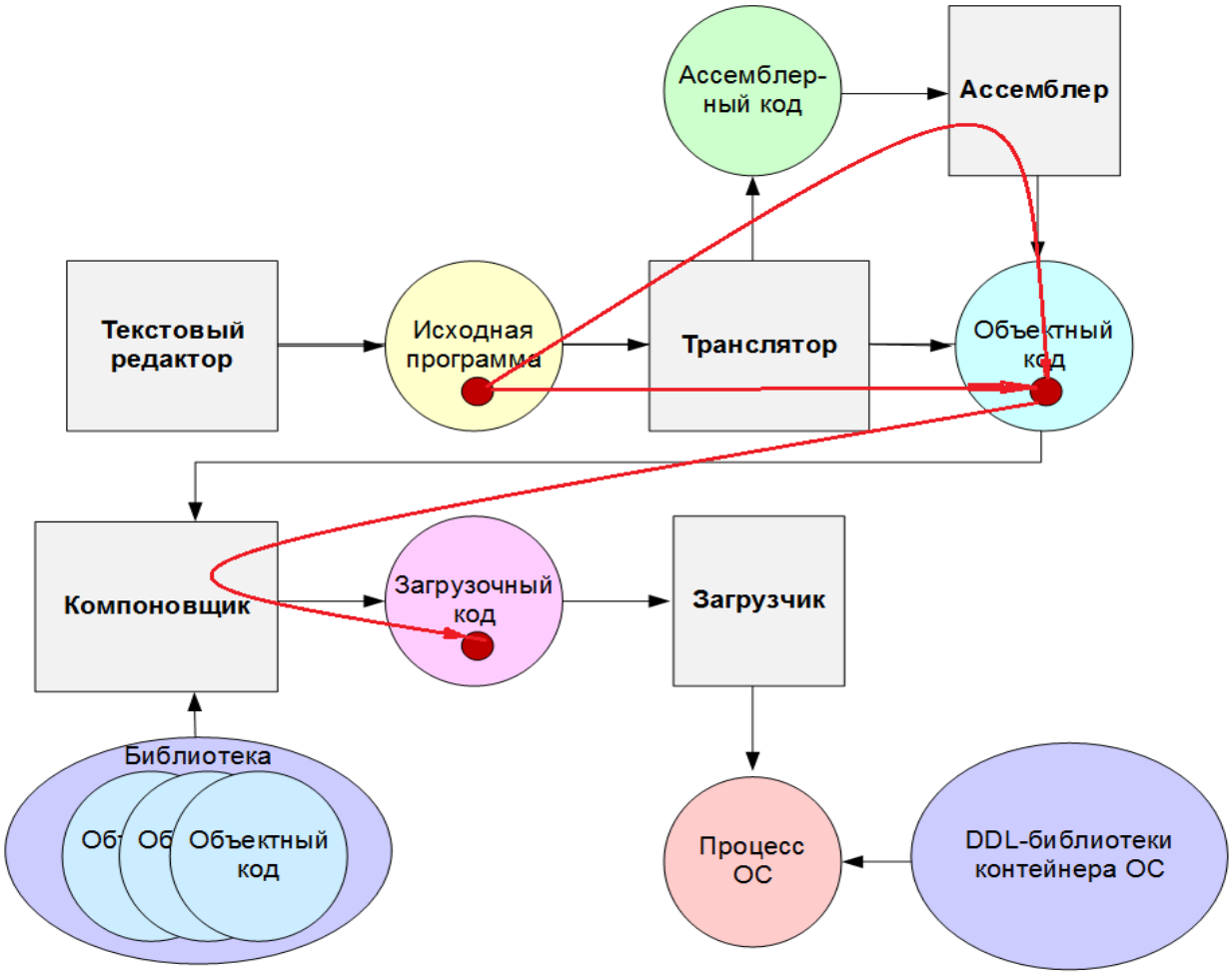
Лекция 22
БГТУ, ФИТ, ПОИТ, 3 семестр
Конструирование программного обеспечения

Генерация кода

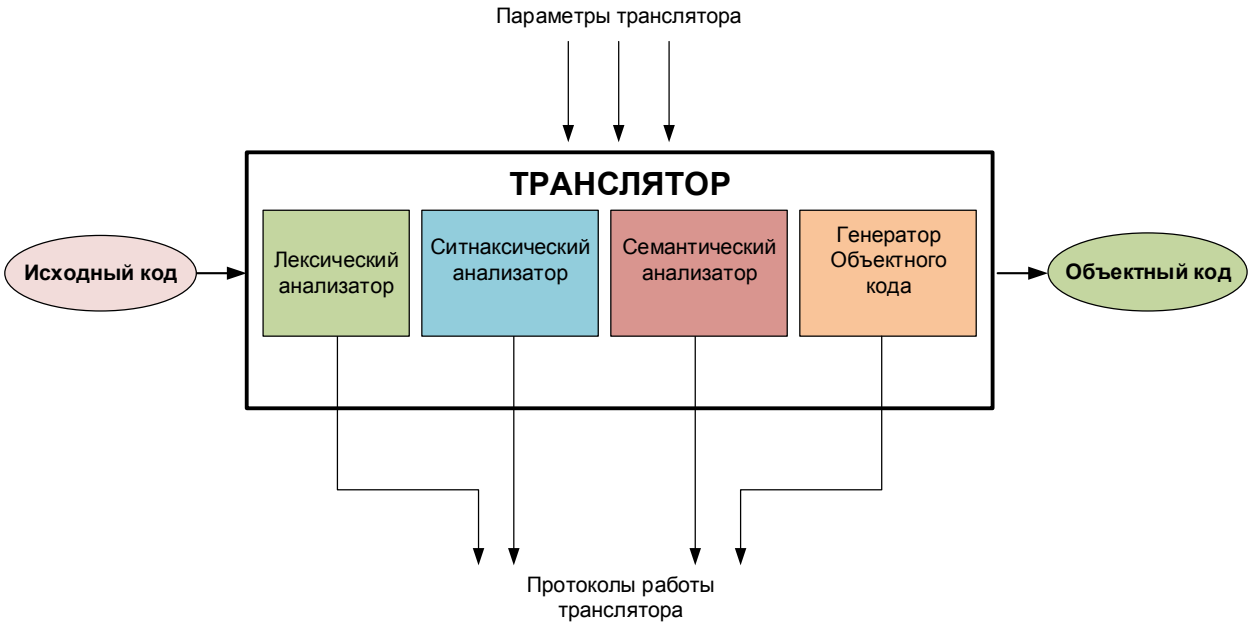
1. Система программирования

Структура системы программирования:

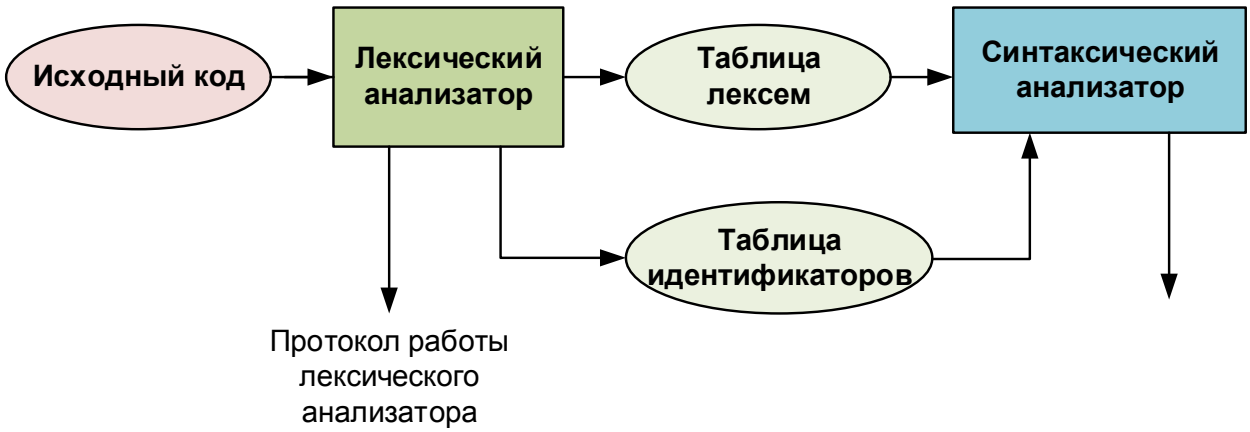


2. **Транслятор:** общая схема, фазы трансляции

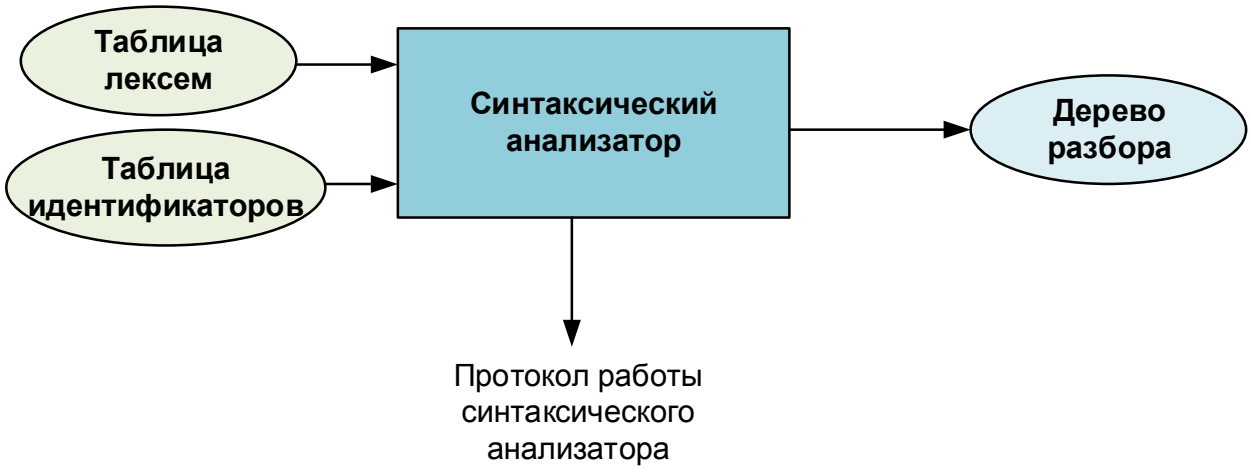
Общая схема транслятора:

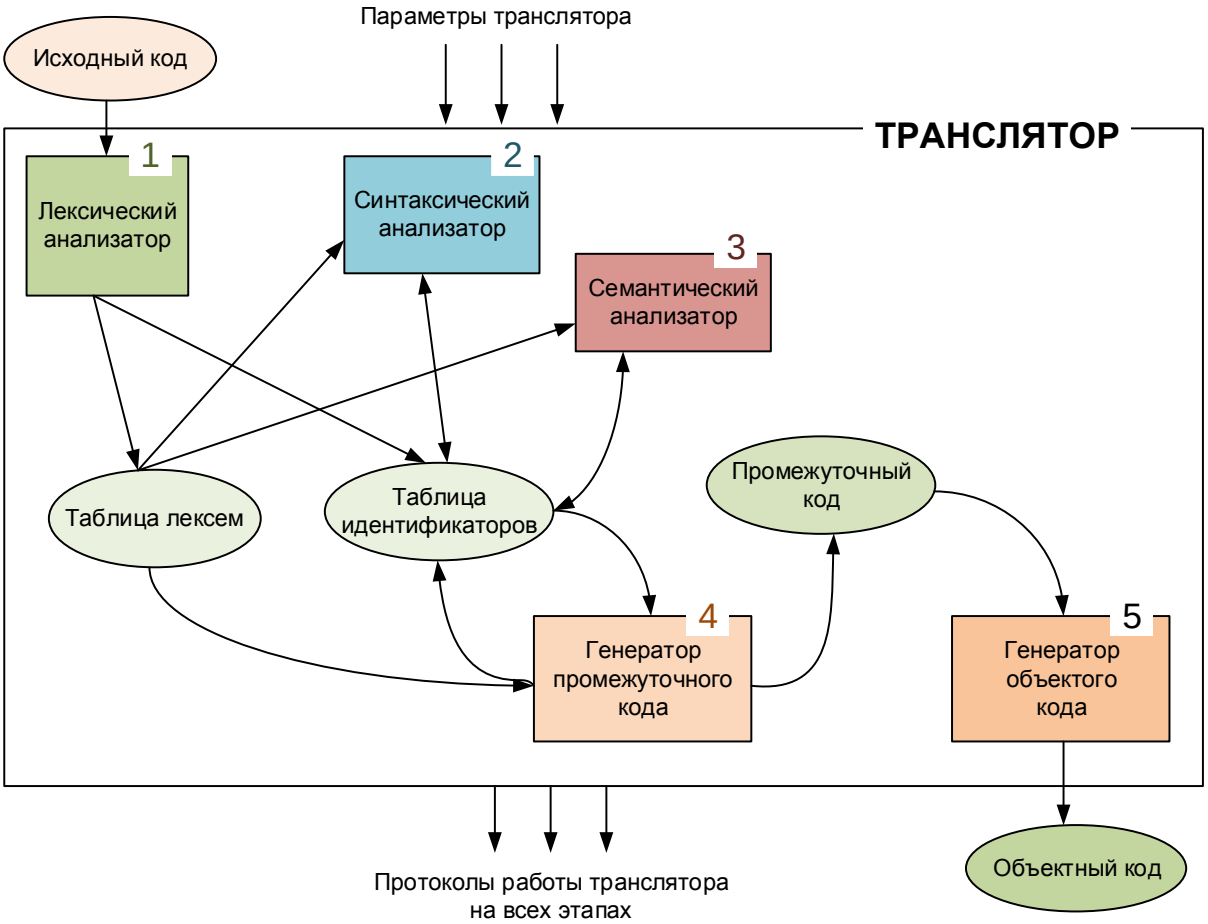


Фазы трансляции (лексический анализ):



Фазы трансляции (синтаксический анализ):





Основной задачей генератора кода является преобразование кода из промежуточного представления в эквивалентный код для целевой машины.

Входом генератора кода является промежуточное представление исходной программы.

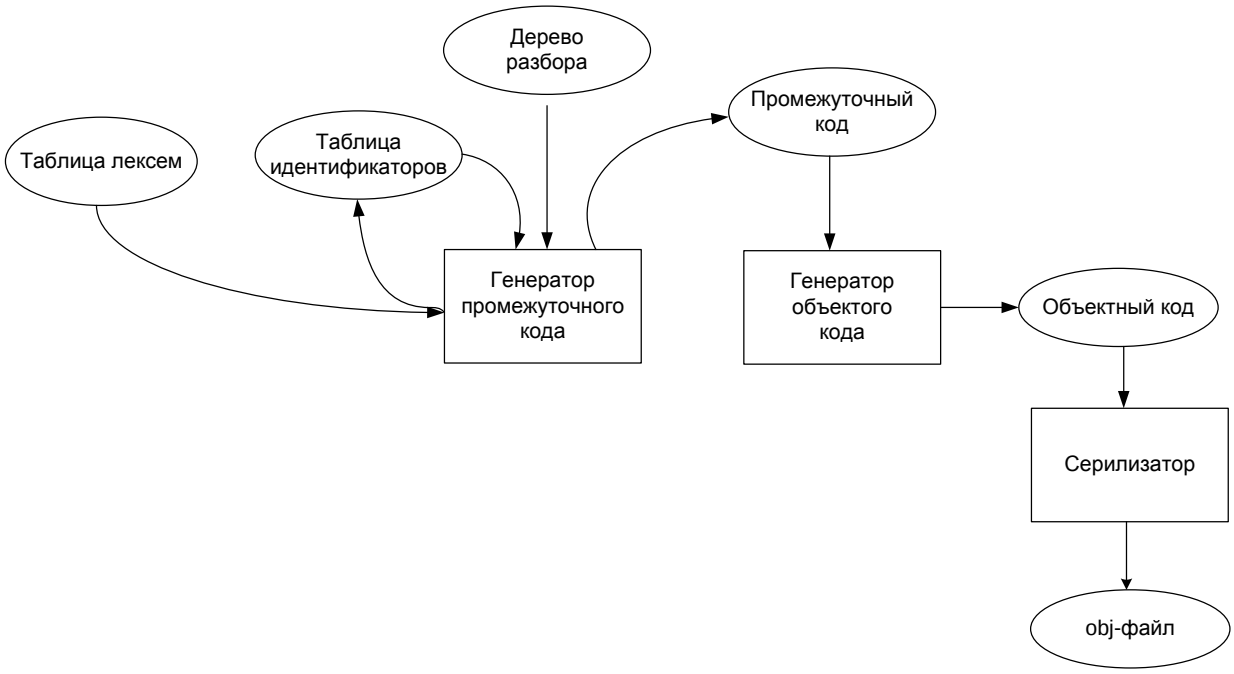
Выходом генератора кода является код для целевой машины, эквивалентный исходному коду.

