Лабораторная работа № 5. «Управление памятью и сборка мусора»

5 июня 2024 г.

Сергей Виленский, ИУ9-62Б

Цель работы

Целью данной работы является изучение способов организации кучи и алгоритмов сборки мусора.

Индивидуальный вариант

Управление памятью подсчётом ссылок + метод близнецов.

Реализация и тестирование

src/DYN/DYN_BoolExpr.ref

```
*$FROM src/DYN/DYN_Expr.ref
$EXTERN DYN_Expr;

/**
  <DYN_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 DYN_BoolExpr {
  (not t.BoolExpr) = (not <DYN_BoolExpr t.BoolExpr>);
 (t.BoolExprL and t.BoolExprR) = (<DYN_BoolExpr t.BoolExprL> and <DYN_BoolExpr t.BoolExprR>);
 (t.BoolExprL or t.BoolExprR) = (<DYN_BoolExpr t.BoolExprL> or <DYN_BoolExpr t.BoolExprR>);
 (isinstance t.ObjectPtr s.Name) = (isinstance <DYN_Expr t.ObjectPtr> s.Name);
```

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

src/DYN/DYN_Class.ref

```
*$FROM src/DYN/DYN_LocalVars.ref
*$FROM src/DYN/DYN_Code.ref
*$FROM LibraryEx
$EXTERN DYN_LocalVars, DYN_Code, Map;

/**
  <DYN_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 DYN_Class {
    (class s.Name t.Base (fields e.Fields) e.Methods)
        = (class s.Name t.Base
            (fields e.Fields)
            <Map {
                (method s.Name_ t.Params (var e.LocalVars) e.Code)
              = (method s.Name_ t.Params <DYN_LocalVars (var e.LocalVars)> <DYN_Code () e.Code>)
           (method s.Name_ t.Params e.Code) = (method s.Name_ t.Params <DYN_Code () e.Code>);
            } e.Methods>
          );
   (class s.Name t.Base e.Methods) = <DYN_Class (class s.Name t.Base (fields) e.Methods)>;
}
```

src/DYN/DYN_Code.ref

```
*$FROM src/DYN/DYN_Statement.ref
$EXTERN DYN_Statement;

/**
  <DYN_Code t.LocalRefs t.Statement*> == e.Code
*/
$ENTRY DYN_Code {
  t.LocalRefs t.Statement e.Statements
   = <DYN_Statement t.LocalRefs t.Statement> <DYN_Code t.LocalRefs e.Statements>;
```

```
  t.LocalRefs = /* пусто */;
}
```

src/DYN/DYN_Definition.ref

```
*$FROM src/DYN/DYN_Function.ref
*$FROM src/DYN/DYN_GlobalRefs.ref
*$FROM src/DYN/DYN_DynVarType.ref
*$FROM src/DYN/DYN_Class.ref
*$FROM LibraryEx
$EXTERN DYN_Function, DYN_GlobalRefs, DYN_DynVarType, DYN_Class, LoadExpr;

/**
  <DYN_Definition
      t.Struct
    | t.Const
    | t.GlobalVar
    | t.Function
    | t.GlobalRefs
    | t.DynVarType
    | t.Class> == t.Definition
*/
$ENTRY DYN_Definition {
  (function main t.Params (var e.Vars) e.Body)
    = <DYN_Function
        (function main t.Params (var e.Vars)
          (call prepareHeap__)
          e.Body)
      >;
  (function main t.Params e.Body)
    = <DYN_Definition (function main t.Params (var) e.Body)>;

  (refs e.Names) = <DYN_GlobalRefs (refs e.Names)>;
  (dynvar e.Dynvar) = <DYN_DynVarType (dynvar e.Dynvar)>;

  (function e.Function) = <DYN_Function (function e.Function)>;
  (class e.Class) = <DYN_Class (class e.Class)>;
  t.Other = t.Other;
}
```

src/DYN/DYN_DynVarType.ref

```
*$FROM LibraryEx
$EXTERN Map;

/**
```

```
  <DYN_DynVarType
    (dynvar s.Name
      (refs s.Name*)?
      (s.Name t.ConstExpr)*
    )
> == t.Struct | t.Class
*/
$ENTRY DYN_DynVarType {
    (dynvar s.Name (refs e.Names) e.Defs)
    = (const <Implode_Ext <Explode s.Name> '_refcount__'> "=" <Len e.Names>)
        (struct s.Name
          <Map {s.Name_ = (s.Name_ 1)} e.Names>
          e.Defs
        );
   (dynvar s.Name e.Defs) = <DYN_DynVarType (dynvar s.Name (refs) e.Defs)>;
}

Len { /* Lenw */
  t._ e.Names = <Add <Len e.Names> 1>;
  /* пусто */ = 0;

*  e.Expr, <Lenw e.Expr> : s.Len e._ = s.Len;
}
```

src/DYN/DYN_Expr.ref

```
*$FROM src/DYN/DYN_Code.ref
*$FROM src/DYN/DYN_LocalVars.ref
*$FROM LibraryEx
$EXTERN DYN_Code, DYN_LocalVars, Map;

/**
  <DYN_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 DYN_Expr {
```

```
  (L t.Expr) = (L <DYN_Expr t.Expr>);
  ("-" t.Expr) = ("-" <DYN_Expr t.Expr>);
  (call t.Expr e.Exprs) = (call <Map DYN_Expr t.Expr e.Exprs>);
 (mcall t.Object s.Method e.Args) = (mcall <DYN_Expr t.Object> s.Method <Map DYN_Expr e.Args>);
 (t.ExprL "=" t.ExprR) = (<DYN_Expr t.ExprL> "=" <DYN_Expr t.ExprR>);
 (let t.LocalVars e.Code t.Expr) = (let <DYN_LocalVars t.LocalVars> <DYN_Code e.Code> <DYN_Expr
 (t.ExprL s.BinOp t.ExprR) = (<DYN_Expr t.ExprL> s.BinOp <DYN_Expr t.ExprR>);
  e.Other = e.Other;
}
```

