

## Генерация кода в ассемблер

### 1. Объединение процедур, расположенных в разных модулях

Процедура — именованная, правильным образом оформленная группа команд, которая объявляется один раз и может многократно вызываться по имени в любом месте программы.

#### MASM:

EXTRN <имя> — объявление внешнего имени по отношению к данному модулю.  
PROC и ENDP — начало, конец процедуры.  
CALL <ИМЯПРОЦЕДУРЫ> — команда вызова процедуры.  
RET <ЧИСЛО> — команда возврата управления вызывающей программе,  
где <число> — количество байт, удаляемых из стека при возврате из процедуры (необязательный параметр).

### Объединение процедур, расположенных в разных модулях

Каждый модуль должен сообщать транслятору, что некоторый объект (процедура, переменная) должен быть видимым вне этого модуля.

Транслятор также должен знать, что некоторые объекты определены вне данного модуля.

Все внешние ссылки в объединяемых модулях разрешаются на этапе **компоновки**.

### Организация интерфейса с процедурой

Для передачи аргументов в языке ассемблера существуют следующие способы:

- через регистры;
- через общую область памяти;
- через стек;
- с помощью директив **extern** и **public**.

### Передача аргументов через стек

Вызывающая процедура заносит в стек параметры и передает управление вызываемой процедуре. При передаче управления процедуре в вершину стека

поверх параметров автоматически записывается 4 байта с адресом возврата в вызывающую программу.

Стек обслуживается тремя регистрами:

- ESI – указатель дна стека (начала сегмента стека);
- ESP – указатель вершины стека;
- EBP – указатель базы.

Регистры ESI и ESP указывают на дно и вершину стека соответственно.

Регистр EBP используется для произвольного доступа к данным в стеке.

<b>Пролог процедуры</b> (настройка регистра EBP в процедуре )	<code>push ebp</code> // указатель базы сохраняется в стеке <code>mov ebp, esp</code> // настраивает EBP на вершину стека
<b>Эпилог процедуры</b>	Восстановление    // выполняет действия, обеспечивающие стека                // корректный возврат из процедуры

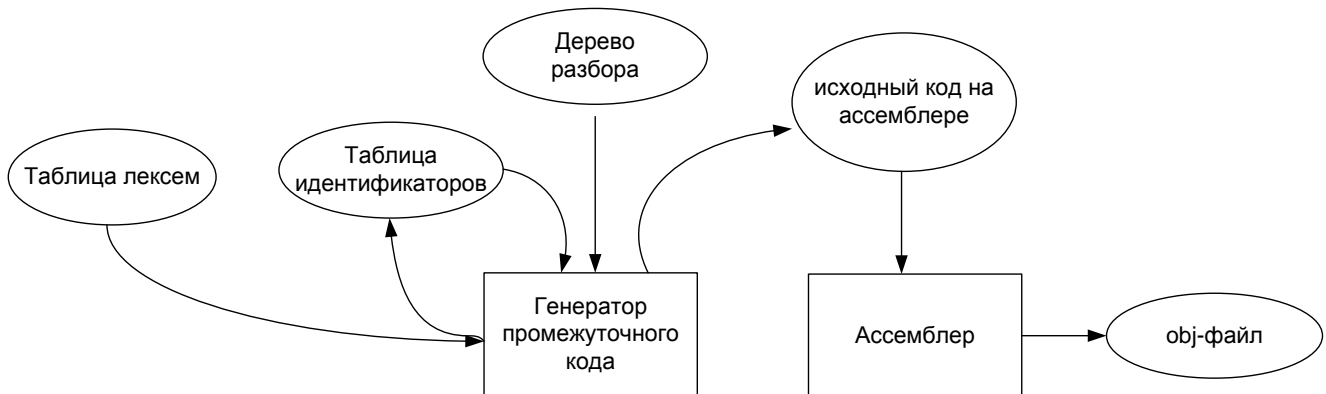
**Восстановление стека:**

<i>cdecl</i>	<code>add esp, &lt;длина_параметров_в_байтах&gt;</code>	// в вызывающем коде
<i>stdcall</i>	<code>ret &lt;длина_параметров_в_байтах&gt;</code>	// в вызываемом коде