Генерацию кода можно разделить на две части:

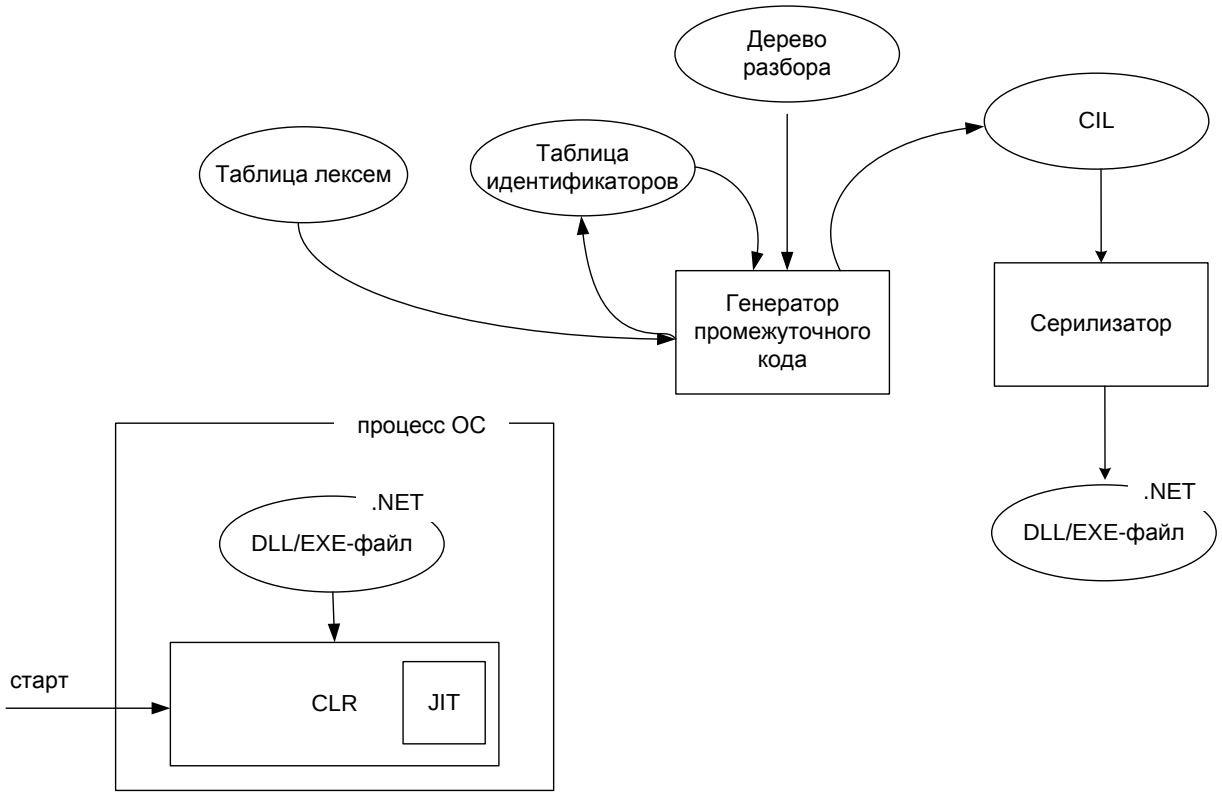
1. генерация промежуточного кода (не зависит от платформы);
2. генерация объектного кода на основе промежуточного (для конкретной платформы).

Процесс генерации кода может управляться параметрами.

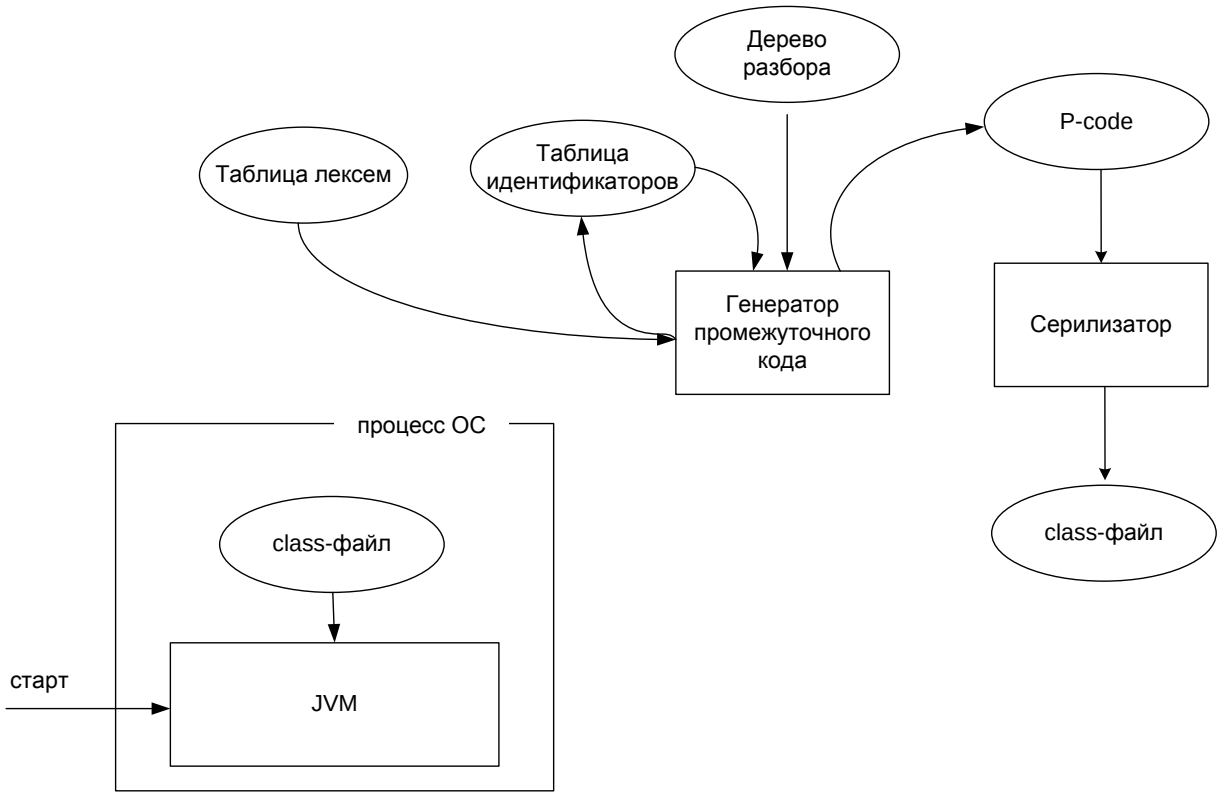
3. Генерация кода C/C++



4. Генерация кода языка .NET (C#, VB.NET)