src/DYN/DYN_Function.ref

```
*$FROM src/DYN/DYN_LocalVars.ref
*$FROM src/DYN/DYN_Code.ref
*$FROM LibraryEx
$EXTERN DYN_LocalVars, DYN_Code, Map;

/**
 <DYN_Function (function s.Name (s.Name*) t.LocalVars? e.Code)> == t.Function
*/
$ENTRY DYN_Function {
  (function s.Name t.Params (var e.LocalVars) e.Code)
   , <DYN_LocalVars (var e.LocalVars)> : (e.LocalRefs) t.LocalVars_
   = (function s.Name t.Params t.LocalVars_
       <Map {
         s.LocalRef = (s.LocalRef "=" 0);
       } e.LocalRefs>
       <DYN_Code (e.LocalRefs) e.Code (return 0)>
     );

  (function s.Name t.Params e.Code)
    = (function s.Name t.Params <DYN_Code () e.Code (return 0)>);
}
```

src/DYN/DYN_GlobalRefs.ref

```
/**
  <DYN_GlobalRefs (refs s.Name*)> == t.GlobalVar*
*/
$ENTRY DYN_GlobalRefs {
    (refs s.Name e.Names)
      = (var s.Name 1 "=" 0)
        <DYN_GlobalRefs (refs e.Names)>;
    (refs) = /* пусто */;
}
```

src/DYN/DYN_LocalVars.ref

```
*$FROM LibraryEx
$EXTERN Map;

/**
 <DYN_LocalVars (var (refs s.Name*)? (s.Name t.ConstExpr)*)> == t.LocalRefs t.LocalVars
*/
$ENTRY DYN_LocalVars {
   (var (refs e.Names) e.Defs) = (e.Names) (var <Map {s.Name = (s.Name 1)} e.Names> e.Defs);
    (var e.Defs) = () (var e.Defs);

*   ((refs e.Names) e.Defs) = (<Map {s.Name = (s.Name 1)} e.Names>);
    (e.Defs) = (e.Defs);
}
```

src/DYN/DYN_Program.ref

```
*$FROM src/DYN/DYN_Definition.ref
*$FROM LibraryEx
$EXTERN DYN_Definition, LoadExpr;

/**
  <DYN_Program t.Definition*> == e.Program
*/
$ENTRY DYN_Program {
   e.Definitions = <_DYN_Program <LoadExpr 'src/DYN/DYN_RunTimeLib.txt'> e.Definitions>;
}

_DYN_Program {
  t.Definition e.Definitions
    = <DYN_Definition t.Definition> <_DYN_Program e.Definitions>;
  /* пусто */ = /* пусто */;
}
```

src/DYN/DYN_RunTimeLib.txt

```
(function inc__ (ptr)
    ((L ptr) "=" ((L (L ptr)) "+" 1))
)

(function dec__ (ptr)
    ((L ptr) "=" ((L (L ptr)) "-" 1))
)

(const min_block_order__ "=" 3)
```

```
(var heap_order__ 1 "=" min_block_order__)
(var heap_start__ 1)

(const block_prev__ "=" 0)
(const block_next__ "=" 1)
(const block_size__ "=" 2)
(const block_buzy__ "=" 3)

(const ref_links_count__ "=" 1)
(const ref_refs_count__ "=" 2)

(function getHeapBlock__ (size)
   (return ((L PROGRAM_SIZE) "+" ((L size) "-" min_block_order__)))
)
(function getNextBlock__ (block)
    (return (L ((L block) "+" block_next__)))
)

(function unlinkBlock__ (block)
    (var (prevBlock 1) (nextBlock 1))

    (prevBlock "=" (L ((L block) "+" block_prev__)))
    (nextBlock "=" (L ((L block) "+" block_next__)))

    (if ((L prevBlock) "<>" 0)
        (((L prevBlock) "+" block_next__) "=" (L nextBlock))
    else
     ((call getHeapBlock__ (L ((L block) "+" block_size__))) "=" (L nextBlock))
    )
    (if ((L nextBlock) "<>" 0)
        (((L nextBlock) "+" block_prev__) "=" (L prevBlock))
    )

    (((L block) "+" block_buzy__) "=" 1)
)

(function linkBlocks__ (prevBlock nextBlock)
    (if ((L prevBlock) "<>" 0)
        (((L prevBlock) "+" block_next__) "=" (L nextBlock))
    )
    (if ((L nextBlock) "<>" 0)
        (((L nextBlock) "+" block_prev__) "=" (L prevBlock))
    )
)

(function insertBlock__ (block)
```

```
    (var (listPtr 1))

   (listPtr "=" (call getHeapBlock__ (L ((L block) "+" block_size__))))

    (call linkBlocks__ (L block) (L (L listPtr)))
    (((L block) "+" block_prev__) "=" 0)
    (((L block) "+" block_buzy__) "=" 0)
    ((L listPtr) "=" (L block))
)

(function splitBlock__ (block)
    (var (blockOrder 1) (newBlock 1) (listNew 1))

    (call unlinkBlock__ (L block))

    (blockOrder "=" ((L ((L block) "+" block_size__)) "-" 1))

    (newBlock "=" ((L block) "+" (1 "<<" (L blockOrder))))
    (((L block) "+" block_size__) "=" (L blockOrder))
    (((L newBlock) "+" block_size__) "=" (L blockOrder))

    (call insertBlock__ (L newBlock))
    (call insertBlock__ (L block))
)

(function unionBlock__ (block)
    (var (buddy 1))

    (if ((L ((L block) "+" block_buzy__)) "==" 1)
        (return 0)
    )

   (if ((((L block) "-" (L heap_start__)) "&" (1 "<<" (L ((L block) "+" block_size__)))) "==" 0)
      (buddy "=" ((L block) "+" (1 "<<" (L ((L block) "+" block_size__)))))
    else
        (buddy "=" (L block))
     (block "=" ((L block) "-" (1 "<<" (L ((L block) "+" block_size__)))))
    )

    (if (((L ((L buddy) "+" block_buzy__)) "==" 1) or
     ((L ((L block) "+" block_size__)) "<>" (L ((L buddy) "+" block_size__)))
    )
        (return 0)
    )

    (call unlinkBlock__ (L block))
```

```
    (call unlinkBlock__ (L buddy))

    (call inc__ ((L block) "+" block_size__))
    (call insertBlock__ (L block))

    (call unionBlock__ (L block))
)

(function allocBlock__ (size)
    (var (r 1) (n 1) (block 1))

    (r "=" min_block_order__)
   (while ((L size) ">" ((1 "<<" (L r)) "-" (1 "<<" min_block_order__)))
        (call inc__ r)
    )

    (n "=" (L r))
    (while ((L (call getHeapBlock__ (L n))) "==" 0)
        (call inc__ n)
        (if ((L n) ">" (L heap_order__))
            (return 0)
        )
    )

    (while ((L n) "<>" (L r))
        (call splitBlock__ (L (call getHeapBlock__ (L n))))
        (call dec__ n)
    )

    (block "=" (L (call getHeapBlock__ (L r))))

    (call unlinkBlock__ (L block))
    (block "=" ((L block) "+" (1 "<<" min_block_order__)))
    (((L block) "-" ref_links_count__) "=" 1)
    (((L block) "-" ref_refs_count__) "=" 0)
    (return (L block))
)

(function deallocBlock__ (block)
    (block "=" ((L block) "-" (1 "<<" min_block_order__)))
    (call insertBlock__ (L block))
    (call unionBlock__ (L block))
)

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

