Функции. Модульная структура программы

Модульная структура Си-программы (классический Си ,не ООП):

- Функция – алгоритмическая единица
- Файл – физическая единица, набор функций + общие данные
- Проект – список файлов «исходников», объектных модулей и библиотек, собираемых в исполняемый модуль (exe-файл)
- Решение (Solution) – несколько проектов над общим кодом

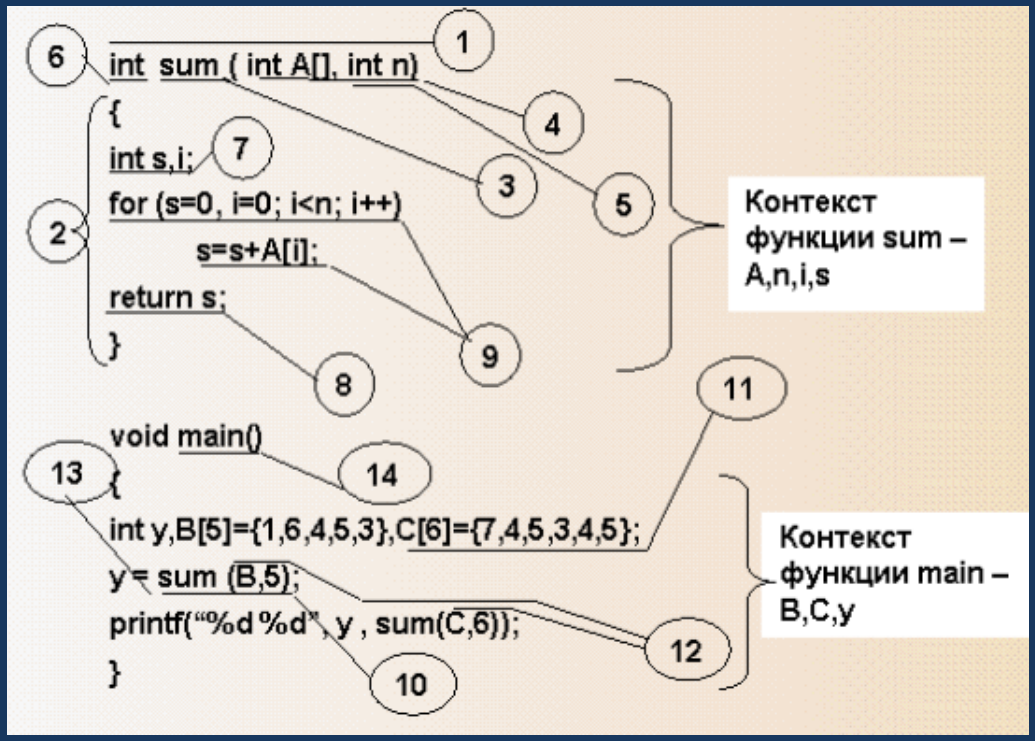
Объектно-ориентированное программирование (ООП):

- Метод – функция в классе
- Класс – набор методов над общими данными – свойствами объекта (обычно 1 класс = 1 файл, вложенные классы)
- Проект – набор классов (Пакет (Java) – адресация кода классов и группирование на основе древовидной структуры имен)

Объектный модуль – «полуфабрикат» компиляции, внутреннее двоичное представление кода + интерфейс подключения
Библиотека – архив объектных модулей