5. Генерация кода языка Java

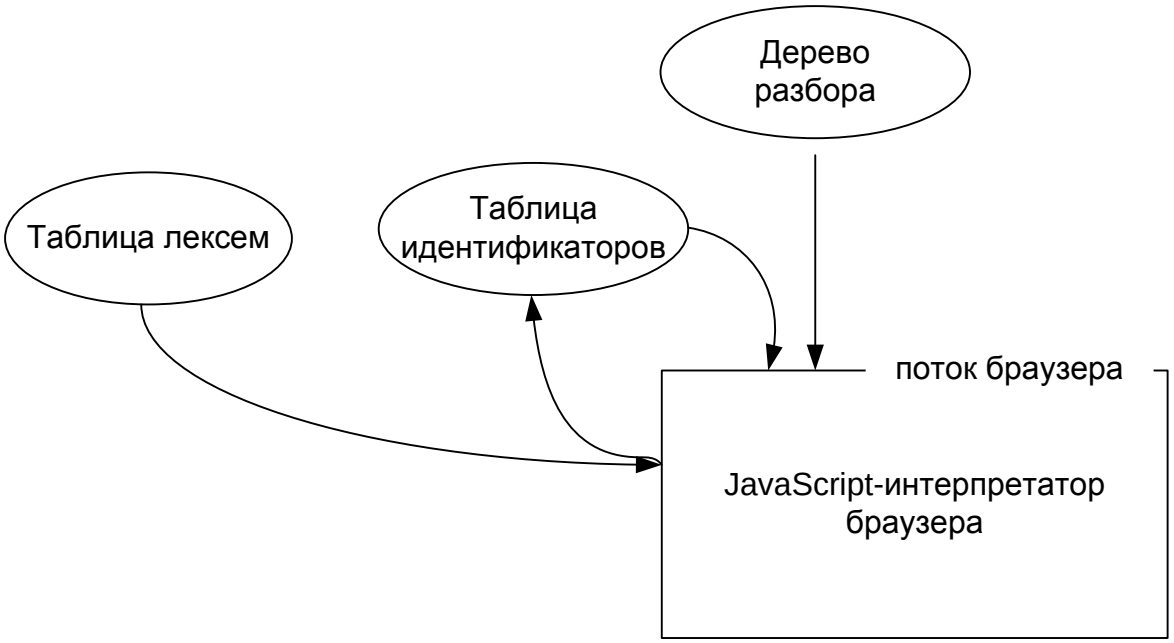


6. **Сериализация:** процесс преобразование структуры данных в последовательность битов.

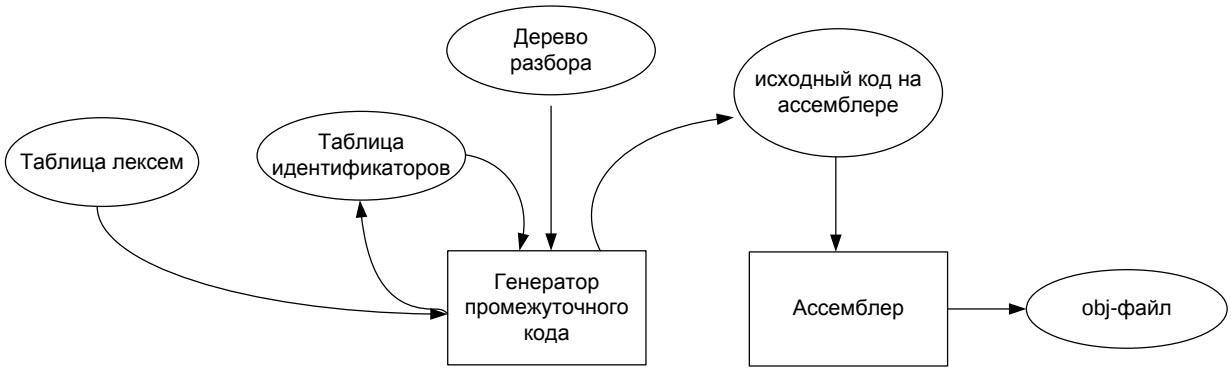
7. **Десериализация:** процесс преобразования последовательности битов в структуру данных.

8. Курсовой проект

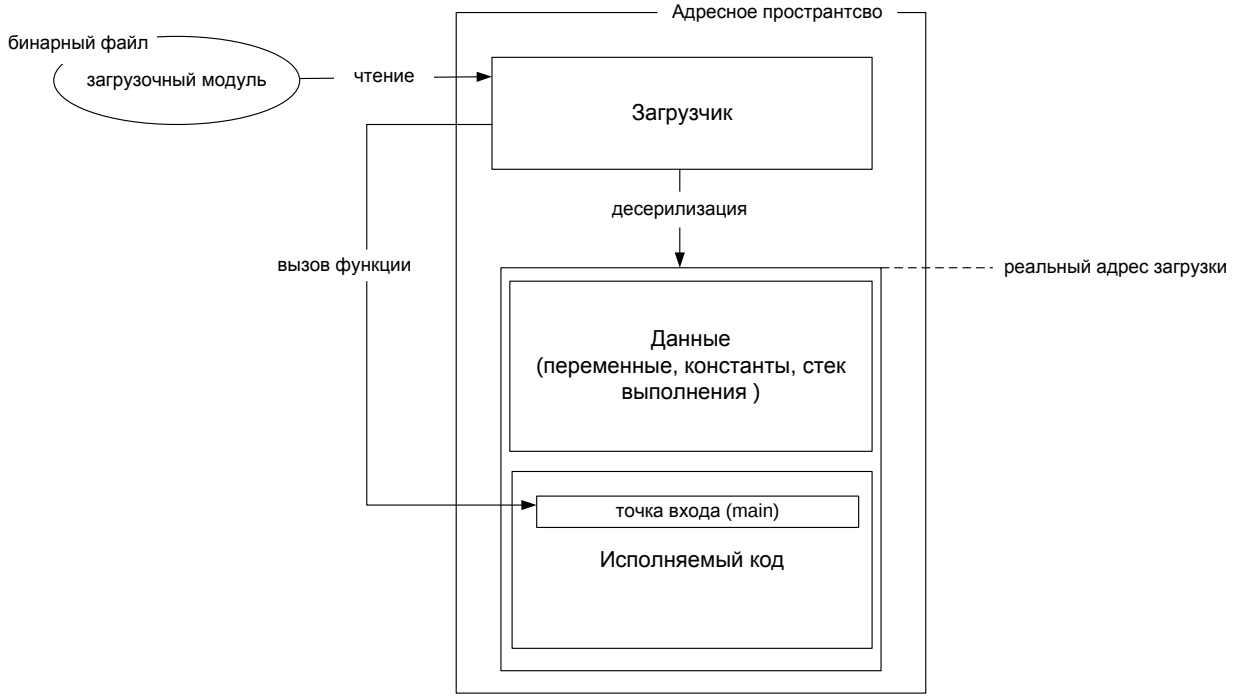
8.1 Интерпретация кода JavaScript



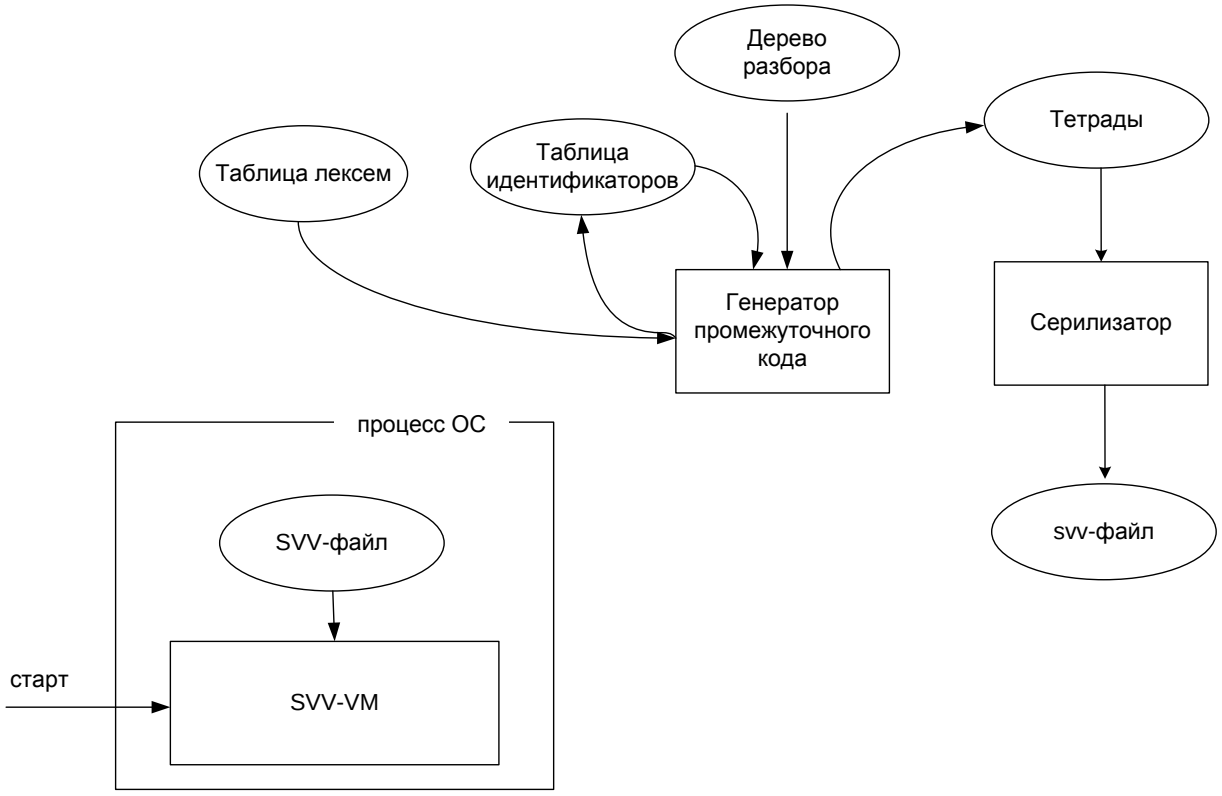
8.2 Генерация исходного ассемблерного кода