```
)

(function printInt__ (int)
    (if ((L int) ">=" 10)
        (call printInt__ ((L int) "/" 10))
    )
    (call out__ (((L int) "%" 10) "+" 48))
)

(function newLine__ ()
    (call out__ 10)
)

(function space__ ()
    (call out__ 32)
)

(function printHeap__ ()
    (block ((i 1))
        (i "=" min_block_order__)
        (while ((L i) "<=" (L heap_order__))
            (block ((block 1) (count 1))
                (count "=" 0)
                (block "=" (L (call getHeapBlock__ (L i))))
                (while ((L block) "<>" 0)
                    (call inc__ count)
                    (block "=" (call getNextBlock__ (L block)))
                )
                (call printInt__ (L count))
                (call space__)
            )
            (call inc__ i)
        )
        (call newLine__)
    )
)

(function refInc__ (ref)
    (if ((L ref) "<>" 0)
        (call inc__ ((L ref) "-" ref_links_count__))
    )
)

(function refDec__ (ref)
    (if ((L ref) "<>" 0)
        (call dec__ ((L ref) "-" ref_links_count__))
```

```
        (if ((L ((L ref) "-" ref_links_count__)) "==" 0)
            (block ((i 1) (refcount 1))
                (refcount "=" (L ((L ref) "-" ref_refs_count__)))
                (i "=" 0)
                (while ((L i) "<>" (L refcount))
                    (call refDec__ (L ((L ref) "+" (L i))))
                    (call inc__ i)
                )
            )
            (call deallocBlock__ (L ref))
        )
    )
)

(function prepareHeap__ ()
    (block ((heap_size 1) (memory_size 1))
        (heap_size "=" (1 "<<" min_block_order__))
     (memory_size "=" ((L MEMORY_SIZE) "-" ((L PROGRAM_SIZE) "+" 64)))
        ((call getHeapBlock__ (L heap_order__)) "=" 0)
        (while ((L heap_size) "<=" (L memory_size))
            ((call getHeapBlock__ (L heap_order__)) "=" 0)
            (call inc__ heap_order__)
            (heap_size "=" ((L heap_size) "<<" 1))
        )
        (call dec__ heap_order__)
    )
   (heap_start__ "=" ((call getHeapBlock__ (L heap_order__)) "+" 1))
    (((L heap_start__) "+" 2) "=" (L heap_order__))
    (call insertBlock__ (L heap_start__))
)
```

src/DYN/DYN_Statement.ref

```
*$FROM src/DYN/DYN_Code.ref
*$FROM src/DYN/DYN_Expr.ref
*$FROM src/DYN/DYN_LocalVars.ref
*$FROM src/DYN/DYN_BoolExpr.ref
*$FROM LibraryEx
$EXTERN DYN_Code, DYN_Expr, DYN_LocalVars, DYN_BoolExpr, Map;

/**
  <DYN_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.Expr ":-" t.Expr)
    | (gc-alloc t.Expr s.Name)
    | (ref-return t.Expr)
  > == t.Statement*
*/
$ENTRY DYN_Statement {
 t.LocalRefs (t.ObjectPtr "=" t.Expr) = (<DYN_Expr t.ObjectPtr> "=" <DYN_Expr t.Expr>);
 t.LocalRefs (call t.Expr e.Args) = (call <Map DYN_Expr t.Expr e.Args>);
 t.LocalRefs (mcall t.Object s.Method e.Args) = (mcall <DYN_Expr t.Object> s.Method <Map DYN_Exp
  t.LocalRefs (if t.BoolExpr e.CodeT else e.CodeF)
   = (if <DYN_BoolExpr t.BoolExpr> <DYN_Code t.LocalRefs e.CodeT> else <DYN_Code t.LocalRefs e.C
 t.LocalRefs (if t.BoolExpr e.Code) = (if <DYN_BoolExpr t.BoolExpr> <DYN_Code t.LocalRefs e.Cod
 t.LocalRefs (while t.BoolExpr e.Code) = (while <DYN_BoolExpr t.BoolExpr> <DYN_Code t.LocalRefs
  t.LocalRefs (block t.LocalVars e.Code)
   = (block <DYN_LocalVars t.LocalVars> <DYN_Code t.LocalRefs e.Code>);
 t.LocalRefs (init t.ObjectPtr s.Name) = (init <DYN_Expr t.ObjectPtr> s.Name);