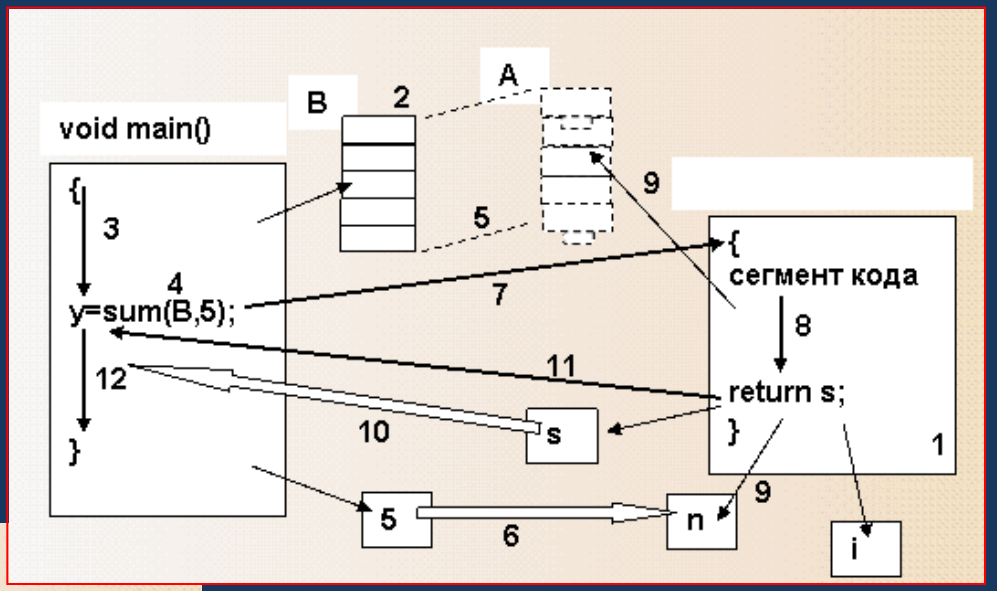
Синтаксис функции

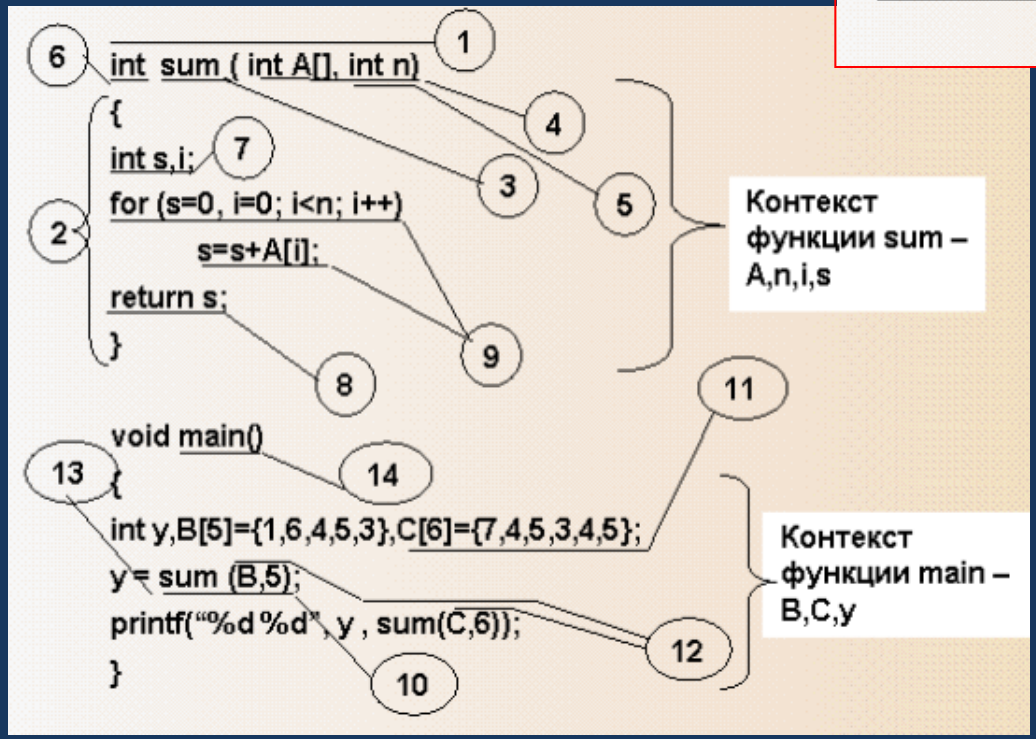
1. Функция = заголовок + тело
2. Тело = блок {…}
3. Имя функции
4. Список формальных параметров (ФП)
5. ФП - синтаксис **определения переменной,** для каждого – тип данных (ТД) слева
6. Тип результата, результат – значение, возвращаемое функцией в выражение в точку вызова (13)
7. Локальные переменные (ЛП)
8. return - оператор возвращает значение (копия на выход)
9. Контекст функции = ФП + ЛП
10. Вызов функции – имя + список фактических параметров (ФкП)
11. ФкП - имя или значение объекта, видимого в контексте вызова **(синтаксис выражения)**



