 1;
      end else if (char == CHAR_NEWLINE) begin  // конец коммента
            isComment = 0;
        end
    end
 // при чтении файла считывается символ с кодом 255. Странно. Удаляем.
    --lexemsSizesArray[lexemsCount];
    // если чтение последней лексемы не закончено, заканчиваем
    if (lexemsSizesArray[lexemsCount] > 0) begin
        ++lexemsCount;
    end
    $fclose(fd);  // закрыаем файл

    // // <<<<<======= ДЕБАГ: печать массива считанных лексем
    // for (i = 0; i != lexemsCount; ++i) begin
    //     for (j = 0; j != lexemsSizesArray[i]; ++j) begin
    //         $write("%s", lexemsArray[i][j]);
    //     end
    //     $write("\n---\n");
    // end


    // program_size по умолчанию имеет индекс 0 в массиве.
  // Остальные мнемотики и метки не привязаны к индексам массива.
    labelsArray[0][0] = CHAR_P;
    labelsArray[0][1] = CHAR_R;
    labelsArray[0][2] = CHAR_O;
    labelsArray[0][3] = CHAR_G;
    labelsArray[0][4] = CHAR_R;
    labelsArray[0][5] = CHAR_A;
    labelsArray[0][6] = CHAR_M;
    labelsArray[0][7] = CHAR_UNDERSCORE;
```

```
labelsArray[0][8] = CHAR_S;
labelsArray[0][9] = CHAR_I;
labelsArray[0][10] = CHAR_Z;
labelsArray[0][11] = CHAR_E;
labelsSizesArray[0] = 12;
labelsCount = 1;

// 41 встроенная мнемотика......................
labelsArray[labelsCount][0] = CHAR_A;
labelsArray[labelsCount][1] = CHAR_D;
labelsArray[labelsCount][2] = CHAR_D;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_ADD;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_S;
labelsArray[labelsCount][1] = CHAR_U;
labelsArray[labelsCount][2] = CHAR_B;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_SUB;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_M;
labelsArray[labelsCount][1] = CHAR_U;
labelsArray[labelsCount][2] = CHAR_L;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_MUL;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_D;
labelsArray[labelsCount][1] = CHAR_I;
labelsArray[labelsCount][2] = CHAR_V;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_DIV;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_M;
labelsArray[labelsCount][1] = CHAR_O;
labelsArray[labelsCount][2] = CHAR_D;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_MOD;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_N;
labelsArray[labelsCount][1] = CHAR_E;
labelsArray[labelsCount][2] = CHAR_G;
```

```
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_NEG;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_B;
labelsArray[labelsCount][1] = CHAR_I;
labelsArray[labelsCount][2] = CHAR_T;
labelsArray[labelsCount][3] = CHAR_A;
labelsArray[labelsCount][4] = CHAR_N;
labelsArray[labelsCount][5] = CHAR_D;
labelsSizesArray[labelsCount] = 6;
labelsValuesArray[labelsCount] = CMD_BITAND;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_B;
labelsArray[labelsCount][1] = CHAR_I;
labelsArray[labelsCount][2] = CHAR_T;
labelsArray[labelsCount][3] = CHAR_O;
labelsArray[labelsCount][4] = CHAR_R;
labelsSizesArray[labelsCount] = 5;
labelsValuesArray[labelsCount] = CMD_BITOR;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_B;
labelsArray[labelsCount][1] = CHAR_O;
labelsArray[labelsCount][2] = CHAR_T;
labelsArray[labelsCount][3] = CHAR_N;
labelsArray[labelsCount][4] = CHAR_O;
labelsArray[labelsCount][5] = CHAR_T;
labelsSizesArray[labelsCount] = 6;
labelsValuesArray[labelsCount] = CMD_BITNOT;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_L;
labelsArray[labelsCount][1] = CHAR_S;
labelsArray[labelsCount][2] = CHAR_H;
labelsArray[labelsCount][3] = CHAR_I;
labelsArray[labelsCount][4] = CHAR_F;
labelsArray[labelsCount][5] = CHAR_T;
labelsSizesArray[labelsCount] = 6;
labelsValuesArray[labelsCount] = CMD_LSHIFT;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_R;
labelsArray[labelsCount][1] = CHAR_S;
labelsArray[labelsCount][2] = CHAR_H;
```