  (e.LocalRefs) (return t.Expr)
    = (return
        (let ((let_ret__ 1))
          (let_ret__ "=" <DYN_Expr t.Expr>)
          <Map {
            s.LocalRef = (call refDec__ (L s.LocalRef));
          } e.LocalRefs>
          (L let_ret__)
        )
      );

  t.LocalRefs (t.Var ":-" (call e.Args))
  = (block ((blockVar__ 1) (blockAddr__ 1))
      (blockAddr__ "=" <DYN_Expr (call e.Args)>)
      (blockVar__ "=" <DYN_Expr t.Var>)
      (call refDec__ (L (L blockVar__)))
      ((L blockVar__) "=" (L blockAddr__))
    );

  t.LocalRefs (t.Var ":-" t.Addr)
    = (block ((blockVar__ 1) (blockAddr__ 1))
        (blockAddr__ "=" <DYN_Expr t.Addr>)
        (blockVar__ "=" <DYN_Expr t.Var>)
```

```
        (call refInc__ (L blockAddr__))
        (call refDec__ (L (L blockVar__)))
        ((L blockVar__) "=" (L blockAddr__))
      );

  t.LocalRefs (gc-alloc t.Expr s.Name)
    = (block ((ref 1))
        (ref "=" <DYN_Expr t.Expr>)
        (call refDec__ (L (L ref)))
        ((L ref) "=" (call allocBlock__ s.Name))
     (((L (L ref)) "-" ref_refs_count__) "=" <Implode_Ext <Explode s.Name> '_refcount__'>)
      );

  (e.LocalRefs) (ref-return t.Expr)
    = (return
        (let ((let_ret__ 1))
          (let_ret__ "=" <DYN_Expr t.Expr>)
          (call refInc__ (L let_ret__))
          <Map {
            s.LocalRef = (call refDec__ (L s.LocalRef));
          } e.LocalRefs>
          (L let_ret__)
        )
      );

  t.LocalRefs e.Other = e.Other
}
```

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) e.Methods)>;
}
```

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 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.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

:_inc__
GETFP GETSP SETFP
GETFP 2 ADD LOAD GETFP 2 ADD LOAD LOAD 1 ADD SAVE
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB 0 SAVE
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __inc__ JMP
:__inc__
GETFP SETSP SETFP
1 RETN

:_dec__
GETFP GETSP SETFP
GETFP 2 ADD LOAD GETFP 2 ADD LOAD LOAD 1 SUB SAVE
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB 0 SAVE
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __dec__ JMP
:__dec__
GETFP SETSP SETFP
1 RETN

:_heap_order__
3

:_heap_start__
0

:_getHeapBlock__
GETFP GETSP SETFP
GETFP GETSP SETFP
```

```
1 PUSHN
GETFP 1 SUB _PROGRAM_SIZE LOAD GETFP LOAD 2 ADD LOAD 3 SUB ADD SAVE
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __getHeapBlock__ JMP
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB 0 SAVE
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __getHeapBlock__ JMP
:__getHeapBlock__
GETFP SETSP SETFP
1 RETN

:_getNextBlock__
GETFP GETSP SETFP
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB GETFP LOAD 2 ADD LOAD 1 ADD LOAD SAVE
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __getNextBlock__ JMP
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB 0 SAVE
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __getNextBlock__ JMP
:__getNextBlock__
GETFP SETSP SETFP
1 RETN

:_unlinkBlock__
GETFP GETSP SETFP
2 PUSHN
GETFP 1 SUB GETFP 2 ADD LOAD 0 ADD LOAD SAVE
GETFP 2 SUB GETFP 2 ADD LOAD 1 ADD LOAD SAVE
GETFP 1 SUB LOAD
0
CMP _true_if_0_unlinkBlock__ JNE
_false_if_0_unlinkBlock__ JMP
:_true_if_0_unlinkBlock__
GETFP 1 SUB LOAD 1 ADD GETFP 2 SUB LOAD SAVE
_exit_if_0_unlinkBlock__ JMP
:_false_if_0_unlinkBlock__
GETFP 2 ADD LOAD 2 ADD LOAD _getHeapBlock__ CALL GETRV GETFP 2 SUB LOAD SAVE
:_exit_if_0_unlinkBlock__
GETFP 2 SUB LOAD
0
```

```
CMP _true_if_1_unlinkBlock__ JNE
_exit_if_1_unlinkBlock__ JMP
:_true_if_1_unlinkBlock__
GETFP 2 SUB LOAD 0 ADD GETFP 1 SUB LOAD SAVE
:_exit_if_1_unlinkBlock__
GETFP 2 ADD LOAD 3 ADD 1 SAVE
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB 0 SAVE
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __unlinkBlock__ JMP
:__unlinkBlock__
GETFP SETSP SETFP
1 RETN

:_linkBlocks__
GETFP GETSP SETFP
GETFP 2 ADD LOAD
0
CMP _true_if_0_linkBlocks__ JNE
_exit_if_0_linkBlocks__ JMP
:_true_if_0_linkBlocks__
GETFP 2 ADD LOAD 1 ADD GETFP 3 ADD LOAD SAVE
:_exit_if_0_linkBlocks__
GETFP 3 ADD LOAD
0
CMP _true_if_1_linkBlocks__ JNE
_exit_if_1_linkBlocks__ JMP
:_true_if_1_linkBlocks__
GETFP 3 ADD LOAD 0 ADD GETFP 2 ADD LOAD SAVE
:_exit_if_1_linkBlocks__
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB 0 SAVE
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __linkBlocks__ JMP
:__linkBlocks__
GETFP SETSP SETFP
2 RETN

:_insertBlock__
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB GETFP 2 ADD LOAD 2 ADD LOAD _getHeapBlock__ CALL GETRV SAVE
GETFP 1 SUB LOAD LOAD GETFP 2 ADD LOAD _linkBlocks__ CALL GETRV DROP
GETFP 2 ADD LOAD 0 ADD 0 SAVE
```

```
GETFP 2 ADD LOAD 3 ADD 0 SAVE
GETFP 1 SUB LOAD GETFP 2 ADD LOAD SAVE
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB 0 SAVE
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __insertBlock__ JMP
:__insertBlock__
GETFP SETSP SETFP
1 RETN