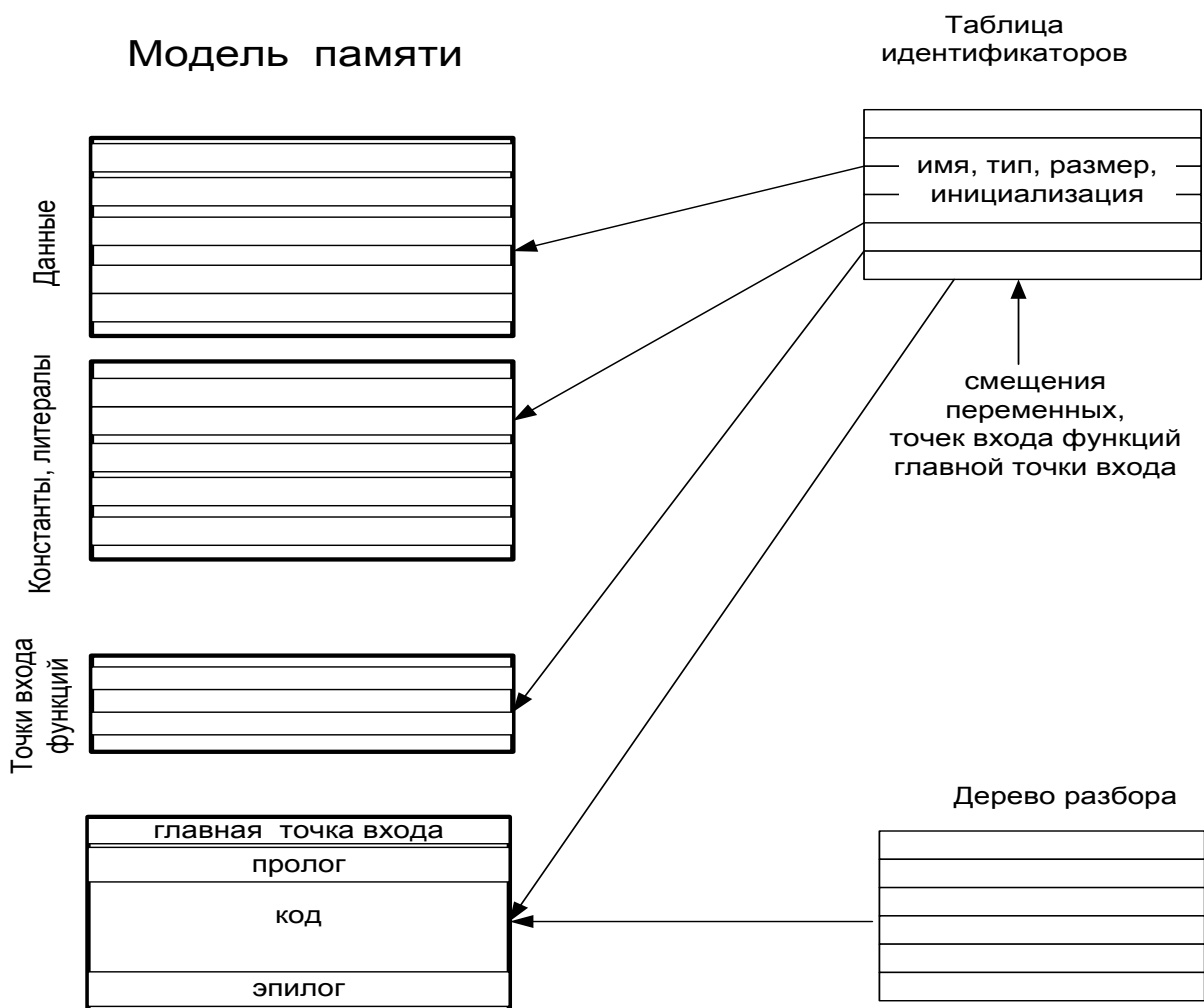
В программах, написанных на языке ассемблера, используется соглашение о вызовах `stdcall`.