```
labelsArray[labelsCount][3] = CHAR_I;
labelsArray[labelsCount][4] = CHAR_F;
labelsArray[labelsCount][5] = CHAR_T;
labelsSizesArray[labelsCount] = 6;
labelsValuesArray[labelsCount] = CMD_RSHIFT;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_D;
labelsArray[labelsCount][1] = CHAR_U;
labelsArray[labelsCount][2] = CHAR_P;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_DUP;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_D;
labelsArray[labelsCount][1] = CHAR_R;
labelsArray[labelsCount][2] = CHAR_O;
labelsArray[labelsCount][3] = CHAR_P;
labelsSizesArray[labelsCount] = 4;
labelsValuesArray[labelsCount] = CMD_DROP;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_S;
labelsArray[labelsCount][1] = CHAR_W;
labelsArray[labelsCount][2] = CHAR_A;
labelsArray[labelsCount][3] = CHAR_P;
labelsSizesArray[labelsCount] = 4;
labelsValuesArray[labelsCount] = CMD_SWAP;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_R;
labelsArray[labelsCount][1] = CHAR_O;
labelsArray[labelsCount][2] = CHAR_T;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_ROT;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_O;
labelsArray[labelsCount][1] = CHAR_V;
labelsArray[labelsCount][2] = CHAR_E;
labelsArray[labelsCount][3] = CHAR_R;
labelsSizesArray[labelsCount] = 4;
labelsValuesArray[labelsCount] = CMD_OVER;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_D;
```

```
labelsArray[labelsCount][1] = CHAR_R;
labelsArray[labelsCount][2] = CHAR_O;
labelsArray[labelsCount][3] = CHAR_P;
labelsArray[labelsCount][4] = CHAR_N;
labelsSizesArray[labelsCount] = 5;
labelsValuesArray[labelsCount] = CMD_DROPN;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_P;
labelsArray[labelsCount][1] = CHAR_U;
labelsArray[labelsCount][2] = CHAR_S;
labelsArray[labelsCount][3] = CHAR_H;
labelsArray[labelsCount][4] = CHAR_N;
labelsSizesArray[labelsCount] = 5;
labelsValuesArray[labelsCount] = CMD_PUSHN;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_L;
labelsArray[labelsCount][1] = CHAR_O;
labelsArray[labelsCount][2] = CHAR_A;
labelsArray[labelsCount][3] = CHAR_D;
labelsSizesArray[labelsCount] = 4;
labelsValuesArray[labelsCount] = CMD_LOAD;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_S;
labelsArray[labelsCount][1] = CHAR_A;
labelsArray[labelsCount][2] = CHAR_V;
labelsArray[labelsCount][3] = CHAR_E;
labelsSizesArray[labelsCount] = 4;
labelsValuesArray[labelsCount] = CMD_SAVE;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_G;
labelsArray[labelsCount][1] = CHAR_E;
labelsArray[labelsCount][2] = CHAR_T;
labelsArray[labelsCount][3] = CHAR_I;
labelsArray[labelsCount][4] = CHAR_P;
labelsSizesArray[labelsCount] = 5;
labelsValuesArray[labelsCount] = CMD_GETIP;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_S;
labelsArray[labelsCount][1] = CHAR_E;
labelsArray[labelsCount][2] = CHAR_T;
labelsArray[labelsCount][3] = CHAR_I;
```

```
labelsArray[labelsCount][4] = CHAR_P;
labelsSizesArray[labelsCount] = 5;
labelsValuesArray[labelsCount] = CMD_SETIP;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_G;
labelsArray[labelsCount][1] = CHAR_E;
labelsArray[labelsCount][2] = CHAR_T;
labelsArray[labelsCount][3] = CHAR_S;
labelsArray[labelsCount][4] = CHAR_P;
labelsSizesArray[labelsCount] = 5;
labelsValuesArray[labelsCount] = CMD_GETSP;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_S;
labelsArray[labelsCount][1] = CHAR_E;
labelsArray[labelsCount][2] = CHAR_T;
labelsArray[labelsCount][3] = CHAR_S;
labelsArray[labelsCount][4] = CHAR_P;
labelsSizesArray[labelsCount] = 5;
labelsValuesArray[labelsCount] = CMD_SETSP;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_G;
labelsArray[labelsCount][1] = CHAR_E;
labelsArray[labelsCount][2] = CHAR_T;
labelsArray[labelsCount][3] = CHAR_F;
labelsArray[labelsCount][4] = CHAR_P;
labelsSizesArray[labelsCount] = 5;
labelsValuesArray[labelsCount] = CMD_GETFP;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_S;
labelsArray[labelsCount][1] = CHAR_E;
labelsArray[labelsCount][2] = CHAR_T;
labelsArray[labelsCount][3] = CHAR_F;
labelsArray[labelsCount][4] = CHAR_P;
labelsSizesArray[labelsCount] = 5;
labelsValuesArray[labelsCount] = CMD_SETFP;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_G;
labelsArray[labelsCount][1] = CHAR_E;
labelsArray[labelsCount][2] = CHAR_T;
labelsArray[labelsCount][3] = CHAR_R;
labelsArray[labelsCount][4] = CHAR_V;
```