:_splitBlock__
GETFP GETSP SETFP
3 PUSHN
GETFP 2 ADD LOAD _unlinkBlock__ CALL GETRV DROP
GETFP 1 SUB GETFP 2 ADD LOAD 2 ADD LOAD 1 SUB SAVE
GETFP 2 SUB GETFP 2 ADD LOAD 1 GETFP 1 SUB LOAD LSHIFT ADD SAVE
GETFP 2 ADD LOAD 2 ADD GETFP 1 SUB LOAD SAVE
GETFP 2 SUB LOAD 2 ADD GETFP 1 SUB LOAD SAVE
GETFP 2 SUB LOAD _insertBlock__ CALL GETRV DROP
GETFP 2 ADD LOAD _insertBlock__ CALL GETRV DROP
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB 0 SAVE
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __splitBlock__ JMP
:__splitBlock__
GETFP SETSP SETFP
1 RETN

:_unionBlock__
GETFP GETSP SETFP
1 PUSHN
GETFP 2 ADD LOAD 3 ADD LOAD
1
CMP _true_if_0_unionBlock__ JEQ
_exit_if_0_unionBlock__ JMP
:_true_if_0_unionBlock__
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB 0 SAVE
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __unionBlock__ JMP
:_exit_if_0_unionBlock__
GETFP 2 ADD LOAD _heap_start__ LOAD SUB 1 GETFP 2 ADD LOAD 2 ADD LOAD LSHIFT BITAND
0
```

```
CMP _true_if_1_unionBlock__ JEQ
_false_if_1_unionBlock__ JMP
:_true_if_1_unionBlock__
GETFP 1 SUB GETFP 2 ADD LOAD 1 GETFP 2 ADD LOAD 2 ADD LOAD LSHIFT ADD SAVE
_exit_if_1_unionBlock__ JMP
:_false_if_1_unionBlock__
GETFP 1 SUB GETFP 2 ADD LOAD SAVE
GETFP 2 ADD GETFP 2 ADD LOAD 1 GETFP 2 ADD LOAD 2 ADD LOAD LSHIFT SUB SAVE
:_exit_if_1_unionBlock__
GETFP 1 SUB LOAD 3 ADD LOAD
1
CMP _true_if_2_unionBlock__ JEQ
_bool_0_unionBlock__ JMP
:_bool_0_unionBlock__
GETFP 2 ADD LOAD 2 ADD LOAD
GETFP 1 SUB LOAD 2 ADD LOAD
CMP _true_if_2_unionBlock__ JNE
_exit_if_2_unionBlock__ JMP

:_true_if_2_unionBlock__
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB 0 SAVE
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __unionBlock__ JMP
:_exit_if_2_unionBlock__
GETFP 2 ADD LOAD _unlinkBlock__ CALL GETRV DROP
GETFP 1 SUB LOAD _unlinkBlock__ CALL GETRV DROP
GETFP 2 ADD LOAD 2 ADD _inc__ CALL GETRV DROP
GETFP 2 ADD LOAD _insertBlock__ CALL GETRV DROP
GETFP 2 ADD LOAD _unionBlock__ CALL GETRV DROP
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB 0 SAVE
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __unionBlock__ JMP
:__unionBlock__
GETFP SETSP SETFP
1 RETN

:_allocBlock__
GETFP GETSP SETFP
3 PUSHN
GETFP 1 SUB 3 SAVE
:_loop_while_0_allocBlock__
GETFP 2 ADD LOAD
```

```
1 GETFP 1 SUB LOAD LSHIFT 1 3 LSHIFT SUB
CMP _true_while_0_allocBlock__ JGT
_exit_while_0_allocBlock__ JMP
:_true_while_0_allocBlock__
GETFP 1 SUB _inc__ CALL GETRV DROP
_loop_while_0_allocBlock__ JMP
:_exit_while_0_allocBlock__
GETFP 2 SUB GETFP 1 SUB LOAD SAVE
:_loop_while_1_allocBlock__
GETFP 2 SUB LOAD _getHeapBlock__ CALL GETRV LOAD
0
CMP _true_while_1_allocBlock__ JEQ
_exit_while_1_allocBlock__ JMP
:_true_while_1_allocBlock__
GETFP 2 SUB _inc__ CALL GETRV DROP
GETFP 2 SUB LOAD
_heap_order__ LOAD
CMP _true_if_0_allocBlock__ JGT
_exit_if_0_allocBlock__ JMP
:_true_if_0_allocBlock__
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB 0 SAVE
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __allocBlock__ JMP
:_exit_if_0_allocBlock__
_loop_while_1_allocBlock__ JMP
:_exit_while_1_allocBlock__
:_loop_while_2_allocBlock__
GETFP 2 SUB LOAD
GETFP 1 SUB LOAD
CMP _true_while_2_allocBlock__ JNE
_exit_while_2_allocBlock__ JMP
:_true_while_2_allocBlock__
GETFP 2 SUB LOAD _getHeapBlock__ CALL GETRV LOAD _splitBlock__ CALL GETRV DROP
GETFP 2 SUB _dec__ CALL GETRV DROP
_loop_while_2_allocBlock__ JMP
:_exit_while_2_allocBlock__
GETFP 3 SUB GETFP 1 SUB LOAD _getHeapBlock__ CALL GETRV LOAD SAVE
GETFP 3 SUB LOAD _unlinkBlock__ CALL GETRV DROP
GETFP 3 SUB GETFP 3 SUB LOAD 1 3 LSHIFT ADD SAVE
GETFP 3 SUB LOAD 1 SUB 1 SAVE
GETFP 3 SUB LOAD 2 SUB 0 SAVE
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB GETFP LOAD 3 SUB LOAD SAVE
```

```
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __allocBlock__ JMP
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB 0 SAVE
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __allocBlock__ JMP
:__allocBlock__
GETFP SETSP SETFP
1 RETN