## 2. Генерация исходного ассемблерного кода

### 2.1 Подход к генерации кода по дереву разбора с использованием стека.



### 2.2 Модель памяти SVV-2015



## 2.3 Дерево разбора

Дерево разбора:

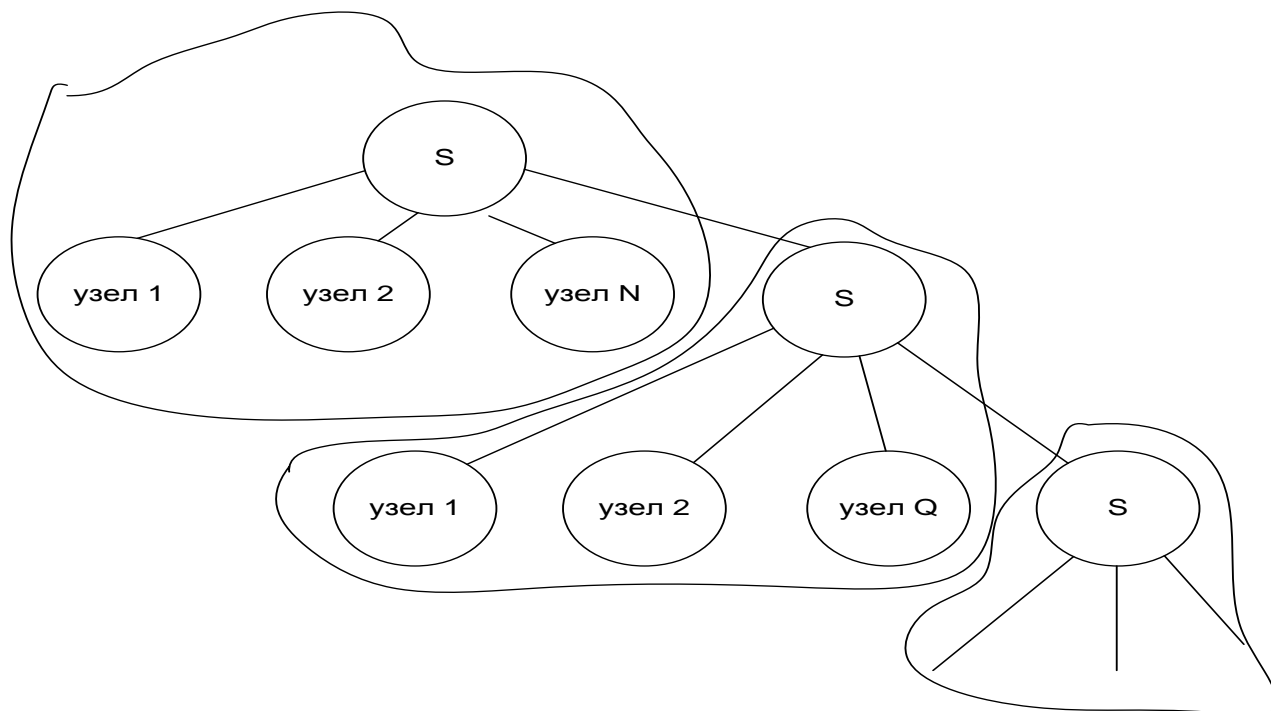
$S$

– стартовый символ.

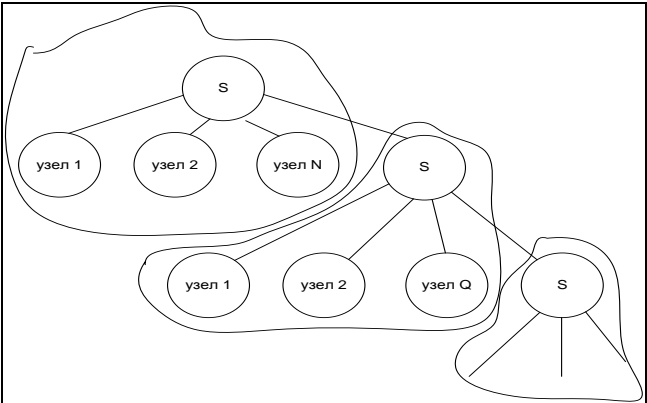
узел 1, узел 2, ..., узел N

– каждый узел описывает функцию.

Каждой функции соответствует блок кода.