Вызов функции

1. Функция = сегмент кода
2. ФП – массив в контексте main
3. Исполнение кода main
4. ФкП ставятся в соответствие ФП
5. Массивы «отображаются»
6. Базовые ТД «копируются»
7. Вызов – сохранение «точки возврата» и переход к телу
8. Исполнение кода тела





9. Использование ФП и ЛП
10. return - оператор возвращает значение (копия на выход)
11. Переход в точку возврата
12. Результат используется в точке вызова

Способы передачи ФП и результата

Передача параметра по значению - копирование значения фактического параметра в формальный, неявное присваивание:

- ФП являются собственными переменными функции, аналогом локальных переменных
- при изменении ФП значения соответствующих им фактических параметров не меняются
- передача параметров по значению используется для задании функции входных данных

Передача параметра по ссылке - отображение формального параметра в фактический:

- ФП существуют как синонимы фактических, они не являются «настоящими» переменными, под которые отводится память
- при изменении значений ФП эти изменения проецируются на соответствующие им фактические
- передача параметров по ссылке может использоваться для данных, которые являются либо чисто выходными (неявный результат функции), либо представляют собой внешние для функции данные, которые она должна изменить
- реализация - неявно передается **адрес** фактического параметра

Результат передается по значению

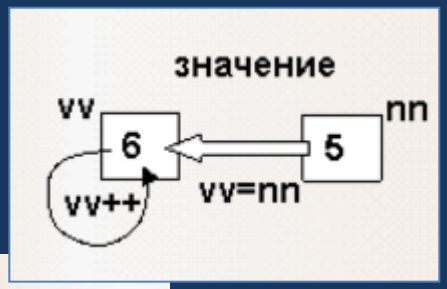
Исторические предпосылки синтаксиса

- Си – архитектурно-ориентированный язык («что пишем – то делается», «чистый» код, производительность, управление памятью (адресная арифметика), операции = типовая система команд, типы данных – аппаратно-представимые)
- Память процессора:
  - регистры – машинные слова, непосредственно адресуемые в командах (MOV R0,R2)
  - память (виртуальное АП), адресуемое прямо в командах или косвенно через регистры (способы адресации операндов)
  - кэш-память процессора – «прозрачная» быстродействующая память ограниченного объема в чипе процессора для подкачки по требованию блоков памяти (**не видна на программном уровне**)

Вывод: для передачи данных объемом более 1-2 слов их необходимо размещать в памяти и передавать **адрес**

Способы передачи ФП и результата

- **Передача параметра по значению -** все, кроме массивов (**неявно**)
- **Передача массива по ссылке – неявно**
- **Передача параметра по ссылке явно –** int &a в списке ФП
- **Явное использование указателей**
- **Результат – по значению** все, кроме массивов
- **Результат – по ссылке явно**



Процедура

```
void Hello(){ printf("Hello, Im here"); }
void main(){ Hello(); }
```

Входы и выход - значения

```
int nod(int a,int b){
        for (int n=a; !(n%a==0 && n%b==0);n++);
        return n; }
void main(){ int y=nod(36,54); }
```

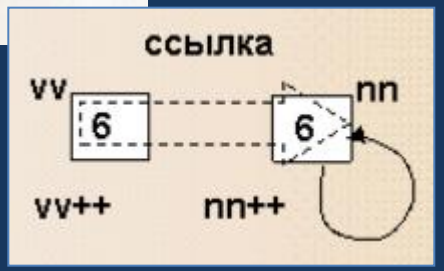
Ссылки на изменяемые параметры

```
void inc(int &n){ n++; }
void main(){ int m=6; inc(m); }
```

```
void swap(int &a, int &b){
        int c=a; a=b;b=c; }
void main(){ int m1=5,m2=7; swap(m1,m2); }
```

Массив по ссылке

```
void swap(int A[],int n){
        for (int i=0,j=n-1;i<j;i++,j--)
                { int c=A[i];A[i]=A[j];A[j]=c; }
        }
void main(){
        int B[]={3,6,4,5,7};
        swap(B,5); }
```



Результат по ссылке

```
void max(int &s,int A[], int n){
        int i;
        for (i=1,s=A[0];i<n;i++)
                if (A[i]>s) s=A[i];
         }
void main(){
        int m,B[]={3,6,4,5,7};
        max(m,B,5); }
```