:_deallocBlock__
GETFP GETSP SETFP
GETFP 2 ADD GETFP 2 ADD LOAD 1 3 LSHIFT SUB SAVE
GETFP 2 ADD LOAD _insertBlock__ CALL GETRV DROP
GETFP 2 ADD LOAD _unionBlock__ CALL GETRV DROP
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB 0 SAVE
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __deallocBlock__ JMP
:__deallocBlock__
GETFP SETSP SETFP
1 RETN

:_out__
GETFP GETSP SETFP
GETFP 2 ADD LOAD OUT
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB 0 SAVE
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __out__ JMP
:__out__
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
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB 0 SAVE
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __printInt__ JMP
:__printInt__
GETFP SETSP SETFP
1 RETN

:_newLine__
GETFP GETSP SETFP
10 _out__ CALL GETRV DROP
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB 0 SAVE
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __newLine__ JMP
:__newLine__
GETFP SETSP SETFP
JMP

:_space__
GETFP GETSP SETFP
32 _out__ CALL GETRV DROP
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB 0 SAVE
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __space__ JMP
:__space__
GETFP SETSP SETFP
JMP

:_printHeap__
GETFP GETSP SETFP
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB 3 SAVE
:_loop_while_0_printHeap__
GETFP 1 SUB LOAD
_heap_order__ LOAD
CMP _true_while_0_printHeap__ JLE
_exit_while_0_printHeap__ JMP
:_true_while_0_printHeap__
GETFP GETSP SETFP
```

```
2 PUSHN
GETFP 2 SUB 0 SAVE
GETFP 1 SUB GETFP LOAD 1 SUB LOAD _getHeapBlock__ CALL GETRV LOAD SAVE
:_loop_while_1_printHeap__
GETFP 1 SUB LOAD
0
CMP _true_while_1_printHeap__ JNE
_exit_while_1_printHeap__ JMP
:_true_while_1_printHeap__
GETFP 2 SUB _inc__ CALL GETRV DROP
GETFP 1 SUB GETFP 1 SUB LOAD _getNextBlock__ CALL GETRV SAVE
_loop_while_1_printHeap__ JMP
:_exit_while_1_printHeap__
GETFP 2 SUB LOAD _printInt__ CALL GETRV DROP
_space__ CALL GETRV DROP
GETFP SETSP SETFP
GETFP 1 SUB _inc__ CALL GETRV DROP
_loop_while_0_printHeap__ JMP
:_exit_while_0_printHeap__
_newLine__ CALL GETRV DROP
GETFP SETSP SETFP
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB 0 SAVE
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __printHeap__ JMP
:__printHeap__
GETFP SETSP SETFP
JMP

:_refInc__
GETFP GETSP SETFP
GETFP 2 ADD LOAD
0
CMP _true_if_0_refInc__ JNE
_exit_if_0_refInc__ JMP
:_true_if_0_refInc__
GETFP 2 ADD LOAD 1 SUB _inc__ CALL GETRV DROP
:_exit_if_0_refInc__
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB 0 SAVE
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __refInc__ JMP
:__refInc__
GETFP SETSP SETFP
```

```
1 RETN

:_refDec__
GETFP GETSP SETFP
GETFP 2 ADD LOAD
0
CMP _true_if_0_refDec__ JNE
_exit_if_0_refDec__ JMP
:_true_if_0_refDec__
GETFP 2 ADD LOAD 1 SUB _dec__ CALL GETRV DROP
GETFP 2 ADD LOAD 1 SUB LOAD
0
CMP _true_if_1_refDec__ JEQ
_exit_if_1_refDec__ JMP
:_true_if_1_refDec__
GETFP GETSP SETFP
2 PUSHN
GETFP 2 SUB GETFP LOAD 2 ADD LOAD 2 SUB LOAD SAVE
GETFP 1 SUB 0 SAVE
:_loop_while_0_refDec__
GETFP 1 SUB LOAD
GETFP 2 SUB LOAD
CMP _true_while_0_refDec__ JNE
_exit_while_0_refDec__ JMP
:_true_while_0_refDec__
GETFP LOAD 2 ADD LOAD GETFP 1 SUB LOAD ADD LOAD _refDec__ CALL GETRV DROP
GETFP 1 SUB _inc__ CALL GETRV DROP
_loop_while_0_refDec__ JMP
:_exit_while_0_refDec__
GETFP SETSP SETFP
GETFP 2 ADD LOAD _deallocBlock__ CALL GETRV DROP
:_exit_if_1_refDec__
:_exit_if_0_refDec__
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB 0 SAVE
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __refDec__ JMP
:__refDec__
GETFP SETSP SETFP
1 RETN

:_prepareHeap__
GETFP GETSP SETFP
GETFP GETSP SETFP
2 PUSHN
```

```
GETFP 1 SUB 1 3 LSHIFT SAVE
GETFP 2 SUB _MEMORY_SIZE LOAD _PROGRAM_SIZE LOAD 64 ADD SUB SAVE
_heap_order__ LOAD _getHeapBlock__ CALL GETRV 0 SAVE
:_loop_while_0_prepareHeap__
GETFP 1 SUB LOAD
GETFP 2 SUB LOAD
CMP _true_while_0_prepareHeap__ JLE
_exit_while_0_prepareHeap__ JMP
:_true_while_0_prepareHeap__
_heap_order__ LOAD _getHeapBlock__ CALL GETRV 0 SAVE
_heap_order__ _inc__ CALL GETRV DROP
GETFP 1 SUB GETFP 1 SUB LOAD 1 LSHIFT SAVE
_loop_while_0_prepareHeap__ JMP
:_exit_while_0_prepareHeap__
_heap_order__ _dec__ CALL GETRV DROP
GETFP SETSP SETFP
_heap_start__ _heap_order__ LOAD _getHeapBlock__ CALL GETRV 1 ADD SAVE
_heap_start__ LOAD 2 ADD _heap_order__ LOAD SAVE
_heap_start__ LOAD _insertBlock__ CALL GETRV DROP
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB 0 SAVE
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __prepareHeap__ JMP
:__prepareHeap__
GETFP SETSP SETFP
JMP