```
labelsSizesArray[labelsCount] = 5;
labelsValuesArray[labelsCount] = CMD_GETRV;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_S;
labelsArray[labelsCount][1] = CHAR_E;
labelsArray[labelsCount][2] = CHAR_T;
labelsArray[labelsCount][3] = CHAR_R;
labelsArray[labelsCount][4] = CHAR_V;
labelsSizesArray[labelsCount] = 5;
labelsValuesArray[labelsCount] = CMD_SETRV;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_C;
labelsArray[labelsCount][1] = CHAR_M;
labelsArray[labelsCount][2] = CHAR_P;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_CMP;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_J;
labelsArray[labelsCount][1] = CHAR_M;
labelsArray[labelsCount][2] = CHAR_P;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_JMP;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_J;
labelsArray[labelsCount][1] = CHAR_L;
labelsArray[labelsCount][2] = CHAR_T;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_JLT;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_J;
labelsArray[labelsCount][1] = CHAR_G;
labelsArray[labelsCount][2] = CHAR_T;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_JGT;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_J;
labelsArray[labelsCount][1] = CHAR_E;
labelsArray[labelsCount][2] = CHAR_Q;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_JEQ;
```

```
++labelsCount;

labelsArray[labelsCount][0] = CHAR_J;
labelsArray[labelsCount][1] = CHAR_L;
labelsArray[labelsCount][2] = CHAR_E;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_JLE;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_J;
labelsArray[labelsCount][1] = CHAR_G;
labelsArray[labelsCount][2] = CHAR_E;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_JGE;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_J;
labelsArray[labelsCount][1] = CHAR_N;
labelsArray[labelsCount][2] = CHAR_E;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_JNE;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_C;
labelsArray[labelsCount][1] = CHAR_A;
labelsArray[labelsCount][2] = CHAR_L;
labelsArray[labelsCount][3] = CHAR_L;
labelsSizesArray[labelsCount] = 4;
labelsValuesArray[labelsCount] = CMD_CALL;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_R;
labelsArray[labelsCount][1] = CHAR_E;
labelsArray[labelsCount][2] = CHAR_T;
labelsArray[labelsCount][3] = CHAR_N;
labelsSizesArray[labelsCount] = 4;
labelsValuesArray[labelsCount] = CMD_RETN;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_I;
labelsArray[labelsCount][1] = CHAR_N;
labelsSizesArray[labelsCount] = 2;
labelsValuesArray[labelsCount] = CMD_IN;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_O;
```

```
   labelsArray[labelsCount][1] = CHAR_U;
   labelsArray[labelsCount][2] = CHAR_T;
   labelsSizesArray[labelsCount] = 3;
   labelsValuesArray[labelsCount] = CMD_OUT;
   ++labelsCount;

   labelsArray[labelsCount][0] = CHAR_H;
   labelsArray[labelsCount][1] = CHAR_A;
   labelsArray[labelsCount][2] = CHAR_L;
   labelsArray[labelsCount][3] = CHAR_T;
   labelsSizesArray[labelsCount] = 4;
   labelsValuesArray[labelsCount] = CMD_HALT;
   ++labelsCount;


// <<<<<===== второй этап - анализ (проходы) и исполнение =====>>>>>

   // первый проход ==========>
   current = 256;  // указание
   for (i = 0; i != lexemsCount; ++i) begin
       if (lexemsArray[i][0] == CHAR_COLON) begin

           // проверка синтаксиса метки
           // проверка первого символа идентификатора
           if (
               lexemsSizesArray[i] < 2 ||
               !(
                   CHAR_a <= lexemsArray[i][1] &&
                             lexemsArray[i][1] <= CHAR_z ||
                   CHAR_A <= lexemsArray[i][1] &&
                             lexemsArray[i][1] <= CHAR_Z ||
                   lexemsArray[i][1] == CHAR_UNDERSCORE
               )
           ) begin
              $write("SyntaxError: Invalid label syntax - '");
              for (j = 0; j != lexemsSizesArray[i]; ++j) begin
                   $write("%s", lexemsArray[i][j]);
               end
         $write("'. Must be - ':[a-zA-Z_][a-zA-Z0-9_-]*'.\n");
               $finish(1);
           end
           // проверка остальных символов идентификатора
           for (j = 2; j != lexemsSizesArray[i]; ++j) begin
               if (!(
                   CHAR_a <= lexemsArray[i][j] &&
                             lexemsArray[i][j] <= CHAR_z  ||
                    CHAR_A <= lexemsArray[i][j] &&
                              lexemsArray[i][j] <= CHAR_Z  ||
                    CHAR_0 <= lexemsArray[i][j] &&
                              lexemsArray[i][j] <= CHAR_9  ||
                lexemsArray[i][j] == CHAR_UNDERSCORE                ||
                    lexemsArray[i][j] == CHAR_HYPHEN
                  )) begin
                 $write("SyntaxError: Invalid label syntax - '");
                  for (k = 0; k != lexemsSizesArray[i]; ++k) begin
                      $write("%s", lexemsArray[i][k]);
                    end
                     $write("'. Must be - ':[a-zA-Z_][a-zA-Z0-
9_-]*'.\n");
                    $finish(1);
                  end
                end
                // проверка уникальности метки
                for (j = 0; j != labelsCount; ++j) begin
               if (lexemsSizesArray[i] == labelsSizesArray[j]) begin
                    isEqual = 1;
                  for (k = 0; k != lexemsSizesArray[i]; ++k) begin
                 if (lexemsArray[i][k] != labelsArray[j][k]) begin
                        isEqual = 0;
                       k = lexemsSizesArray[i] - 1;  // break
                      end
                    end
                    if (isEqual == 1) begin
                  $write("SyntaxError: This label yet exists - ");
                   for (k = 0; k != lexemsSizesArray[i]; ++k) begin
                        $write("%s", lexemsArray[i][k]);
                      end
                      $write(".\n");
                      $finish(1);
                    end
                  end
                end
                // проверка на переполнение массива меток
                if (labelsCount == `MAX_LABELS) begin
                  $write("SyntaxError: Too many labels. Max size -
%d.\n",
                                                      `MAX_LABELS);
                  $finish(1);
                end