Способы передачи ФП и результата

Два результата

```
int len(int &k,int A[],int n){
        for (int i=0;i<n;i++){
                for (k=1;i+k<n && A[i]==A[i+k];k++);
                if (k>1) return i;
                }
        return -1;}
void main(){
        int k1,m1,B[10]={5,7,6,6,6,6,6,6,5,7};
        k1=len(m1,B,10); }
```

```
int &F38(int &n1, int &n2){
return n1 > n2 ? n1 : n2; }
void main8(){ int x=5,y=6,z; z=F38(x,y); F38(x,y)++; }
```

```
int &F39(int &n1, int &n2){
return n1 > n2 ? n1 : n2; }
void main10(){ int x=5,y=6,z; F39(x,y)=0; F39(x,y)++; }
```

Результат – ссылка на мин. элемент

```
int &ref_min(int A[], int n){
for (int i=0,k=0; i<n; i++)
        if (A[i]<A[k]) k=i;
return A[k];}
void main(){
int B[5]={4,8,2,6,4};
ref_min(B,5)++;
for (int i=0; i<5; i++) printf("%d ",B[i]); }
```

Способы передачи ФП и результата

Возвратить массив нельзя, но можно:

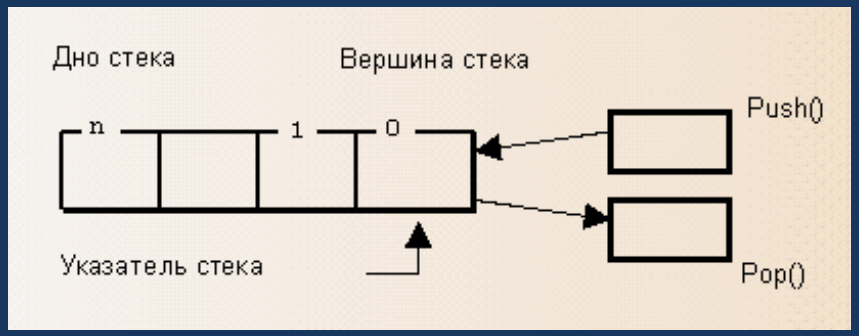
- Передать пустой массив и заполнить в функции
- Возвратить указатель на динамический массив (cprog 5.6)
- Поместить массив в struct (объект) (cprog 5.3)
  - struct можно передавать и возвращать по ссылке и по значению
  - При передаче по значению в стеке – копия
  - При возврате по значению (return) в main создается временная переменная, а функция получает на нее ссылку (адрес)

В любом случае за соответствием между размерностью массива и количеством занятых элементов должна следить программа

Стек

Стек - последовательность, включение и исключение элементов в которую производится только с одного конца:

- Последовательность записи и извлечения взаимно обратны
  PUSH (A), PUSH(B), PUSH(C), POP()->C, POP()->B, POP()->A
- Моделирует принцип вложенности:
  - Вложенность конструкций при синтаксическом анализе – стековый автомат
  - Вложенный вызов функций main – A – B – return – A – return – main, стек – история вызовов в обратном порядке
  - Вложенные прерывания процессора (Interrupt)
- Представление – область памяти (массив) с указателем (индексом) элемента в вершине стека (последний записанный). **SP – Stack Pointer**
- В аппаратном сегменте стека (SS) – стек «кверху дном»




```
int a[100], int sp=-1;

void PUSH(int v){ a[++sp]=v; }

int POP(){ return sp == -1 ? 0 : a[sp--]; }

int GET(int n) { return sp == -1 ? 0 : a[sp-n]; }
```

Стек и вызов функции

В стеке – **текущий контекст вызова функции** (фрейм/кадр вызова):

- В «main» в стеке резервируется место для формальных параметров, в которые записываются значения фактических параметров
- при вызове функции в стек записывается точка возврата - адрес той части программы, где находится вызов функции (CS:IP)
- в начале тела функции в стеке резервируется место для локальных переменных

```
void main(){
   int d,k=7;
   d = F(k,12);
   }
```

SP →

```
12_b
8__a
4__CS:IP
0__c
```

```
int F(int a,int b){
   int c;
   c = a + b + 5;
   return c;
}
```

```
PUSH #12
PUSH K[sp]      c=a
CALL F
POP             удалить
POP             удалить
MOV ax,D[sp]    результат
```