8.3 **Загрузка/десериализация**

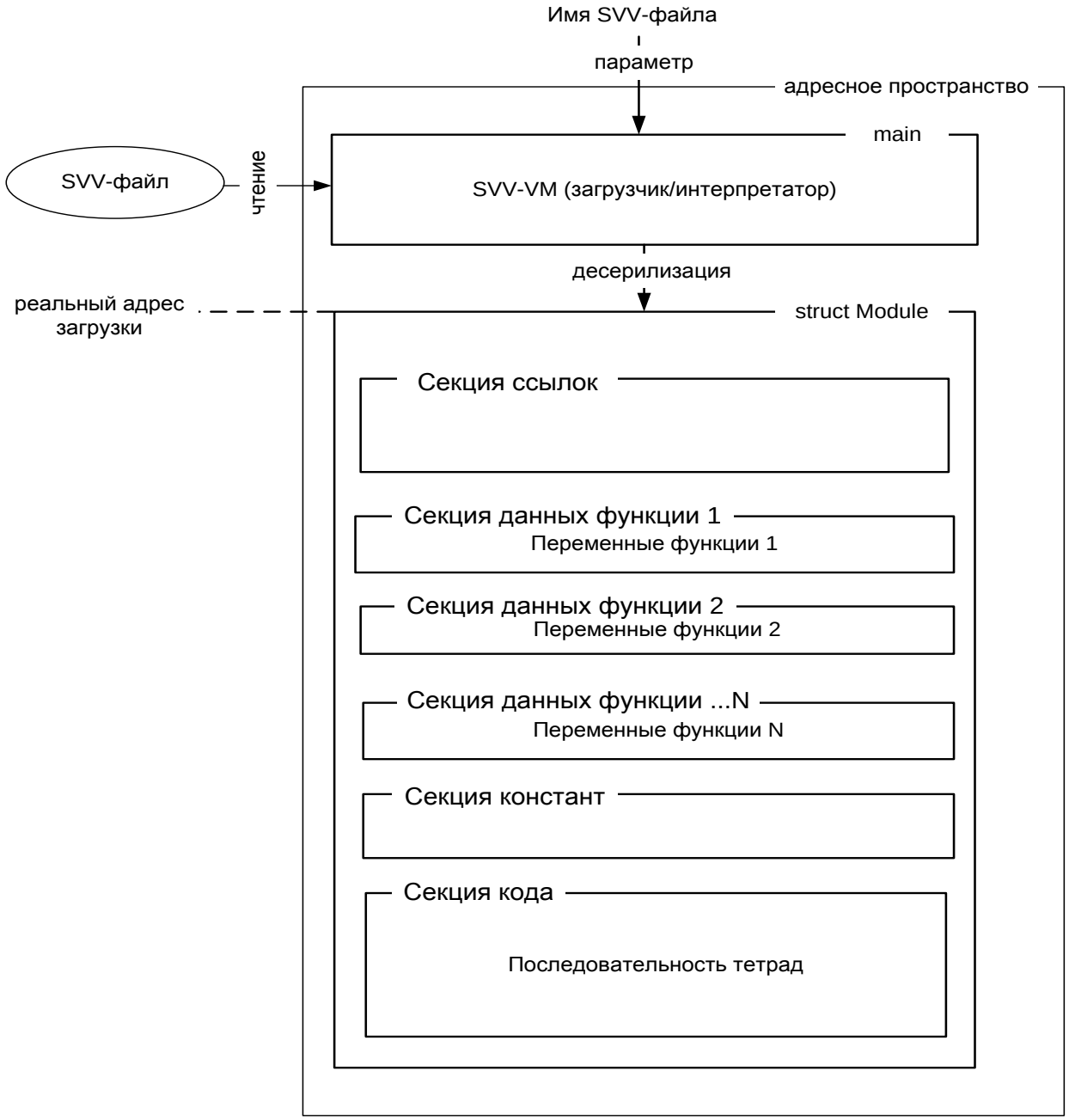


9. Принцип реализации SVV-2015



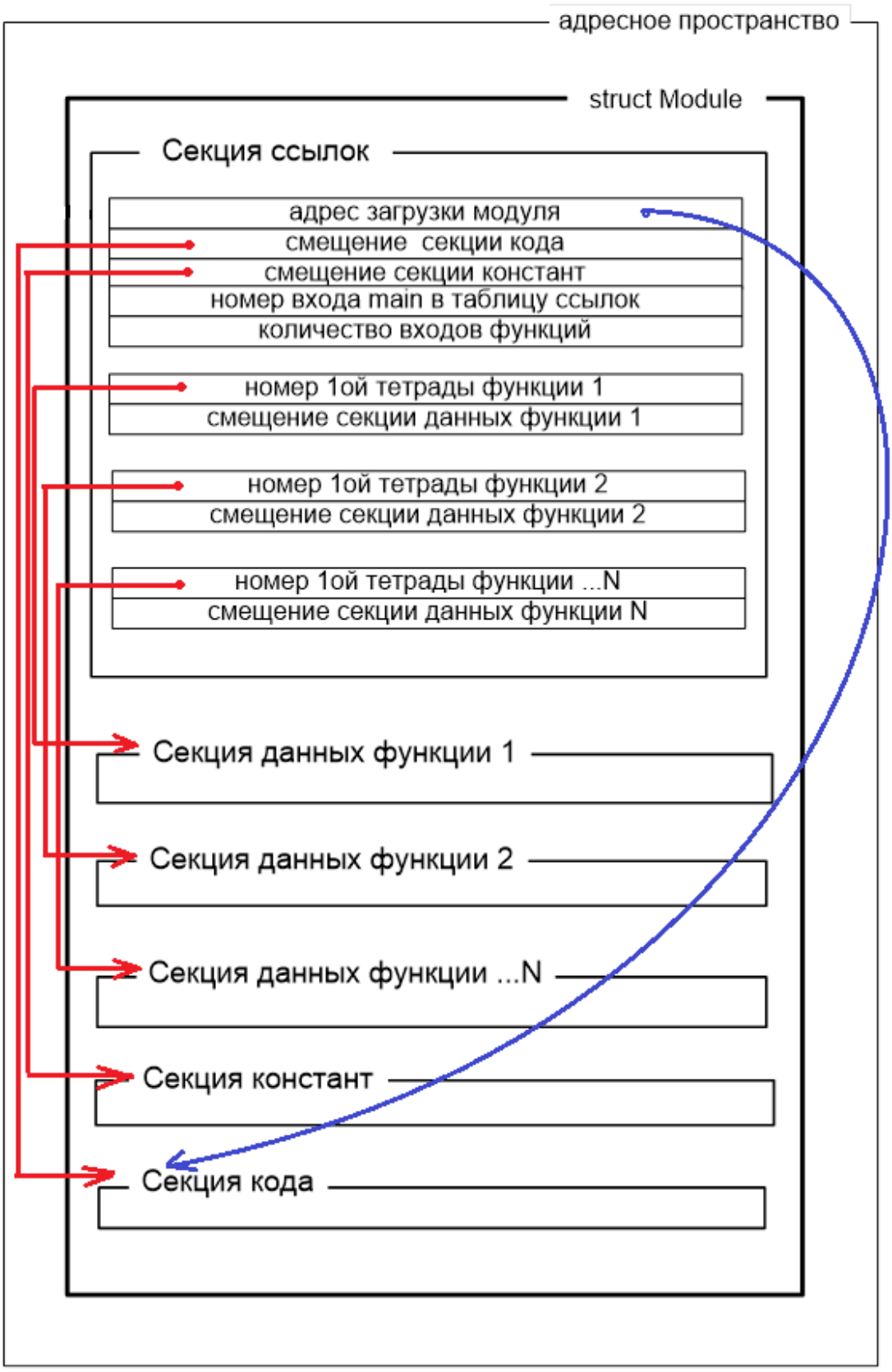
Генератор промежуточного кода упакует тетрады и сохранит в файл.
Загрузчик – десериализует, разместит в памяти и передаст управление.