                // переписываем метку
           labelsSizesArray[labelsCount] = lexemsSizesArray[i] - 1;
```

```
            for (j = 1; j != lexemsSizesArray[i]; ++j) begin
           labelsArray[labelsCount][j - 1] = lexemsArray[i][j];
            end
            // сохраняем адрес памяти
            labelsValuesArray[labelsCount] = current;
            ++labelsCount;
      --current;  // не учитываем метки при подсчете слов программы
       end
       ++current;
    end
    // записываем значение переменной PROGRAM_SIZE
    program_size = current;
    labelsValuesArray[0] = program_size;

 // // <<<<<======= ДЕБАГ: печать массива сохраненный мнемотик и меток
    // for (i = 0; i != labelsCount; ++i) begin
    //     for (j = 0; j != labelsSizesArray[i]; ++j) begin
    //         $write("%s", labelsArray[i][j]);
    //     end
    //     $write(": value-%d\n", labelsValuesArray[i]);
    // end

    // второй проход ==========>
    current = 256;
    for (i = 0; i != lexemsCount; ++i) begin

        // игнорируем метки
        if (lexemsArray[i][0] == CHAR_COLON) begin
            --current;  // не оставляем дыр в памяти от меток
        end else if (  // распознаем первый симол числа
            lexemsArray[i][0] == CHAR_PLUS  ||
            lexemsArray[i][0] == CHAR_MINUS ||
            CHAR_0 <= lexemsArray[i][0] &&
                      lexemsArray[i][0] <= CHAR_9
        ) begin
            isNegativ = 0;
            j = 0;
            if (lexemsArray[i][0] == CHAR_MINUS) begin
                isNegativ = 1;
                j = 1;
            end else if (lexemsArray[i][0] == CHAR_PLUS) begin
                j = 1;
            end
            // пустая последовательность десятичным цифр
            if (j == lexemsSizesArray[i]) begin
                $write("SyntaxError: Invalid digit - '");
                for (k = 0; k != lexemsSizesArray[i]; ++k) begin
                    $write("%s", lexemsArray[i][k]);
                 end
                 $write("'. Must be - '[+-]?[0-9]+'.\n");
                 $finish(1);
            end
            // преобразуем строку в число
            memmory[current] = 0;
            for (j = j; j != lexemsSizesArray[i]; ++j) begin
                memmory[current] *= 10;
              memmory[current] += (lexemsArray[i][j] - CHAR_0);
            end
            if (isNegativ == 1) begin
                memmory[current] *= -1;
            end
      end else if (  // распознаем первый символ идентификатора
            CHAR_a <= lexemsArray[i][0] &&
                      lexemsArray[i][0] <= CHAR_z ||
            CHAR_A <= lexemsArray[i][0] &&
                      lexemsArray[i][0] <= CHAR_Z ||
            lexemsArray[i][0] == CHAR_UNDERSCORE
        ) begin
            // распознаем остальные симолы идентификатора
            for (j = 1; j != lexemsSizesArray[i]; ++j) begin
                if (!(
                    CHAR_a <= lexemsArray[i][j] &&
                              lexemsArray[i][j] <= CHAR_z  ||
                    CHAR_A <= lexemsArray[i][j] &&
                              lexemsArray[i][j] <= CHAR_Z  ||
                    CHAR_0 <= lexemsArray[i][j] &&
                              lexemsArray[i][j] <= CHAR_9  ||
            lexemsArray[i][j] == CHAR_UNDERSCORE                ||
                    lexemsArray[i][j] == CHAR_HYPHEN
                )) begin
            $write("SyntaxError: Invalid identificator syntax -
'");
                for (k = 0; k != lexemsSizesArray[i]; ++k) begin
                        $write("%s", lexemsArray[i][k]);
                    end
                     $write("'. Must be - ':[a-zA-Z_][a-zA-Z0-
9_-]*'.\n");
                    $finish(1);
                end
            end