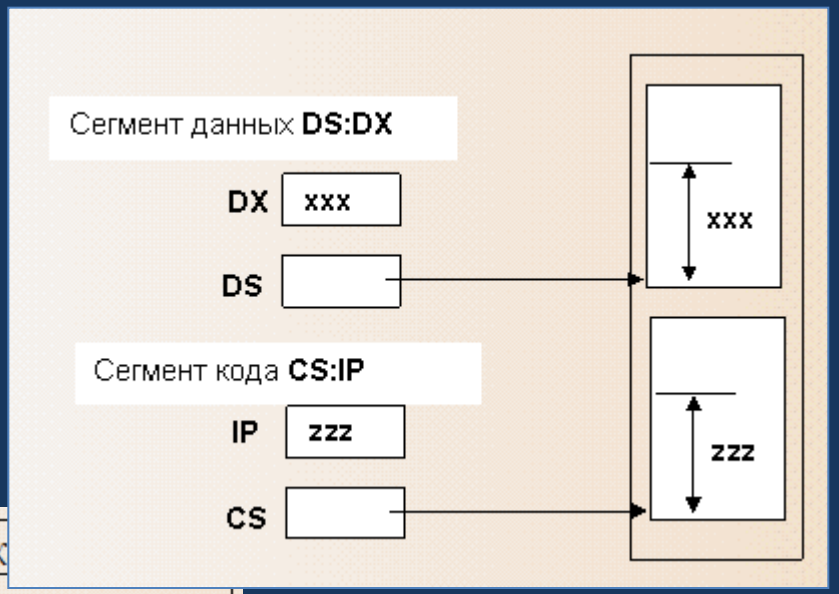
Формальные параметры – «ожидаемая» структура стека

Фактические параметры – реальная структура стека

```
F:PUSH #0       int  c
MOV 8[sp],ax    c=a
ADD 12[sp],ax   c+=b
ADD  #5,ax      c+=5
POP bx          удалить c
RET
```

Виды (сегменты) памяти

Сегмент – часть адресного пространства с относительной внутренней адресацией (смещение,offset), начальным адресом (база), системой доступа (R,RW) . Используется для хранения однотипных компонент программы



Сегмент	Регистры	Что содержит	К
Сегмент команд	CS- сегментный, IP- адрес команды	Программный код (операции, операторы)	Трансляция
Сегмент данных	DS – сегментный	Глобальные (статические) данные	Трансляция
Сегмент стека	SS – сегментный, SP- указатель стека	Локальные данные функций, «история» работы программы	При загрузке
Динамическая память	DS – сегментный	Динамические переменные, создаваемые при работе программы	При загрузке, выполнении
Динамически связываемые библиотеки (DLL)	CS- сегментный, IP- адрес команды	Программный код разделяемых библиотек	При загрузке