9.1 **Модель памяти SVV-2015** (модель памяти статическая)



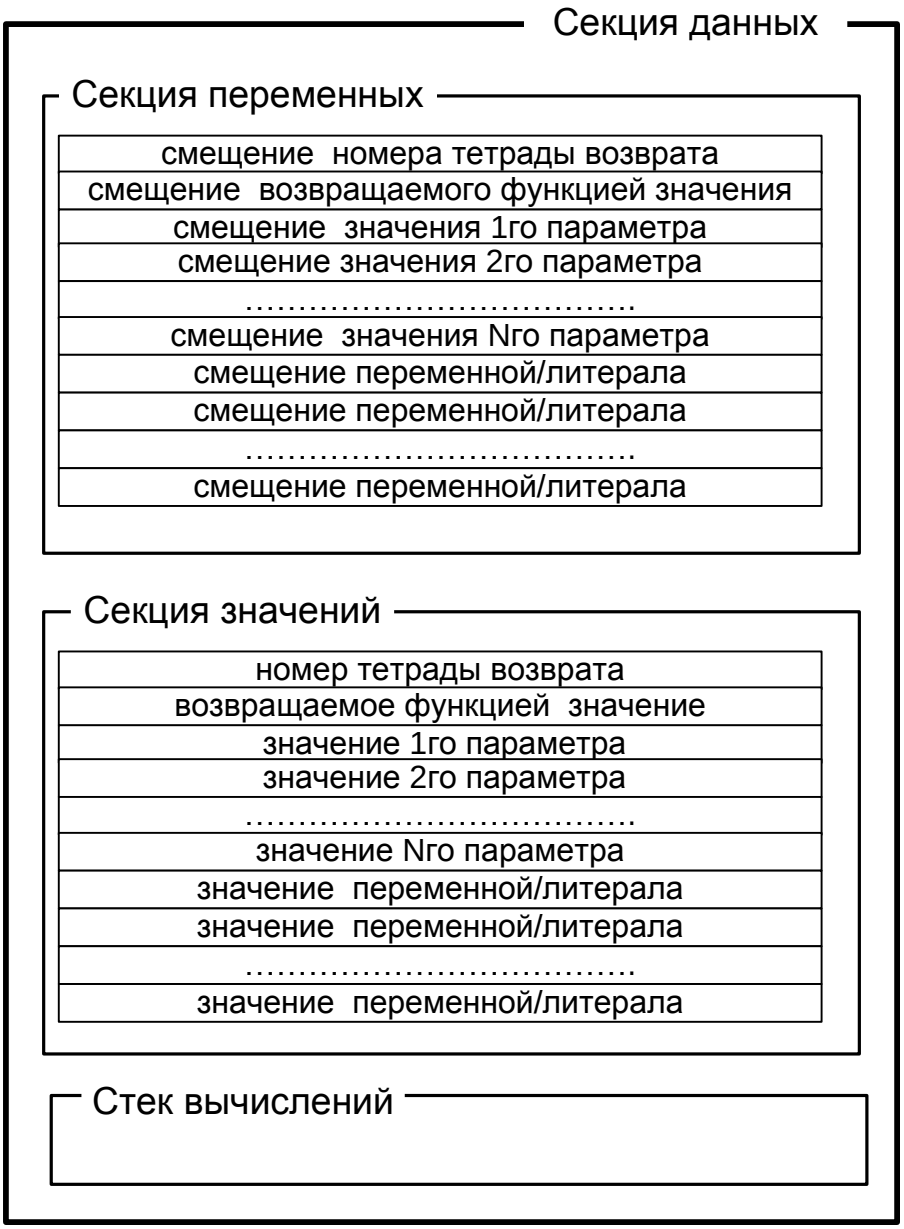

```
struct Module
{
     RefSection   refsection;                // секция ссылок
     DataSection  datasections[GEN_MAX_FUNC]; // секции данных в количестве refsection.functablesize
     ConstSection constsection;              // секция констант
     CodeSection  codesection;               // секция кода
     Module();
     void Serilization(char * filename){};   // серилизация                //
     void DeSerilization(char * filename){}; // десерилизация


     void zapvar(int offvar, int offval,  int offconst, int   initval);
     void zaptetrad(int offcode,  Tetrad t);

};
```

9.2 **Секция ссылок SVV-2015**



```
struct RefSection                  // секция ссылок
  {
    void* loadaddress;              // Run: адрес  структуры Module
    int   offcodesection;           // Gen: смещение секции кода
    int   offconstsection;          // Gen: смещение кода констант
    short mainentrynumber;          // Gen:  номер входа main
    short functablesize;            // Gen: = количеству функции из ТИ
    struct FuncSection              // Gen: входы для функций
    {
      short tfirst;                 // Gen: первая тетрада функции
      int   offdatasection;         // Gen: смещение секции данных функции
      FuncSection();
      FuncSection(short t, int ods);
    } functable[GEN_MAX_FUNC];      // Gen:  таблица функций
    RefSection();
    RefSection(Module* pmodule);
  };
```

9.3 **Секция данных**



```
struct VarSection    //  секция переменных
 {
   int  offtret;            //  Gen:  смещение номера тетрады возврата
   int  offrc;              //  Gen:  смещение возвращаемого значения
   int  offmem[GEN_MAX_ID]; // Gen: смещения параметров и переменных
   VarSection();
   VarSection(Module *pmodule);
   char* Serilization(){};       // серилизация                 //
   char* DeSerilization(){};     // десерилизация
 };
struct ValSection     // секция значений
 {
    int  nt_ret;           // Run: тетрада для возврата
    char mem[GEN_MAX_VAL]; // Run: память для параметров и переменных
    char* Serilization(){};       // серилизация                 //
    char* DeSerilization(){};     // десерилизация
 };

typedef std::stack<void*> CalcStackSection;
struct DataSection
{
    VarSection varsection;    // Gen   секция переменных
    ValSection valsection;    // Run   секция значений
    CalcStackSection calcstacksection; // Run
    DataSection();
    char* Serilization(){};       // серилизация                 //
    char* DeSerilization(){};     // десерилизация
};
```

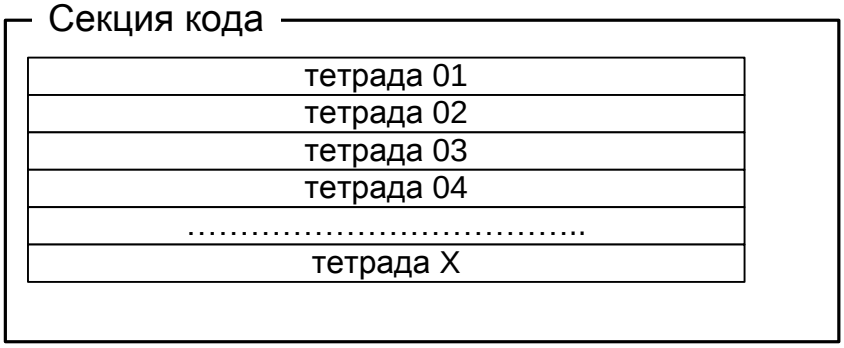
9.4 **Секция констант**

Секция констант

значение инициализации/литерала
значение инициализации/литерала
значение инициализации/литерала
……
значение инициализации/литерала

```
struct ConstSection     // секция констант
{
   char mem[GEN_MAX_CONST];     //Gen
   char* Serilization(){};       // серилизация
   char* DeSerilization(){};     // десерилизация
};
```