            // ищем среди записанных мнемоник и меток нужную
```

```
                wasFound = 0;
                for (j = 0; j != labelsCount; ++j) begin
               if (lexemsSizesArray[i] == labelsSizesArray[j]) begin
                        isEqual = 1;
                    for (k = 0; k != lexemsSizesArray[i]; ++k) begin
                   if (lexemsArray[i][k] != labelsArray[j][k]) begin
                                isEqual = 0;
                             k = lexemsSizesArray[i] - 1;  // break
                            end
                        end
                        if (isEqual == 1) begin
                            wasFound = 1;
                            // подставляем мнемотику или метку
                          memmory[current] = labelsValuesArray[j];
                            j = labelsCount - 1;  // break
                        end
                    end
                end
                // ошибка, если идентификатор не найден
                if (wasFound == 0) begin
               $write("SyntaxError: Identificator not exists - '");
                   for (j = 0; j != lexemsSizesArray[i]; ++j) begin
                        $write("%s", lexemsArray[i][j]);
                    end
                    $write("'.\n");
                    $finish(1);
                end
            end else begin  // лексема не распознана
               $write("SyntaxError: Lexem did not recognized - '");
                for (j = 0; j != lexemsSizesArray[i]; ++j) begin
                    $write("%s", lexemsArray[i][j]);
                end
                $write("'. Must be digit - '[+-]?[0-9]+' ");
                 $write("or identificator - '[a-zA-Z_][a-zA-Z0-9_-
]*'.\n");
                $finish(1);
            end

        // // <<<<<======= ДЕБАГ: итеративная печать исходной лексемы и
        // //          соответсвующего ей записанного в память числа
            // for (j = 0; j != lexemsSizesArray[i]; ++j) begin
            //     $write("%s", lexemsArray[i][j]);
            // end
            // $write("\t%d\n", memmory[current]);

            ++current;
        end


        // инциализация состояния старта виртуальной машины
                                    // начиная с адреса 256 загружены
                              //       слова пользовательской программы
      reg_IP = 256;           // будет выполнена самая первая инструкция
      reg_SP = `MEMMORY_LIMIT;   // прочитать со стека ничего нельзя, записать можно
                              // значение регистров FP и RV не определено


        // открываем файл с входными данными
        fd = $fopen(stdinFile, "r");
        if (!fd) begin
            $write("FileError: File '%s' unexists.\n", stdinFile);
            $finish(1);
        end

        // главный исполняющий цикл машины
        forever begin
        // // <<<<<======= ДЕБАГ: печать выполняемой команды и состояния стека
                    //  $write("lexem~%d\tcmd=%d  |  ",  (reg_IP  -
256), memmory[reg_IP]);
          // for (k = `MEMMORY_LIMIT - 1; k != reg_SP - 1; --k) begin
            //     $write("%d ", memmory[k]);
            // end
            // $write(" ...\n");

            // проверка на валидность значения регистра IP
         if (!(256 <= reg_IP && reg_IP < (256 + program_size))) begin
           $write("RuntimeError: register IP must be between 256 and %d.",
                                            (256 + program_size));
                $write(" Your register IP - %d.\n", reg_IP);
                $finish(1);
            end
            // проверка на переполненность стека ;)
            if (!((256 + program_size) <= reg_SP)) begin
           $write("RuntimeError: line~%d: stack overflow.\n", (reg_IP -
256));
                $finish(1);
            end
            // проверка на валидность значения регистра SP
        //    (по идее, никогда не пригодится, достаточно прочих проверок)
            if (!(reg_SP <= `MEMMORY_LIMIT)) begin
           $write("RuntimeError: line~%d: register SP must be between",
                                                  (reg_IP - 256));
```

```
   $write(" %d and %d.", (256 + program_size), `MEMMORY_LIMIT);
        $write(" Your register SP - %d.\n", reg_SP);
        $finish(1);
    end

// выбор соответсвующей иструкции по следующему слову в памяти машины
    case (memmory[reg_IP])

        CMD_ADD : begin
            // проверка на возможность считывания
            //      нужного количества слов со стека
            if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
                $write(STACK_ERROR, (reg_IP - 256),
                     "ADD", 2, (`MEMMORY_LIMIT - reg_SP));
                $finish(1);
            end

            // считываем со стека два слова (числа)
            y = memmory[reg_SP++];
            x = memmory[reg_SP++];
          // помещаем на вершину стека результат вычислений
            memmory[--reg_SP] = x + y;
            // переходим к следующей инструкции
            ++reg_IP;
        end

        CMD_SUB : begin
            if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
                $write(STACK_ERROR, (reg_IP - 256),
                     "SUB", 2, (`MEMMORY_LIMIT - reg_SP));
                $finish(1);
            end

            y = memmory[reg_SP++];
            x = memmory[reg_SP++];
            memmory[--reg_SP] = x - y;
            ++reg_IP;
        end

        CMD_MUL : begin
            if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
                $write(STACK_ERROR, (reg_IP - 256),
                     "MUL", 2, (`MEMMORY_LIMIT - reg_SP));
                $finish(1);
            end
            y = memmory[reg_SP++];
            x = memmory[reg_SP++];
            memmory[--reg_SP] = x * y;
            ++reg_IP;
        end