Области видимости и время жизни

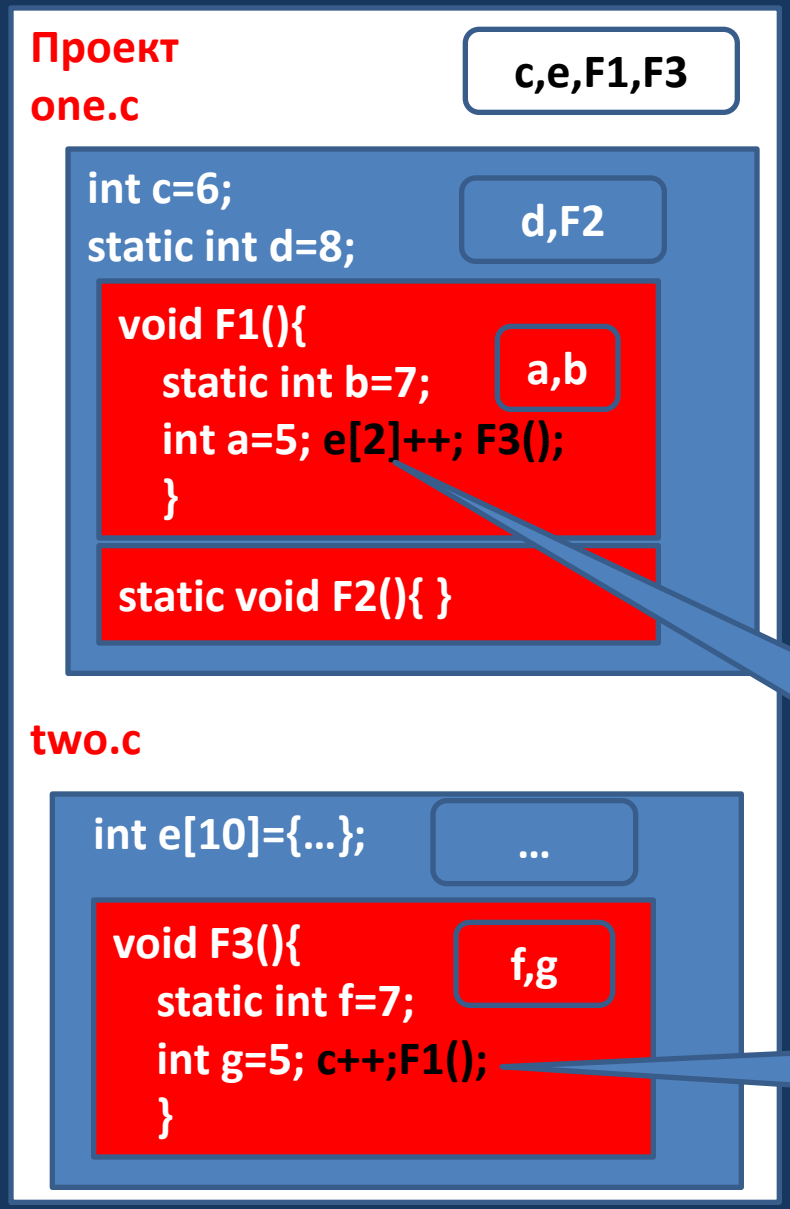
Область видимости – часть исходного текста, в котором объект программы (функция, переменная) могут быть использованы («видимы», доступны):

- Текущая функция (внутри {…})
- Текущий файл
- Весь проект

Время жизни – часть времени исполнения программы, в течение которого объект (переменная) существует:

- Все время загрузки программы (сегмент DS в исполняемом модуле)
- Время исполнения функции – «от { до }» (сегмент стека SS)
- Создается и уничтожается программой – динамические переменные (cprog 5.6) (сегмент «кучи»), *формально не является частью языка*

Виды переменных (классы памяти)



DS: **b,c,d,f**

SS: **a,g** в разное время

CS: **F1,F2,F3**

Определение – описание объекта программы (переменной, функции, класса), транслятор создает его внутреннее представление в памяти

Объявление – сообщение транслятору об имени и свойствах недоступного объекта

```
extern int e[];
extern void F3(void);
```

```
extern int c;
extern void F1(void);
```

Видимость переменных и функций

Определение – текстовое описание объекта программы (переменной, функции, класса), по которому транслятор создает его внутреннее (двоичное) представление (размещает в памяти)