9.5 **Секция кода**




```
enum TETRADETYPE  // типы тетерад
 {
    NILL = 0,    // пустая
    FUNC = 1,    // инициализация функции
    RETN = 2,    // return - записать код возврата и перейти по тетраде возврата
    PARM = 3,    // инииализация параметра функции
    DTI  = 4,    // инициализация переменной
    MAIN = 5,    // инициализация главной функции
    CALLPARM = 6, // записать  значение параметра перед вызовом
    CALL = 7    // вызов функции: записать номер тетрады возврата и перейти
};
struct Tetrad   // тетрада
{
   TETRADETYPE type;    // тип тетрады
   void* result;        // результат ???
   void* parm[4];       // параметры тетрады
   Tetrad() { ... }
   Tetrad(TETRADETYPE t, int p0, int p1, int p2, int p3) { ... }
   char* Serilization(){};        // серилизация
   char* DeSerilization(){};      // десерилизация
};
struct CodeSection   // секция кода
{
   int size;          // количество тетрад
   Tetrad  tetrad[GEN_MAX_TETR];     //Gen
   char* Serilization(){};        // серилизация
   char* DeSerilization(){};      // десерилизация
 };
```

9.6 **Процесс генерации памяти**

```
struct Genstate                     // состояние процесса генерации
{
   short size_funcdatasections;     // количество секций = количеству фцнкций
   struct FuncDataSection           // смещения  в секции данных
   {
     int next_varsection;           // смещение свободной памяти в секции переменных
     int next_valsection;           // смещение свободной памяти в секции значений
   } *funcdatasections;
   int  next_constsection;          // смещение свободной памяти в секции констант
   int  next_codesection;           // смещение свободной памяти в секции  кода
};
```

```
Module* generate(   //  генерация кода
                LEX::LEX& lex,                  // таблицы лексем и идентификаторов
                Mfst::Deducation deduct         // правила вывода
                );
```

9.7 **Расширения таблицы идентификаторов**

```
struct Entry      // строка таблицы идентификаторов
{
 short       idxfirstLE;         // индекс  первой строки в таблице лексем
 char        id[2*ID_MAXSIZE+1];     // идентификатор (автоматически усекается до  ID_MAXSIZE) + префикс_функции#
 IDDATATYPE  iddatatype;         // тип данных
 IDTYPE      idtype;             // тип идентикатора
 union { ... } value;        // значение идентификатора
 struct GenFuncExt   //заполняется при генерации функции
 {
  int offrcvar;   // смещение адреса возвращаемого значения
  int offrcval;   // смещение возвращаемого значения
  int offretvar;  // смещение адреса для тетрады возврата
  int offretval;  // смещение значения тетрады возврата
 } genfuncext;    // расширение, заполняемое при генерации кода

 struct GenVarExt  //заполняется при генерации переменных
 {
  int offvar;      // смещение адреса  значения
  int offval;      // смещение значения
  int offconst;    // смещение значения инициализации
 } genvarext;      // расширение, заполняемое при генерации кода

 Entry (char pid[2*ID_MAXSIZE+1], IDDATATYPE dt, IDTYPE it, short lef, int val){ ... }
 Entry (char pid[2*ID_MAXSIZE+1], IDDATATYPE dt, IDTYPE it, short lef, char* val){ ... }
 Entry (){ ... }

};
```

10. Генерация (разработка - отладка)

```
integer function fi(int x, int y)
{
 declare integer z;
 z = x*(x+y);
 return z;
};
main
{
 declare integer x;
 rerurn x;
};

---------------------------------------------------------------
FUNC:  номер тетрады на которую надо перейти (следующая)
       Run: переход на заданную тетраду
PARM:  смещение значения параметра, смещение инициализирующей константы параметра, количество байт
       Run:  записать инициализирующую константу в значение
DTI:  смещение значения переменной, смещение инициализирующей константы переменной, количество байт
      Run:  записать инициализирующую константу в значение

MOV:  смещение значения переменной1, смещение значения переменной2, длина2
      Run:  скопировать значение переменной1  в значение перменной2
```

```
#include "Gen.h"

int _tmain(int argc, _TCHAR* argv[])
{

    int s  = 0;
    int t =  0;
    int t_x = 0,  t_x_m = 0,  t_y = 0, t_z = 0;
    LEX::LEX lex;           // лексического анализа
    lex.lextable.table[ s]  = LT::Entry('t',1);     // 0   LT::Entry( лексема , номер  исходной строки )
    lex.lextable.table[++s] = LT::Entry('i',1);     // 1
    lex.idtable.table[t] =  IT::Entry ("fi", IT::IDDATATYPE::INT,IT::IDTYPE::F, s, 0);
    lex.lextable.table[s].idxTI = t;
    lex.lextable.table[++s] = LT::Entry('f',1);     // 2
    lex.lextable.table[++s] = LT::Entry('(',1);     // 3
    lex.lextable.table[++s] = LT::Entry('t',1);     // 4
    lex.lextable.table[++s] = LT::Entry('i',1);     // 5
    lex.idtable.table[++t] =  IT::Entry ("fi#x", IT::IDDATATYPE::INT,IT::IDTYPE::P, s, 0);
    t_x = lex.lextable.table[s].idxTI = t;
    lex.lextable.table[++s] = LT::Entry(',',1);     // 6
    lex.lextable.table[++s] = LT::Entry('t',1);     // 7
    lex.lextable.table[++s] = LT::Entry('i',1);     // 8
    lex.idtable.table[++t] =  IT::Entry ("fi#y", IT::IDDATATYPE::INT,IT::IDTYPE::P, s, 0);
    t_y =lex.lextable.table[s].idxTI = t;
    lex.lextable.table[++s] = LT::Entry(')',1);     // 9
    lex.lextable.table[++s] = LT::Entry('{',2);     // 10
    lex.lextable.table[++s] = LT::Entry('d',3);     // 11
    lex.lextable.table[++s] = LT::Entry('t',3);     // 12
    lex.lextable.table[++s] = LT::Entry('i',3);     // 13
    lex.idtable.table[++t] =  IT::Entry ("fi#z", IT::IDDATATYPE::INT,IT::IDTYPE::V, s, 0);
    t_z = lex.lextable.table[s].idxTI = t;
    lex.lextable.table[++s] = LT::Entry(';',3);     // 14
    lex.lextable.table[++s] = LT::Entry('i',4);     // 15
    lex.lextable.table[s].idxTI = t_z;
    lex.lextable.table[++s] = LT::Entry('=',4);     // 16
    lex.lextable.table[++s] = LT::Entry('i',4);     // 17
    lex.lextable.table[s].idxTI = t_x;
    lex.lextable.table[++s] = LT::Entry('v',4);     // 18
```

```
integer function fi(integer x, integer y)
{
 declare  integer z;
 z= x*(x+y);
 return z;
}
```

```
lex.lextable.table[++s] = LT::Entry('(',4);      // 19
lex.lextable.table[++s] = LT::Entry('i',4);      // 20
lex.lextable.table[s].idxTI = t_x;
lex.lextable.table[++s] = LT::Entry('v',4);      // 21
lex.lextable.table[++s] = LT::Entry('i',4);      // 22
lex.lextable.table[s].idxTI = t_y;
lex.lextable.table[++s] = LT::Entry(')',4);      // 23
lex.lextable.table[++s] = LT::Entry(';',4);      // 24
lex.lextable.table[++s] = LT::Entry('r',5);      // 25
lex.lextable.table[++s] = LT::Entry('i',5);      // 26
lex.lextable.table[s].idxTI = t_z;
lex.lextable.table[++s] = LT::Entry(';',5);      // 27
lex.lextable.table[++s] = LT::Entry('}',6);      // 28
lex.lextable.table[++s] = LT::Entry(';',6);      // 29
lex.lextable.table[++s] = LT::Entry('m',7);      // 30
lex.idtable.table[++t] =  IT::Entry ("m", IT::IDDATATYPE::INT,IT::IDTYPE::M, s, 0);
lex.lextable.table[s].idxTI = t;
lex.lextable.table[++s] = LT::Entry('{',8);      // 31
lex.lextable.table[++s] = LT::Entry('d',9);      // 32
lex.lextable.table[++s] = LT::Entry('t',9);      // 33
lex.lextable.table[++s] = LT::Entry('i',9);      // 34
lex.idtable.table[++t] =  IT::Entry ("m#x", IT::IDDATATYPE::INT,IT::IDTYPE::V, s, 0);
t_x_m = lex.lextable.table[s].idxTI = t;
lex.lextable.table[++s] = LT::Entry(';',9);      // 35
lex.lextable.table[++s] = LT::Entry('r',10);     // 36
lex.lextable.table[++s] = LT::Entry('i',10);     // 37
lex.lextable.table[s].idxTI  = t_x_m;
lex.lextable.table[++s] = LT::Entry(';',10);     // 38
lex.lextable.table[++s] = LT::Entry('}',11);     // 39
lex.lextable.table[++s] = LT::Entry(';',11);     // 40
lex.lextable.table[++s] = LT::Entry('$',12);     // 41
lex.lextable.size = ++s;
lex.idtable.size = ++t;


Gen::Module  *module = Gen::generate(lex);
```

```
main
{
 declare integer x;
 declare integer y;
...
```

```
Module* generate(LEX::LEX& lex)
{
    Module* rc  = new Module();

    std::list<short> func   =  lex.idtable.getIDTYPE(IT::IDTYPE::F);    // индексы функций из ИТ
    std::list<short> mfunc  =  lex.idtable.getIDTYPE(IT::IDTYPE::M);    // индексы main из ИТ
    std::list<short> ndx_itvar;                                         // индексы ИТ-переменных параметров и лексем

    rc->refsection.functablesize   = func.size() + mfunc.size();        //  = количество функций + количесвто main-функций
    rc->refsection.mainentrynumber = func.size() + mfunc.size() - 1;    //  =  main - последняя

     Genstate genstate(rc->refsection.functablesize, rc->refsection.offconstsection);
```