:_genNode
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB 0 SAVE
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB GETFP LOAD 1 SUB SAVE
GETFP 1 SUB LOAD LOAD _refDec__ CALL GETRV DROP
GETFP 1 SUB LOAD 2 _allocBlock__ CALL GETRV SAVE
GETFP 1 SUB LOAD LOAD 2 SUB 1 SAVE
GETFP SETSP SETFP
GETFP 1 SUB LOAD 0 ADD 0 SAVE
GETFP 1 SUB LOAD 1 ADD GETFP 2 ADD LOAD SAVE
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB GETFP LOAD 1 SUB LOAD SAVE
GETFP 1 SUB LOAD _refInc__ CALL GETRV DROP
GETFP LOAD 1 SUB LOAD _refDec__ CALL GETRV DROP
```

```
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __genNode JMP
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB 0 SAVE
GETFP LOAD 1 SUB LOAD _refDec__ CALL GETRV DROP
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __genNode JMP
:__genNode
GETFP SETSP SETFP
1 RETN

:_linkNodes
GETFP GETSP SETFP
GETFP GETSP SETFP
2 PUSHN
GETFP 2 SUB GETFP LOAD 3 ADD LOAD SAVE
GETFP 1 SUB GETFP LOAD 2 ADD LOAD 0 ADD SAVE
GETFP 2 SUB LOAD _refInc__ CALL GETRV DROP
GETFP 1 SUB LOAD LOAD _refDec__ CALL GETRV DROP
GETFP 1 SUB LOAD GETFP 2 SUB LOAD SAVE
GETFP SETSP SETFP
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB 0 SAVE
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __linkNodes JMP
:__linkNodes
GETFP SETSP SETFP
2 RETN

:_printLinkedList
GETFP GETSP SETFP
:_loop_while_0_printLinkedList
GETFP 2 ADD LOAD
0
CMP _true_while_0_printLinkedList JNE
_exit_while_0_printLinkedList JMP
:_true_while_0_printLinkedList
GETFP 2 ADD LOAD 1 ADD LOAD _printInt__ CALL GETRV DROP
_space__ CALL GETRV DROP
GETFP 2 ADD GETFP 2 ADD LOAD 0 ADD LOAD SAVE
_loop_while_0_printLinkedList JMP
:_exit_while_0_printLinkedList
_newLine__ CALL GETRV DROP
GETFP GETSP SETFP
```

```
1 PUSHN
GETFP 1 SUB 0 SAVE
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __printLinkedList JMP
:__printLinkedList
GETFP SETSP SETFP
1 RETN

:_main
GETFP GETSP SETFP
4 PUSHN
GETFP 1 SUB 0 SAVE
GETFP 2 SUB 0 SAVE
GETFP 3 SUB 0 SAVE
GETFP 4 SUB 0 SAVE
_prepareHeap__ CALL GETRV DROP
_printHeap__ CALL GETRV DROP
GETFP GETSP SETFP
2 PUSHN
GETFP 2 SUB 100500 _genNode CALL GETRV SAVE
GETFP 1 SUB GETFP LOAD 2 SUB SAVE
GETFP 1 SUB LOAD LOAD _refDec__ CALL GETRV DROP
GETFP 1 SUB LOAD GETFP 2 SUB LOAD SAVE
GETFP SETSP SETFP
GETFP GETSP SETFP
2 PUSHN
GETFP 2 SUB 1501 _genNode CALL GETRV SAVE
GETFP 1 SUB GETFP LOAD 3 SUB SAVE
GETFP 1 SUB LOAD LOAD _refDec__ CALL GETRV DROP
GETFP 1 SUB LOAD GETFP 2 SUB LOAD SAVE
GETFP SETSP SETFP
GETFP GETSP SETFP
2 PUSHN
GETFP 2 SUB 666 _genNode CALL GETRV SAVE
GETFP 1 SUB GETFP LOAD 4 SUB SAVE
GETFP 1 SUB LOAD LOAD _refDec__ CALL GETRV DROP
GETFP 1 SUB LOAD GETFP 2 SUB LOAD SAVE
GETFP SETSP SETFP
GETFP 3 SUB LOAD GETFP 2 SUB LOAD _linkNodes CALL GETRV DROP
GETFP 4 SUB LOAD GETFP 3 SUB LOAD _linkNodes CALL GETRV DROP
GETFP GETSP SETFP
2 PUSHN
GETFP 2 SUB GETFP LOAD 3 SUB LOAD SAVE
GETFP 1 SUB GETFP LOAD 1 SUB SAVE
GETFP 2 SUB LOAD _refInc__ CALL GETRV DROP
GETFP 1 SUB LOAD LOAD _refDec__ CALL GETRV DROP
```

```
GETFP 1 SUB LOAD GETFP 2 SUB LOAD SAVE
GETFP SETSP SETFP
GETFP 2 SUB LOAD _printLinkedList CALL GETRV DROP
GETFP 3 SUB LOAD _printLinkedList CALL GETRV DROP
GETFP 4 SUB LOAD _printLinkedList CALL GETRV DROP
GETFP 1 SUB LOAD _printLinkedList CALL GETRV DROP
GETFP GETSP SETFP
2 PUSHN
GETFP 2 SUB 0 SAVE
GETFP 1 SUB GETFP LOAD 2 SUB SAVE
GETFP 2 SUB LOAD _refInc__ CALL GETRV DROP
GETFP 1 SUB LOAD LOAD _refDec__ CALL GETRV DROP
GETFP 1 SUB LOAD GETFP 2 SUB LOAD SAVE
GETFP SETSP SETFP
GETFP GETSP SETFP
2 PUSHN
GETFP 2 SUB 0 SAVE
GETFP 1 SUB GETFP LOAD 3 SUB SAVE
GETFP 2 SUB LOAD _refInc__ CALL GETRV DROP
GETFP 1 SUB LOAD LOAD _refDec__ CALL GETRV DROP
GETFP 1 SUB LOAD GETFP 2 SUB LOAD SAVE
GETFP SETSP SETFP
GETFP GETSP SETFP
2 PUSHN
GETFP 2 SUB 0 SAVE
GETFP 1 SUB GETFP LOAD 4 SUB SAVE
GETFP 2 SUB LOAD _refInc__ CALL GETRV DROP
GETFP 1 SUB LOAD LOAD _refDec__ CALL GETRV DROP
GETFP 1 SUB LOAD GETFP 2 SUB LOAD SAVE
GETFP SETSP SETFP
GETFP GETSP SETFP
2 PUSHN
GETFP 2 SUB 0 SAVE
GETFP 1 SUB GETFP LOAD 1 SUB SAVE
GETFP 2 SUB LOAD _refInc__ CALL GETRV DROP
GETFP 1 SUB LOAD LOAD _refDec__ CALL GETRV DROP
GETFP 1 SUB LOAD GETFP 2 SUB LOAD SAVE
GETFP SETSP SETFP
_printHeap__ CALL GETRV DROP
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB 0 SAVE
GETFP LOAD 1 SUB LOAD _refDec__ CALL GETRV DROP
GETFP LOAD 2 SUB LOAD _refDec__ CALL GETRV DROP
GETFP LOAD 3 SUB LOAD _refDec__ CALL GETRV DROP
GETFP LOAD 4 SUB LOAD _refDec__ CALL GETRV DROP
```