2.4 Модель памяти SVV-2015: точки входа функций и код



Точки входа функций

команда перехода в главную точку входа
команда перехода в блок кода 1
команда перехода в блок кода 2
команда перехода в блок кода 3
команда перехода в блок кода N

Код

главная точка входа
Пролог: выделение памяти, настройка адресов, согласование с ОС, старт главной функции
Блок кода 1 (главная функция)
Блок кода 2 (функция)
Блок кода 3 (функция)
Блок кода N (функция)
Эпилог: освобождение памяти, согласование с ОС

Дерево разбора

Поддерево 1
Поддерево 2
Поддерево 3
Поддерево N

## 2.5 Простой вариант генерации кода на ассемблере

Скелет главной функции:

```
.586                                ; система команд (процессор Pentium)
.model flat,stdcall                 ; модель памяти, соглашение о вызовах
includelib kernel32.lib            ; компоновщику: компоновать с kernel32.lib
                                   ; можем компоновать со стандартной библиотекой

ExitProcess PROTO :DWORD           ; прототип функции

.stack 4096                         ; сегмент стека объемом 4096 - для вычислений

.data                              ; сегмент данных - переменные и параметры

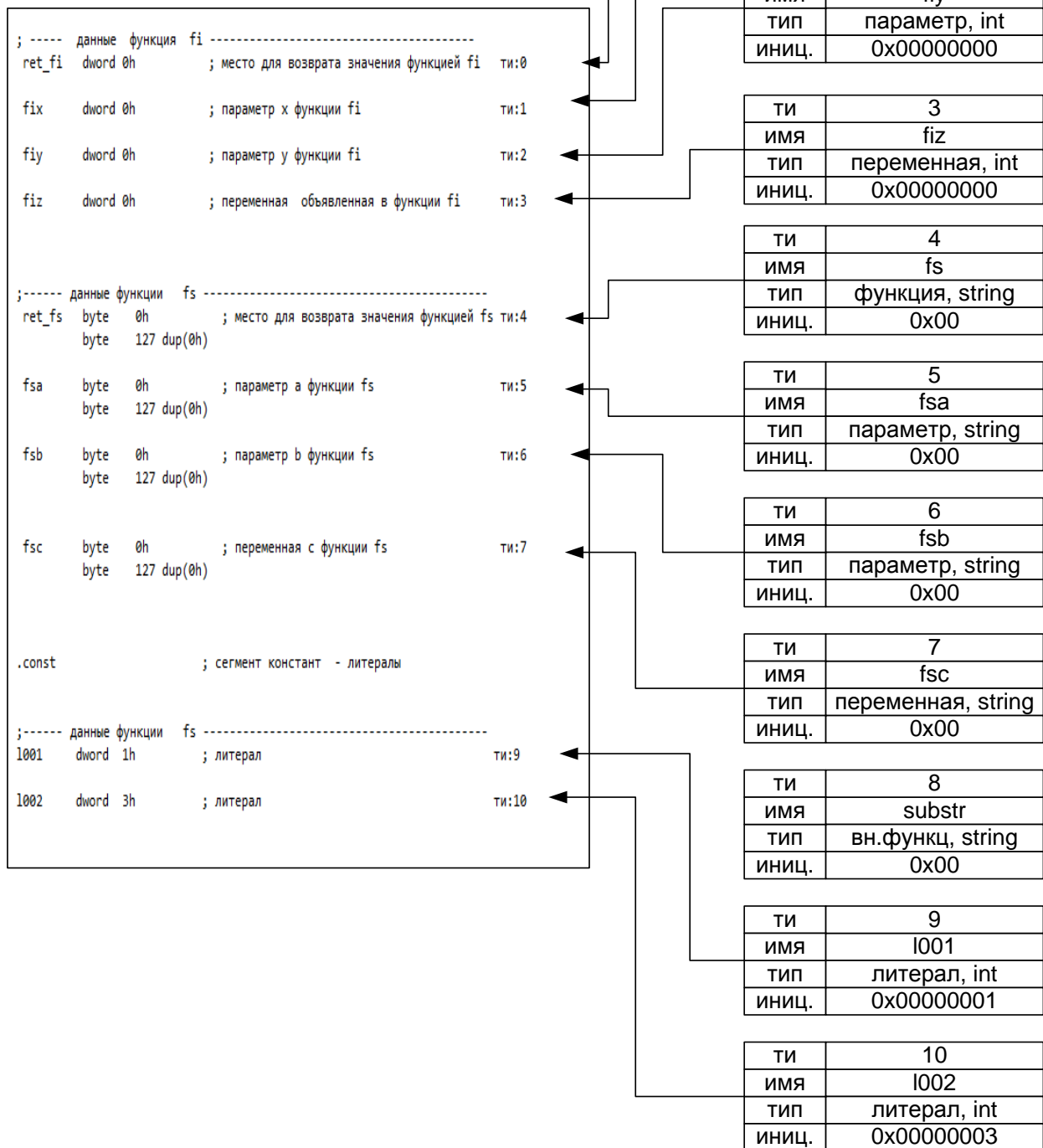
.const                             ; сегмент констант - литералы

.code                              ; сегмент кода - исполняемый код

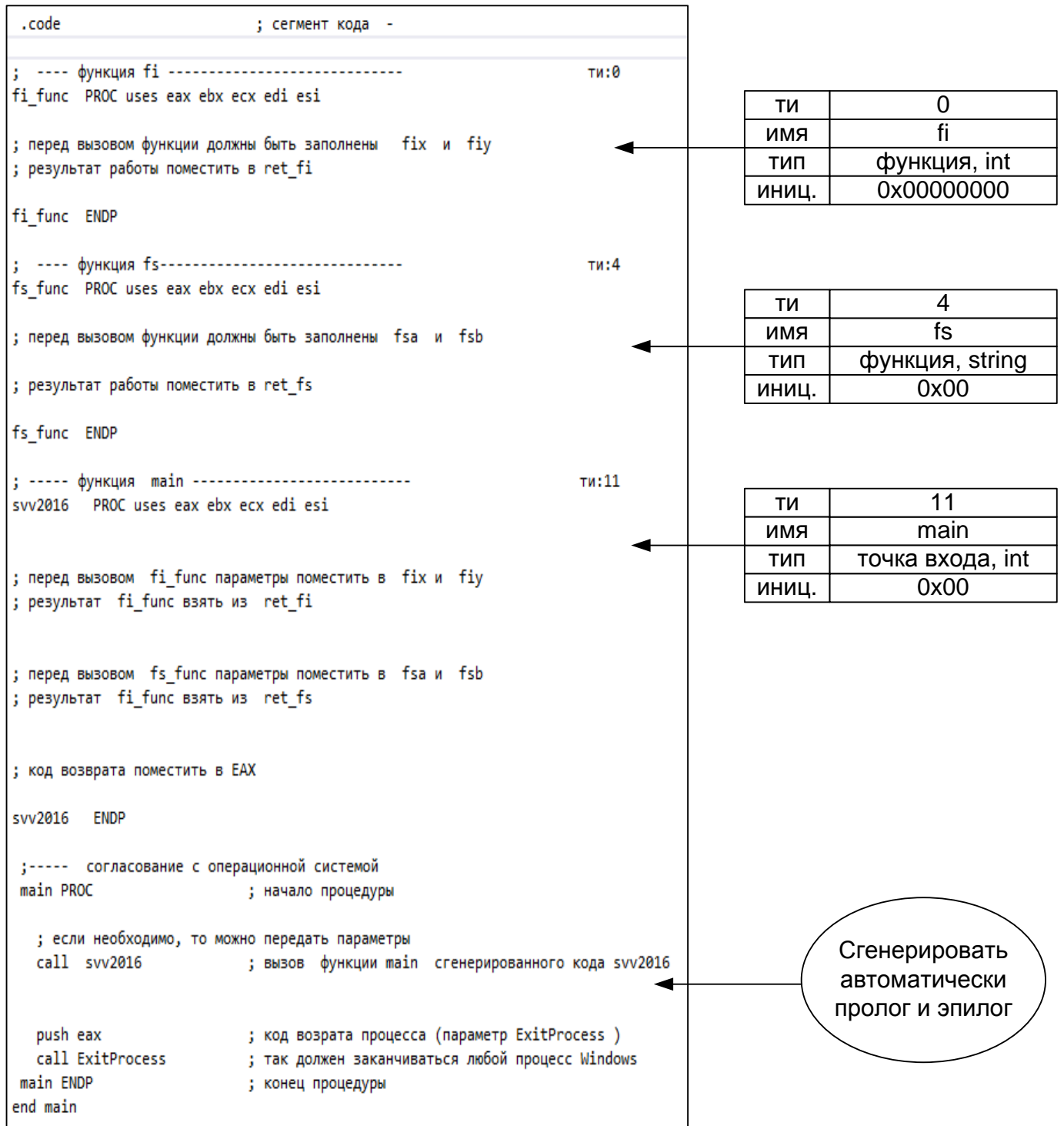
main PROC                          ; начало процедуры - согласование с ОС

    push 0                         ; код возврата процесса (параметр ExitProcess )
    call ExitProcess               ; так должен заканчиваться любой процесс Windows
main ENDP                           ; конец процедуры
end main
```

## 2.6 Модель памяти SVV-2015: данные



## 2.7 Заготовки-шаблоны для функций



## 2.8 Для внешних функций можно сгенерировать функцию-обертку для согласования стандарта вызова:

- функцию написать на C++ и поместить ее библиотеку;
- вызов функции выполнить из функции обертки;
- в генерируемом коде вызывать функцию обертку.



