

# Основы программирования на С, часть 2

Наумов Д.А., доц. каф. КТ

Операционные системы и системное программное обеспечение,  
2019

# Содержание лекции

## 1 Функции

# Функции

Программа на языке Си представляет собой набор функций.

- для того, чтобы программа была выполняемой, она должна содержать функцию с названием `main`, которой передается управление при запуске программы на выполнение;
- отсутствует понятие вложенных функций, все функции находятся на одном уровне видимости (глобальном), и их имена должны быть уникальны в рамках программы.

```
[<тип результата>]<имя функции>([список параметров])  
{ [декларации]  
  [операторы]  
}
```

Возврат значения функцией

- `return <выражение>;`
- по умолчанию - возвращаемое значение типа `int`;
- `void` - отсутствие возвращаемого значения.

Прототип - описание функции, средство контроля за соответствием фактических параметров в вызове функции ее формальным параметрам.

```
/*прототип */
int f(char);

...

/* определение */
int f(char c)
{
    /*...тело функции...*/
}
```

новая нотация

```
/*прототип */
int f();

...

/* определение */
int f(c)
char c;
{
    /*...тело функции...*/
}
```

старая нотация

## Прототип

- нужен, если вызов встречается до определения;
- если прототип отсутствует? функция, которая возвращает int;
- int foo() и int foo(void);
- аргументы всегда передаются в функцию 'по значению';

**Задача.** Написать функцию, суммирующую два вещественных значения в двух вариантах: сумма – возвращаемое значение функции и сумма – параметр функции.

```

. . .
double sum1(double,double);/* прототип
                               функции sum1*/
void sum2(double,double,double*); /* прототип
                               функции sum2*/
main()
{ double a1=2.5, u;
  int a2=3;
  u=sum1(a1,a2);
  printf("u=%f\n",u); /* u=5.5 */
  sum2(a1,a2,&u);
  printf("u=%f\n",u); /* u=5.5 */
}
double sum1(double x,double y)
{ return x+y; }
void sum2(double x,double y,double *z)
{ *z=x+y; }
```

## Список параметров переменной длины

```
int printf(const char *format, ...);
```

### Использование

- функции не известны ни типы передаваемых неименованных параметров, ни их количество;
- подключить стандартную библиотеку stdargs.h;
- описать переменную типа va\_list;
- обратиться к макросу va\_start;
- получать значение следующего параметра - va\_arg;
- завершение - va\_end;

```
int f(p1, p2, ...)
{
    va_list parg;
    va_start(parg, p2); /* p2 - последний
                        *именованный параметр функции f */
    int i = va_arg(parg, int);
    ...
    va_end(parg);
}
```

## Общая структура программы

- может размещаться как в одном, так и в нескольких файлах;
- может содержать одну или несколько функций;
- одна из которых считается главной (`main`) - точка входа в программу;
- определение каждой функции должно полностью размещаться в одном файле;
- один этот файл может содержать несколько определений различных функций.

## Характеристики переменных

- область видимости;
- область существования.

## Область видимости переменной

определяет часть исходного текста программы, из любой точки которой доступна данная переменная.

- видимые в пределах блоков (локальные);
- видимые в пределах файла;
- видимые в пределах программы.



Для переменных, определенных в начале любого блока, областью видимости является весь этот блок.

В случае вложенных блоков переменные, определенные внутри вложенного блока, 'перекрывают' переменные с такими же именами, определенные в объемлющем блоке (и так для любой степени вложенности).

```
main()
{ int x = 1;
  if(x>0)
  { int x = 2;
    printf("x = %d\n", ++x); /*выводит: x = 3 */
  }
  printf("x = %d\n", x); /*выводит: x = 1 */
}
```

Переменные, определенные внутри функции, “перекрывают” формальные параметры с теми же именами.

```
int f(int f)
{
    int f = 1;
    ...
}
```