```
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV SETRV __main JMP
GETFP GETSP SETFP
1 PUSHN
GETFP 1 SUB 0 SAVE
GETFP LOAD 1 SUB LOAD _refDec__ CALL GETRV DROP
GETFP LOAD 2 SUB LOAD _refDec__ CALL GETRV DROP
GETFP LOAD 3 SUB LOAD _refDec__ CALL GETRV DROP
GETFP LOAD 4 SUB LOAD _refDec__ CALL GETRV DROP
GETFP 1 SUB LOAD SETRV GETFP SETSP SETFP
GETRV 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 4096
`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


        // инициализация 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
*$FROM src/OOP/OOP_Program.ref
*$FROM src/DYN/DYN_Program.ref
$EXTERN LoadExpr, SaveFile, Program, OOP_Program, DYN_Program;

$ENTRY Go {
  /* пусто */ = <SaveFile (<Arg 2>) (
    <Program
      <OOP_Program
        <DYN_Program
```

```
          <LoadExpr <Arg 1>>
        >
      >
    >
  )>;
}
```

run.sh

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

source.txt

```
(dynvar LinkedListNode
    (refs LLN_next)
    (LLN_value 1)
)

(function genNode (val)
    (var (refs list))

    (gc-alloc list LinkedListNode)

    (((L list) "+" LLN_next) "=" 0)
    (((L list) "+" LLN_value) "=" (L val))

    (ref-return (L list))
)

(function linkNodes (node1 node2)
    (((L node1) "+" LLN_next) ":-" (L node2))
)

(function printLinkedList (linkedList)
    (while ((L linkedList) "<>" 0)
        (call printInt__ (L ((L linkedList) "+" LLN_value)))
        (call space__)
        (linkedList "=" (L ((L linkedList) "+" LLN_next)))
    )
    (call newLine__)
)

(function main ()
    (var (refs list node1 node2 node3))
    (call printHeap__)
```

```
    (node1 ":-" (call genNode 100500))
    (node2 ":-" (call genNode 1501))
    (node3 ":-" (call genNode 666))
    (call linkNodes (L node1) (L node2))
    (call linkNodes (L node2) (L node3))

    (list ":-" (L node2))

    (call printLinkedList (L node1))
    (call printLinkedList (L node2))
    (call printLinkedList (L node3))
    (call printLinkedList (L list))

    (node1 ":-" 0)
    (node2 ":-" 0)
    (node3 ":-" 0)
    (list ":-" 0)


    (call printHeap__)
    (return 0)
)
```

stdin.txt

Тестирование

source.txt

```
(dynvar LinkedListNode
    (refs LLN_next)
    (LLN_value 1)
)

(function genNode (val)
    (var (refs list))

    (gc-alloc list LinkedListNode)

    (((L list) "+" LLN_next) "=" 0)
    (((L list) "+" LLN_value) "=" (L val))
```

```
    (ref-return (L list))
)

(function linkNodes (node1 node2)
    (((L node1) "+" LLN_next) ":-" (L node2))
)

(function printLinkedList (linkedList)
    (while ((L linkedList) "<>" 0)
        (call printInt__ (L ((L linkedList) "+" LLN_value)))
        (call space__)
        (linkedList "=" (L ((L linkedList) "+" LLN_next)))
    )
    (call newLine__)
)

(function main ()
    (var (refs list node1 node2 node3))
    (call printHeap__)


    (node1 ":-" (call genNode 100500))
    (node2 ":-" (call genNode 1501))
    (node3 ":-" (call genNode 666))
    (call linkNodes (L node1) (L node2))
    (call linkNodes (L node2) (L node3))

    (list ":-" (L node2))

    (call printLinkedList (L node1))
    (call printLinkedList (L node2))
    (call printLinkedList (L node3))
    (call printLinkedList (L list))

    (node1 ":-" 0)
    (node2 ":-" 0)
    (node3 ":-" 0)
    (list ":-" 0)


    (call printHeap__)
    (return 0)
)
```

cmd stdout

```
*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:
*Compiling src/DYN/DYN_Program.ref:
*Compiling src/DYN/DYN_Definition.ref:
*Compiling src/DYN/DYN_Function.ref:
*Compiling src/DYN/DYN_LocalVars.ref:
*Compiling src/DYN/DYN_Code.ref:
*Compiling src/DYN/DYN_Statement.ref:
*Compiling src/DYN/DYN_Expr.ref:
*Compiling src/DYN/DYN_BoolExpr.ref:
*Compiling src/DYN/DYN_GlobalRefs.ref:
*Compiling src/DYN/DYN_DynVarType.ref:
*Compiling src/DYN/DYN_Class.ref:
** Compilation succeeded **
rm: cannot remove '*.rasl': No such file or directory
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
100500 1501 666
1501 666
666
```

```
1501 666
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1


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

Вывод

В результате выполнения данной работы были изучены способы организации кучи и алгоритмов сборки мусора.