Расширение ТИ (1.8)

```
struct Entry      // строка таблицы идентификаторов
{
 short      idxfirstLE;         // индекс  первой строки в таблице лексем
 char       id[2*ID_MAXSIZE+1];     // идентификатор (автоматически усекается до  ID_MAXSIZE) + префикс_функции#
 IDDATATYPE  iddatatype;        // тип данных
 IDTYPE     idtype;             // тип идентикатора
 union { ... } value;       // значение идентификатора
 struct GenFuncExt   //заполняется при генерации функции
 {
  int offrcvar;   // смещение адреса возвращаемого значения
  int offrcval;   // смещение возвращаемого значения
  int offretvar;  // смещение адреса для тетрады возврата
  int offretval;  // смещение значения тетрады возврата
 } genfuncext;    // расширение, заполняемое при генерации кода

 struct GenVarExt  //заполняется при генерации переменных
 {
  int offvar;      // смещение адреса  значения
  int offval;      // смещение значения
  int offconst;    // смещение значения инициализации
 } genvarext;      // расширение, заполняемое при генерации кода

 Entry (char pid[2*ID_MAXSIZE+1], IDDATATYPE dt, IDTYPE it, short lef, int val){ ... }
 Entry (char pid[2*ID_MAXSIZE+1], IDDATATYPE dt, IDTYPE it, short lef, char* val){ ... }
 Entry (){ ... }

};
```

```
  func.push_back(mfunc.back());                          // добавить в список функций  функцию main
  short len_ret = 0, len_var = 0, j, r;
  for (int k = 0; k< func.size(); k++)                   //   цикл по функциям
  {
       j = *std::next(func.begin(),k);                  // индекс  функции  в ТИ
       len_ret =  IT::IdTable::getlendatatype(lex.idtable.table[j].iddatatype);   // длина  значения  типа   функции

       rc->datasections[k].valsection.nt_ret = GEN_INIT_INT;                    // отладка: инициализация памяти
       memset(rc->datasections[k].valsection.mem, GEN_INIT_BYTE, len_ret);       // отладка: инициализация памяти

       lex.idtable.table[j].set_genfuncext(k,
                                    (int)&rc->datasections[k].varsection.offtret     - (int)rc,
                                    (int)&rc->datasections[k].valsection.nt_ret      - (int)rc,
                                    (int)&rc->datasections[k].varsection.offrc       - (int)rc,
                                    (int)&rc->datasections[k].valsection.mem         - (int)rc);

       genstate.set_initvar(k, (int)&rc->datasections[k].varsection.offmem - (int)rc,
                            (int)&rc->datasections[k].valsection.mem + len_ret - (int)rc);


       ndx_itvar =  lex.idtable.getvarlit(lex.idtable.table[j].id);   // список ИТ-индексов  переменных id-функции
       for(int l = 0; l < ndx_itvar.size(); l++)                      // цикл по переменным и литералам
       {
           r = *std::next(ndx_itvar.begin(), l);                       // индекс переменной  в ТИ
           len_var =  IT::IdTable::getlendatatype(lex.idtable.table[r].iddatatype);   // длина значения типа

           rc->zapvar(genstate.funcdatasections[k].next_varsection,
                      genstate.funcdatasections[k].next_valsection,
                      genstate.next_constsection,  len_var,  (void*)&lex.idtable.table[l].value);

           lex.idtable.table[r].set_genvarext(   //  записать смещения в ИТ  переменной
                                        genstate.funcdatasections[k].next_varsection,
                                        genstate.funcdatasections[k].next_valsection,
                                        genstate.next_constsection
                                       );
           genstate.set_nextvar(k, len_var);     // сдвиг смещений после добавления переменной
       };
  };
```

```
// потом цикл по правилам вывода

//----------------------- S->tif(F){NR;};S   ---------------------------------------------------- -------------------
IT::Entry ite1 = lex.idtable.table[lex.lextable.table[0+1].idxTI];                          // нашли вход в ТИ
rc->refsection.functable[ite1.genfuncext.nfuncsection].tfirst = genstate.next_tetrad;       // номер 1ой тетр -> секция ссыл
Gen::Tetrad tetr1(TETRADETYPE::FUNC, genstate.next_tetrad+1);                               // создали тетраду FUNC
genstate.next_tetrad = rc->zaptetrad(tetr1);                                                // записали тетраду

//----------------------- F->ti,F ------------------------------------------------------------------ ----------------
IT::Entry ite2 = lex.idtable.table[lex.lextable.table[4+1].idxTI];                         // нашли вход в ТИ
int len2 =  IT::IdTable::getlendatatype(ite2.iddatatype);                                   // определили длину типа
Gen::Tetrad tetr2(TETRADETYPE::PARM,ite2.genvarext.offval, ite2.genvarext.offconst, len2);  // создали тетраду PARM
genstate.next_tetrad  = rc->zaptetrad(tetr2);                                               // записали тетраду

//---------------------- F->ti---------------------------------------------------------------------- -----------------
IT::Entry ite3 = lex.idtable.table[lex.lextable.table[7+1].idxTI];                          // нашли вход в ТИ
int len3 =  IT::IdTable::getlendatatype(ite3.iddatatype);                                   // определили длину типа
Gen::Tetrad tetr3(TETRADETYPE::PARM,ite3.genvarext.offval, ite3.genvarext.offconst, len3);  // создали тетраду PARM
genstate.next_tetrad  = rc->zaptetrad(tetr3);                                               // записали тетраду

//---------------------- N->dti,N------------------------------------------------------------------- --------------
IT::Entry ite4 = lex.idtable.table[lex.lextable.table[11+2].idxTI];                         // нашли вход в ТИ
int len4 =  IT::IdTable::getlendatatype(ite4.iddatatype);                                   // определили длину типа
Gen::Tetrad tetr4(TETRADETYPE::DTI,ite4.genvarext.offval, ite4.genvarext.offconst, len4);   // создали тетраду PARM
genstate.next_tetrad  = rc->zaptetrad(tetr4);                                               // записали тетраду

//---------------------- N->i=E;-------------------------------------------------------------------- -------------
// - формирование польской записи
// - создание аккумуляторной переменной
// - генерация тетрад вычисления выражения (PUSH, POP, SUM, MULT )
// - результат выражения в  аккумуляторе, адрес смещения аккумулятора в стеке
IT::Entry ite5 = lex.idtable.table[lex.lextable.table[15+0].idxTI];                         // нашли вход в ТИ
```