Переменные, определенные вне блоков (т.е., фактически, вне тела какой-либо функции), доступны с точки определения до конца файла. Если на такую переменную нужно сослаться до того, как она определена, необходимо ее описание со спецификатором `extern`.

```
int x;
main()
{
    extern int y;
    x = y = 10;
    printf("x=%d, y=%d\n", x, y); /* x=10, y=10 */
}
...
int y;
```

В файле вне функций не может встречаться несколько определений переменных (возможно, разных типов) с одним и тем же именем.

```
int x;  
main()  
{  
    ...  
}  
float x; /* ошибка: повторное определение x */
```

Указание `static`, примененное к нелокальной переменной или функции, ограничивает область их видимости концом файла.

Если используемая переменная определена в другом программном файле, она также должна быть описана со спецификатором `extern`. При таком описании переменной память под нее не отводится, а только декларируется тип переменной, что позволяет компилятору осуществлять проверки типизации.

## Область существования переменной

множество всех точек программы, при приходе управления на которые переменная существует, т.е. для нее выделена память.

## Статические переменные

- существуют на всем протяжении работы программы;
- выделяется на этапе редактирования внешних связей и загрузки программы;
- тогда же происходит и инициализация статических переменных;
- инициализатором для статической переменной может служить только константное выражение;
- при отсутствии инициализатора статические переменные по умолчанию инициализируются нулем;
- все переменные, определенные вне функций, являются статическими.

## Определение статической переменной, локализованной в блоке

```
int max; /*статическая переменная вне блока*/  
int f(int param)  
{  
    static int min; /* статическая переменная,  
                    определенная внутри блока */  
    ...  
}
```

## Сохранение значений статической переменной при выходе из блока

```
#include <stdio.h>
void print_a();
main()
{ int i;
  for(i=0; i<5; i++)
    print_a();
}

void print_a()
{
  static int a = 1;
  printf("a = %d\n", a++);
}
```

в результате напечатается:

```
a = 1
a = 2
a = 3
a = 4
a = 5
```

## Автоматические переменные

- все переменные определенные внутри блока (функции) и не являющиеся статическими;
- существуют на протяжении работы блока, в котором они определены, включая блоки, вложенные в данный.
- выделение памяти под автоматические переменные и их инициализация осуществляется каждый раз при входе в блок;
- инициализация для автоматических переменных, фактически, эквивалентна присваиванию, т.е. в качестве инициализирующего выражения может выступать любое выражение, которое может стоять в правой части оператора присваивания;
- при отсутствии инициализатора начальное значение по умолчанию для автоматических переменных не определено.



```
#include <stdio.h>
void print_a();
main()
{ int i;
  for(i=0; i<5; i++)
    print_a();
}

void print_a()
{
  int a = 1;
  printf("a = %d\n", a++);
}
```

в результате напечатается:

```
a = 1
a = 1
a = 1
a = 1
a = 1
```

## Регистровые переменные

- квалификатор `register` в определении переменных указывает компилятору, что данную переменную в целях ускорения программы имеет смысл разместить на регистрах, однако компилятор может проигнорировать это указание;
- может применяться только к автоматическим переменным и формальным параметрам функций;
- для регистровых переменных не определено понятие адреса (т.е. не определена операция `&`).

...

**a = 2**

адрес возврата

*сохраненный fp**сохраненные регистры***auto переменная i****auto переменная r**(если ее не удалось  
разместить на регистрах)**a = 1**

адрес возврата

*сохраненный fp**сохраненные регистры***auto переменная i****auto переменная r**(если ее не удалось  
разместить на регистрах)

```
int f(int a){
    int i;
    register int r;
    ...
    r = a--;
    if(r){
        return f(r);
    }
    return r;
}
```

```
main(){
    ...
    int b = f(2);
    ...
}
```

стековый кадр –  
непрерывная область памяти  
стека, используемая функцией  
для своей работы  
*fp (frame pointer)* – указатель  
стекового кадра