        CMD_DIV : begin
            if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
                $write(STACK_ERROR, (reg_IP - 256),
                     "DIV", 2, (`MEMMORY_LIMIT - reg_SP));
                $finish(1);
            end

            y = memmory[reg_SP++];
            x = memmory[reg_SP++];

            if (y == 0) begin
        $write("RuntimeError: line~%d: Division by zero.\n",
                                           (reg_IP - 256));
                $finish(1);
            end

            memmory[--reg_SP] = x / y;
            ++reg_IP;
        end

        CMD_MOD : begin
            if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
                $write(STACK_ERROR, (reg_IP - 256),
                     "MOD", 2, (`MEMMORY_LIMIT - reg_SP));
                $finish(1);
            end

            y = memmory[reg_SP++];
            x = memmory[reg_SP++];

            if (y == 0) begin
        $write("RuntimeError: line~%d: Division by zero.\n",
                                           (reg_IP - 256));
                $finish(1);
            end

            memmory[--reg_SP] = x % y;
            ++reg_IP;
        end
```

```
CMD_NEG : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
        $write(STACK_ERROR, (reg_IP - 256),
              "NEG", 1, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    x = memmory[reg_SP++];
    memmory[--reg_SP] = -x;
    ++reg_IP;
end

CMD_BITAND : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
        $write(STACK_ERROR, (reg_IP - 256),
           "BITAND", 2, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    y = memmory[reg_SP++];
    x = memmory[reg_SP++];
    memmory[--reg_SP] = x & y;
    ++reg_IP;
end

CMD_BITOR : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
        $write(STACK_ERROR, (reg_IP - 256),
            "BITOR", 2, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    y = memmory[reg_SP++];
    x = memmory[reg_SP++];
    memmory[--reg_SP] = x | y;
    ++reg_IP;
end

CMD_BITNOT : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
        $write(STACK_ERROR, (reg_IP - 256),
           "BITNOT", 1, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    x = memmory[reg_SP++];
    memmory[--reg_SP] = ~x;
    ++reg_IP;
end

CMD_LSHIFT : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
        $write(STACK_ERROR, (reg_IP - 256),
           "LSHIFT", 2, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    y = memmory[reg_SP++];
    x = memmory[reg_SP++];
    memmory[--reg_SP] = x << y;
    ++reg_IP;
end

CMD_RSHIFT : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
        $write(STACK_ERROR, (reg_IP - 256),
           "RSHIFT", 2, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    y = memmory[reg_SP++];
    x = memmory[reg_SP++];
    memmory[--reg_SP] = x >> y;
    ++reg_IP;
end

CMD_DUP : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
        $write(STACK_ERROR, (reg_IP - 256),
              "DUP", 1, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    x = memmory[reg_SP++];
    memmory[--reg_SP] = x;
    memmory[--reg_SP] = x;
    ++reg_IP;
end

CMD_DROP : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
        $write(STACK_ERROR, (reg_IP - 256),
            "DROP", 1, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    x = memmory[reg_SP++];
    ++reg_IP;
end

CMD_SWAP : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
        $write(STACK_ERROR, (reg_IP - 256),
            "SWAP", 2, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    y = memmory[reg_SP++];
    x = memmory[reg_SP++];
    memmory[--reg_SP] = y;
    memmory[--reg_SP] = x;
    ++reg_IP;
end

CMD_ROT : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 3) begin
        $write(STACK_ERROR, (reg_IP - 256),
             "ROT", 3, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    z = memmory[reg_SP++];
    y = memmory[reg_SP++];
    x = memmory[reg_SP++];
    memmory[--reg_SP] = y;
    memmory[--reg_SP] = z;
    memmory[--reg_SP] = x;
    ++reg_IP;
end

CMD_OVER : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
        $write(STACK_ERROR, (reg_IP - 256),
            "OVER", 2, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    y = memmory[reg_SP++];
```

```
       x = memmory[reg_SP++];
       memmory[--reg_SP] = x;
       memmory[--reg_SP] = y;
       memmory[--reg_SP] = x;
       ++reg_IP;
   end

   CMD_DROPN : begin
       if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
           $write(STACK_ERROR, (reg_IP - 256),
               "DROPN", 1, (`MEMMORY_LIMIT - reg_SP));
           $finish(1);
       end
if ((`MEMMORY_LIMIT - reg_SP) < memmory[reg_SP] + 1) begin
           $write(STACK_ERROR, (reg_IP - 256),
                   "DROPN", (memmory[reg_SP] + 1),
                   (`MEMMORY_LIMIT - reg_SP));
           $finish(1);
       end

       reg_SP += memmory[reg_SP] + 1;
       ++reg_IP;
   end

   CMD_PUSHN : begin
       if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
           $write(STACK_ERROR, (reg_IP - 256),
               "PUSHN", 1, (`MEMMORY_LIMIT - reg_SP));
           $finish(1);
       end

       reg_SP -= memmory[reg_SP] - 1;
       ++reg_IP;
   end

   CMD_LOAD : begin
       if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
           $write(STACK_ERROR, (reg_IP - 256),
                "LOAD", 1, (`MEMMORY_LIMIT - reg_SP));
           $finish(1);
       end

       a = memmory[reg_SP++];

       if (!(256 <= a && a <= `MEMMORY_LIMIT)) begin
           $write(
"RuntimeError: line~%d: LOAD: memmory adress must be between", (reg_IP -
256));
                        $write(" 256 and %d.", `MEMMORY_LIMIT);
                        $write(" Your request adress - %d.\n", a);
                        $finish(1);
                    end

                    memmory[--reg_SP] = memmory[a];
                    ++reg_IP;
                end