### 3. Генерация кода для выражений

```
.data                                ; сегмент данных - переменные и параметры
svv2016x    sdword oh                ; переменная x со знаком
svv2016y    sdword oh                ; переменная y со знаком
svv2016z    sdword oh                ; переменная z со знаком
svv2016a    sdword oh                ; переменная a со знаком
svv2016b    sdword oh                ; переменная b со знаком

.const                                           ; сегмент констант - литералы
1001    sdword 1h                        ; литерал
1002    sdword 3h                        ; литерал
1003    sdword 4h                        ; литерал

.code                                           ; сегмент кода - исполняемый код
```

```
;-----
; x = 1;
; y = x;
; z = x*y;
; z = z + x*y;
; a = 3;
; b = 7;
; z = x*y + b*(x+y);
```

```
; генерация кода: x = 1 --> svv2016x = 1001

; генерация кода: 1001
push 1001

; генерация кода: x=
pop svv2016x
```

```
; генерация кода: y = x --> svv2016y = svv2016x

; генерация кода: svv2016x
push svv2016x

; генерация кода: y=
pop svv2016y
```

```

; генерация кода: z = x*y  --> svv2016z = svv2016x*svv2016y -->svv2016z = svv2016x svv2016y *

; генерация кода: svv2016x
push svv2016x

; генерация кода: svv2016y
push svv2016y

; генерация кода: *
pop eax
pop ebx
imul ebx ; eax = eax*ebx
push eax

; генерация кода: z =
pop svv2016z

```

```

; генерация кода: z = z+ x*y  --> svv2016z = svv2016z+ svv2016x*svv2016y -->svv2016z = svv2016z svv2016x svv2016y *+

; генерация кода: svv2016z
push svv2016z

; генерация кода: svv2016x
push svv2016x

; генерация кода: svv2016y
push svv2016y

; генерация кода: *
pop eax
pop ebx
imul ebx ; eax = eax*ebx
push eax

; генерация кода: +
pop eax
pop ebx
add eax, ebx ; eax = eax+ebx
push eax

; генерация кода: z =
pop svv2016z

```

```

; a = 3 --> a = 1002
; b = 7 ---> b = 1003
; z = x*y + b*(x+y) --> svv2016z = svv2016x*svv2016y + svv2016b * (svv2016x + svv2016y)
;                               svv2016z = svv2016x svv2016y * svv2016b svv2016x svv2016y ++
; генерация кода: a = 3 --> svv2016a = 1002

; генерация кода: 1002
    push 1002
; генерация кода: a =
    pop svv2016a
; генерация кода: 1003
    push 1003
; генерация кода: b =
    pop svv2016b
; генерация кода: svv2016x
    push svv2016x
; генерация кода: svv2016y
    push svv2016y
; генерация кода: *
    pop    eax
    pop    ebx
    imul   ebx        ; eax = eax*ebx
    push   eax
; генерация кода: svv2016b
    push svv2016b
; генерация кода: svv2016x
    push svv2016x
; генерация кода: svv2016y
    push svv2016y
; генерация кода: +
    pop    eax
    pop    ebx
    add    eax, ebx    ; eax = eax+ebx
    push   eax
; генерация кода: *
    pop    eax
    pop    ebx
    imul   ebx        ; eax = eax*ebx
    push   eax
; генерация кода: +
    pop    eax
    pop    ebx
    add    eax, ebx    ; eax = eax+ebx
    push   eax
; генерация кода: svv2016z =
    pop svv2016z

```

### 3.1 Готовые шаблоны:

```
//шаблоны
#define EXPR_INT      "push    %s \n"           // i
#define EXPR_INT_E    "pop     %s \n"           // =
#define EXPR_INT_PLUS "pop     eax\npop     ebx\nadd    eax, ebx\npush  eax\n"           // +
#define EXPR_INT_MUL  "pop     eax\npop     ebx\nimul   eax, ebx\npush  eax\n"           // *
#define GEN1(b, tmpl, var) sprintf_s(b, 1024,tmpl, #var)
#define GEN0(b, tmpl)   sprintf_s(b, 1024,tmpl)
```