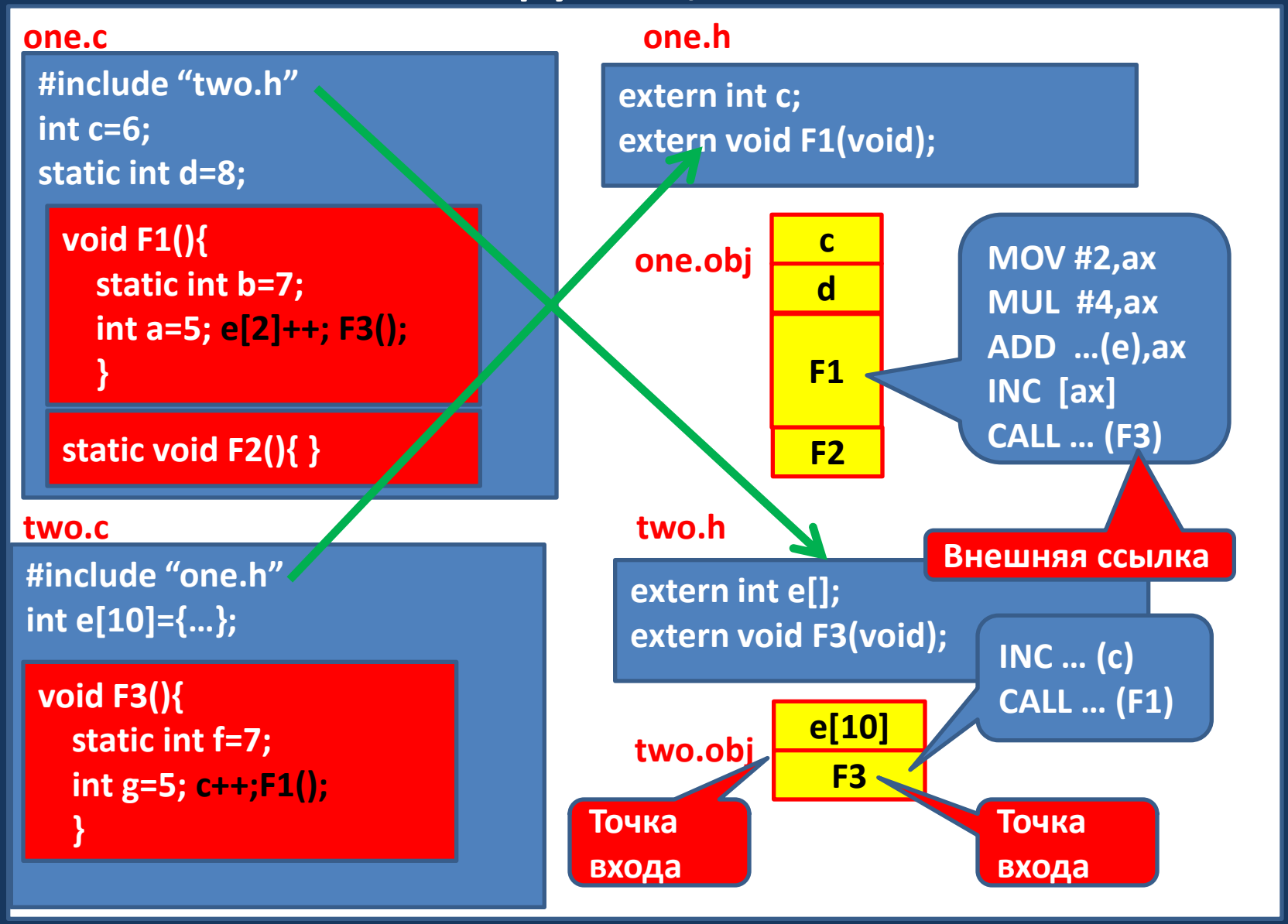
```
int a[100]={1,2,3,4,5};
int F(int b[], int n){… тело…}
```

Объявление – сообщение транслятору об имени и свойствах объекта программы (переменной, функции), который в данной точке недоступен (неизвестен транслятору), но который используется программой