                CMD_SAVE : begin
                    if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
                        $write(STACK_ERROR, (reg_IP - 256),
                             "SAVE", 2, (`MEMMORY_LIMIT - reg_SP));
                        $finish(1);
                    end

                    v = memmory[reg_SP++];
                    a = memmory[reg_SP++];

                    if (!(256 <= a && a <= `MEMMORY_LIMIT)) begin
                        $write("RuntimeError: line~%d: SAVE:\
memmory adress must be between", (reg_IP - 256));
                       $write(" %d and %d.", 256, `MEMMORY_LIMIT);
                        $write(" Your request adress - %d.\n", a);
                        $finish(1);
                    end

                    memmory[a] = v;
                    ++reg_IP;
                end

                CMD_GETIP : begin
                    memmory[--reg_SP] = reg_IP++;
                end

                CMD_SETIP : begin
                    if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
                        $write(STACK_ERROR, (reg_IP - 256),
                            "SETIP", 1, (`MEMMORY_LIMIT - reg_SP));
                        $finish(1);
                    end

                    reg_IP = memmory[reg_SP++];
                end
```

```
CMD_GETSP : begin
    a = reg_SP;
    memmory[--reg_SP] = a;
    ++reg_IP;
end

CMD_SETSP : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
        $write(STACK_ERROR, (reg_IP - 256),
            "SETSP", 1, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    reg_SP = memmory[reg_SP++];

    if (!(256 + program_size <= reg_SP &&
                   reg_SP <= `MEMMORY_LIMIT)) begin
$write("RuntimeError: line~%d: SETSP: value of register");
        $write(" SP must be between %d and %d.",
               256 + program_size, `MEMMORY_LIMIT);
$write("But register SP was set with %s.\n", reg_SP);
        $finish(1);
    end

    ++reg_IP;
end

CMD_GETFP : begin
    memmory[--reg_SP] = reg_FP;
    ++reg_IP;
end

CMD_SETFP : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
        $write(STACK_ERROR, (reg_IP - 256),
            "SETFP", 1, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    reg_FP = memmory[reg_SP++];
    ++reg_IP;
end

CMD_GETRV : begin
    memmory[--reg_SP] = reg_RV;
    ++reg_IP;
end

CMD_SETRV : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
        $write(STACK_ERROR, (reg_IP - 256),
            "SETRV", 1, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    reg_RV = memmory[reg_SP++];
    ++reg_IP;
end

CMD_CMP : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
        $write(STACK_ERROR, (reg_IP - 256),
              "CMP", 2, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    y = memmory[reg_SP++];
    x = memmory[reg_SP++];

    if (x < y) begin
        memmory[--reg_SP] = -1;
    end else if (x == y) begin
        memmory[--reg_SP] = 0;
    end else begin
        memmory[--reg_SP] = 1;
    end

    ++reg_IP;
end

CMD_JMP : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
        $write(STACK_ERROR, (reg_IP - 256),
              "JMP", 1, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    reg_IP = memmory[reg_SP++];
end

CMD_JLT : begin
```

```
    if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
        $write(STACK_ERROR, (reg_IP - 256),
              "JLT", 2, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    a = memmory[reg_SP++];
    x = memmory[reg_SP++];
    reg_IP = ((x < 0) ? a : (reg_IP + 1));
end

CMD_JGT : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
        $write(STACK_ERROR, (reg_IP - 256),
              "JGT", 2, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    a = memmory[reg_SP++];
    x = memmory[reg_SP++];
    reg_IP = ((x > 0) ? a : (reg_IP + 1));
end

CMD_JEQ : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
        $write(STACK_ERROR, (reg_IP - 256),
              "JEQ", 2, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    a = memmory[reg_SP++];
    x = memmory[reg_SP++];
    reg_IP = ((x == 0) ? a : (reg_IP + 1));
end

CMD_JLE : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
        $write(STACK_ERROR, (reg_IP - 256),
              "JLE", 2, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    a = memmory[reg_SP++];
    x = memmory[reg_SP++];
    reg_IP = ((x <= 0) ? a : (reg_IP + 1));
end
```

```
    CMD_JGE : begin
        if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
            $write(STACK_ERROR, (reg_IP - 256),
                  "JGE", 2, (`MEMMORY_LIMIT - reg_SP));
            $finish(1);
        end

        a = memmory[reg_SP++];
        x = memmory[reg_SP++];
        reg_IP = ((x >= 0) ? a : (reg_IP + 1));
    end

    CMD_JNE : begin
        if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
            $write(STACK_ERROR, (reg_IP - 256),
                  "JNE", 2, (`MEMMORY_LIMIT - reg_SP));
            $finish(1);
        end

        a = memmory[reg_SP++];
        x = memmory[reg_SP++];
        reg_IP = ((x != 0) ? a : (reg_IP + 1));
    end