#### Пояснения:

- *<имя>\_s* – это безопасные функции с указанием емкости приемника, например:  
sprintf\_s – возвращает количество байт, записанных в буфер или -1
- *Буфер* – место, где сохраняется генерируемый код.
- *Стрингификация* – операция # (преобразование фрагмента кода в строку)

```
char buf[1024];

// x =1
int k = GEN1(buf,EXPR_INT,      1001);           // 1
      k+= GEN1(buf+k,EXPR_INT_E,  svv2016x);      // =
```

```
// y = x
      k+= GEN1(buf+k,EXPR_INT,      svv2016x);    // x
      k+= GEN1(buf+k,EXPR_INT_E,    svv2016y);    // =
```

```
// z = xy*
      k+= GEN1(buf+k,EXPR_INT,      svv2016x);    // x
      k+= GEN1(buf+k,EXPR_INT,      svv2016y);    // y
      k+= GEN0(buf+k,EXPR_INT_MUL);  // *
      k+= GEN1(buf+k,EXPR_INT_E,    svv2016z);    // =
```

```
// z = zxy*+
      k+= GEN1(buf+k,EXPR_INT,      svv2016z);    // z
      k+= GEN1(buf+k,EXPR_INT,      svv2016x);    // x
      k+= GEN1(buf+k,EXPR_INT,      svv2016y);    // y
      k+= GEN0(buf+k,EXPR_INT_MUL);  // *
      k+= GEN0(buf+k,EXPR_INT_PLUS); // +
      k+= GEN1(buf+k,EXPR_INT_E,    svv2016z);    // =
```

```
// a = 3;
k+= GEN1(buf+k,EXPR_INT,      1002);           // 3
k+= GEN1(buf+k,EXPR_INT_E,    svv2016a);       // =
```

```
// b = 7;
k+= GEN1(buf+k,EXPR_INT,      1003);           // 7
k+= GEN1(buf+k,EXPR_INT_E,    svv2016b);       // =
```

```
// z = xy* bxy*++;
k+= GEN1(buf+k,EXPR_INT,      svv2016x);       // x
k+= GEN1(buf+k,EXPR_INT,      svv2016y);       // y
k+= GEN0(buf+k,EXPR_INT_MUL);                  // *
k+= GEN1(buf+k,EXPR_INT,      svv2016b);       // b
k+= GEN1(buf+k,EXPR_INT,      svv2016x);       // x
k+= GEN1(buf+k,EXPR_INT,      svv2016y);       // y
k+= GEN0(buf+k,EXPR_INT_PLUS);                 // +
k+= GEN0(buf+k,EXPR_INT_MUL);                  // *
k+= GEN0(buf+k,EXPR_INT_PLUS);                 // +
k+= GEN1(buf+k,EXPR_INT_E,      svv2016z);     // =
```

```
std::cout << buf;
```

### 3.2 Сгенерированный код

```
push    1001
pop     svv2016x
push    svv2016x
pop     svv2016y
push    svv2016x
push    svv2016y
pop     eax
pop     ebx
imul    eax, ebx
push    eax
pop     svv2016z
push    svv2016z
push    svv2016x
push    svv2016y
pop     eax
pop     ebx
imul    eax, ebx
push    eax
pop     eax
pop     ebx
add     eax, ebx
push    eax
pop     svv2016z
push    1002
pop     svv2016a
push    1003
pop     svv2016b
push    svv2016x
push    svv2016y
pop     eax
pop     ebx
imul    eax, ebx
push    eax
push    svv2016b
push    svv2016x
push    svv2016y
pop     eax
pop     ebx
add     eax, ebx
push    eax
pop     eax
pop     ebx
imul    eax, ebx
push    eax
pop     eax
pop     ebx
add     eax, ebx
push    eax
pop     svv2016z
```

Последовательность генерации:

- 1) План памяти строится по ТИ
- 2) Выполняется преобразование выражений в ПОЛИЗ
- 3) Заготавливаются шаблоны кода соответствующего правила грамматики
- 4) Код генерируется по дереву разбора