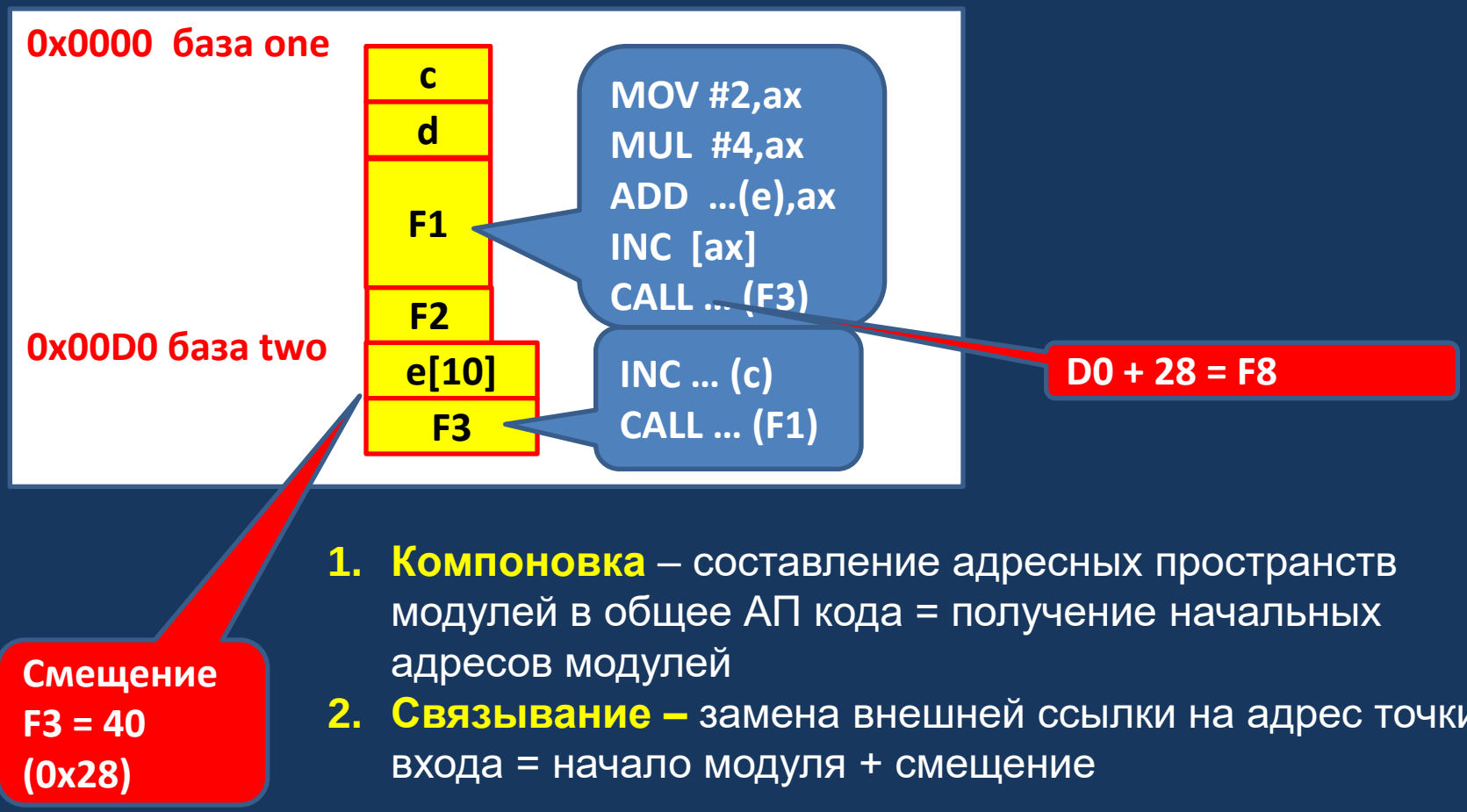
```
extern int a[];
int F(int[], int);
```
- прототип = заголовок функций с абстрактными ТД формальных параметров (имена не важны)

1. Объявление необходимо для генерации кода обращения к объекту программы (переменной, функции) с «точностью до его начального адреса», который на данный момент не известен (внешняя ссылка)
2. Трансляция объявлений не приводит к генерации кода

Модульное программирование. Библиотека функций. Связывание



Модульное программирование. Библиотека функций. Связывание



1. **Компоновка** – составление адресных пространств модулей в общее АП кода = получение начальных адресов модулей
2. **Связывание –** замена внешней ссылки на адрес точки входа = начало модуля + смещение

Контекст кода

Контекст кода – множество объектов (переменных, функций), которые могут быть использованы в данной точке программы «напрямую» (без префиксов):

- Глобальные данные и функции
- Текущее пространство имен (using namespace …)
- Имена импортированных классов
- Статические методы и данные текущего класса
- Методы и данные текущего класса (текущий объект)
- Незатененные данные и неперекрытые методы базового класса
- Формальные параметры текущего метода
- Локальные переменные текущего метода

Контекст кода (что видно) и область видимости (откуда видно) – взаимно обратны

Функция как элемент абстрагирования

- Метафора – вызов функции = **экземпляр функции,** в реальности код разделяется, а для локального контекста создается экземпляр в стеке
- Функция как абстракция параметризованного алгоритма, который может быть вызван с любым **набором параметров,** аналог: переменная = множество значений
- Вызов – элемент **времени выполнения (runtime),** динамическое соответствие параметров
- Еще один уровень абстракции кода– порядок описания не соответствует порядку исполнения
- **Указатель на функцию (имя функции)** как ФП – функциональное программирование, «довесок» к алгоритму (cprog 9.3)
- **Рекурсия** – вызов функцией самой себя, в цикле – дерево вызовов (cprog 7)
- **Поток (нить, thread)** – исполнение функции **параллельно** main (состояние потока = состояние ЦП + отдельный стек потока)