    CMD_CALL : begin
        if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
            $write(STACK_ERROR, (reg_IP - 256),
                 "CALL", 1, (`MEMMORY_LIMIT - reg_SP));
            $finish(1);
        end

        a = memmory[reg_SP++];
        memmory[--reg_SP] = reg_IP + 1;
        reg_IP = a;
    end

    CMD_RETN : begin
        if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
            $write(STACK_ERROR, (reg_IP - 256),
                 "RETN", 2, (`MEMMORY_LIMIT - reg_SP));
            $finish(1);
        end
  if ((`MEMMORY_LIMIT - reg_SP) < memmory[reg_SP] + 2) begin
            $write(STACK_ERROR, (reg_IP - 256),
                    "RETN", memmory[reg_SP] + 2,
                    (`MEMMORY_LIMIT - reg_SP));
            $finish(1);
        end

        N = memmory[reg_SP++];
        a = memmory[reg_SP++];
        reg_SP += N;
        reg_IP = a;
    end

    CMD_IN : begin
        char = $fgetc(fd);
        memmory[--reg_SP] = char;
        ++reg_IP;
    end

    CMD_OUT : begin
        if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
            $write(STACK_ERROR, (reg_IP - 256),
                  "OUT", 1, (`MEMMORY_LIMIT - reg_SP));
            $finish(1);
        end

        char = memmory[reg_SP++];
        $write("%s", char);
        ++reg_IP;
    end

    CMD_HALT : begin
        if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
            $write(STACK_ERROR, (reg_IP - 256),
                 "HALT", 1, (`MEMMORY_LIMIT - reg_SP));
            $finish(1);
        end

        a = memmory[reg_SP++];
   $write("\n\nProgram terminated with code %d.\n", a);
        ++reg_IP;
        $finish;
    end

    default : begin
                                // неотрицательное число
  if (memmory[reg_IP] >= 0) begin  // помещаем на вершину стека
            memmory[--reg_SP] = memmory[reg_IP++];
              // ошибка, если слово не распознано ни как
        end else begin // интсрукция, ни как неотрицательное число
              $write("RuntimeError: line~%d: Unknown command -
'%d'.\n",
                             (reg_SP - 256), memmory[reg_IP]);
                    $finish(1);
                 end
            end
        endcase

    end
end
endmodule
```

main.ref

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

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

run.sh

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

source.txt

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

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

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

(function newLine ()
    (call out 10)
)

(class Animal ()
    (fields (Animal_name 1))

    (method setName (self name)
        (((L self) "+" Animal_name) "=" (L name))
    )

    (method getName (self)
        (return (L ((L self) "+" Animal_name)))
    )

    (method getType (self)
        (return 1)
    )
)

(class Cat (Animal)
    (fields (Cat_breed 1))


    (method getType (self)
        (return 2)
    )

    (method setBreed (self breed)
        (((L self) "+" Cat_breed) "=" (L breed))
    )

    (method getBreed (self)
        (return (L ((L self) "+" Cat_breed)))
    )
)


(class B ()
  (method ff (self) (return self))
  (method gg (self) (return self))
  (method hh (self) (return self))
  (method kk (self) (return self))
)

(class D (B)
  (method gg (self) (return self))
  (method kk (self) (return self))
)

(class E (D)
  (method gg (self) (return self))
  (method ff (self) (return self))
)

(function main ()
    (block ((animal Animal) (cat Cat))
        (init animal Animal)
        (init cat Cat)

        (mcall animal setName 1501)
        (call printInt (mcall animal getName))
        (call newLine)
        (call printInt (mcall animal getType))
        (call newLine)

        (mcall cat setName 100500)
        (mcall cat setBreed 1007)
        (call printInt (mcall cat getName))
        (call newLine)
        (call printInt (mcall cat getBreed))
        (call newLine)
        (call printInt (mcall cat getType))
        (call newLine)

        (if (not (isinstance animal Animal))
            (call printInt 6661)
            (call newLine)
        )
        (if (not (isinstance cat Animal))
            (call printInt 6662)
            (call newLine)
        )
        (if (isinstance animal Cat)
            (call printInt 6663)
            (call newLine)
        )
        (if (not (isinstance cat Cat))
            (call printInt 6664)
            (call newLine)
        )
    )
    (return 0)
)
```

stdin.txt

cmd sh run.sh

```
*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:
*Compiling src/OOP/OOP_Program.ref:
*Compiling src/OOP/OOP_Definition.ref:
*Compiling src/OOP/OOP_Function.ref:
*Compiling src/OOP/OOP_Code.ref:
*Compiling src/OOP/OOP_Statement.ref:
*Compiling src/OOP/OOP_Expr.ref:
*Compiling src/OOP/OOP_BoolExpr.ref:
*Compiling src/OOP/OOP_Class.ref:
** Compilation succeeded **
rm: cannot remove '*.rasl': No such file or directory
1501
1
```

```
100500
1007
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 средств объектно-ориентированного программирования.