Функции

Алтайский государственный университет
Факультет математики и ИТ
Кафедра информатики

Барнаул 2014

Лекция 8

- План
  - Пара заданий для самопроверки
  - Функции
    - Подпрограмма как алгоритмическая структура
    - Функции в языке Си
    - Передача параметров
    - Возврат значений
    - Примеры функций
  - Функции: что еще?
    - Игнорирование возвращаемого значения
    - Тип функции по умолчанию
    - Неопределенное значение функции
    - Тип и аргументы функции `main()`
    - Функции с переменным числом параметров
  - Функции и структура программы
    - Программа из одного файла
    - Программа из многих файлов
    - Области видимости переменных

Пара заданий для самопроверки

- Задание «Поразрядные операции»
- Задание «Оператор выбора»

Задание «Поразрядные операции»

- Что выведет программа?

```
void main()
{
    unsigned char p=0xFF, q=0360, r;
    r = p & ~q;
    printf("%d", r);
}
```

p=11111111
q=11110000

~q=00001111
p=11111111
p&~q=00001111

Вывод: 15

Экзаменационная работа. Задание 1.

- Для выражения укажите порядок вычисления, промежуточные результаты вычисления подвыражений и их тип. Укажите также значения и тип окончательного результата вычисления выражения, стоящего справа от оператора присваивания, и значение с его типом, сохраняемое в переменной, стоящей слева от оператора присваивания.

(f)

```
int Price;
Price = 5U / 2U + 13LU % (4U >> 1U);
```

2, unsigned int

2, unsigned int

1, unsigned long int

3, unsigned long int

3, int

Выражение: Значение: 3 Тип: unsigned long int

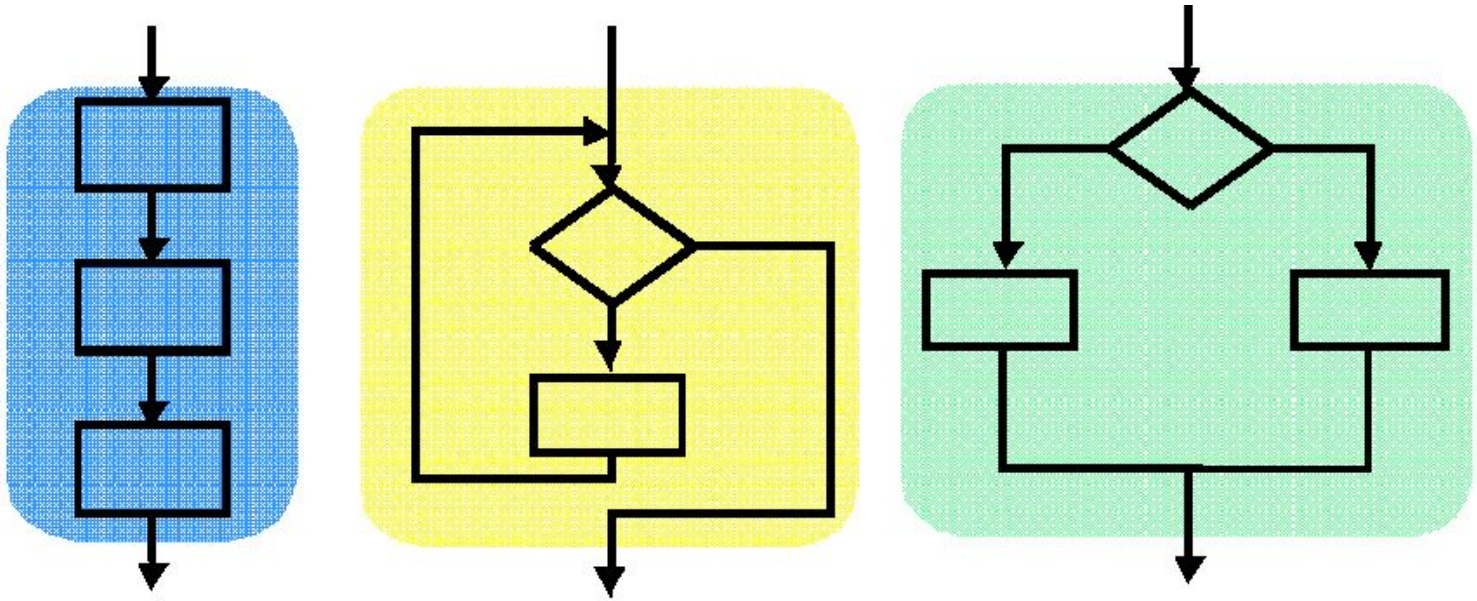
Выражение: Значение: 3 Тип: int

Функции

- Подпрограмма как алгоритмическая структура
- Функции в языке Си
- Передача параметров
- Возврат значений
- Примеры функций

Алгоритмические структуры

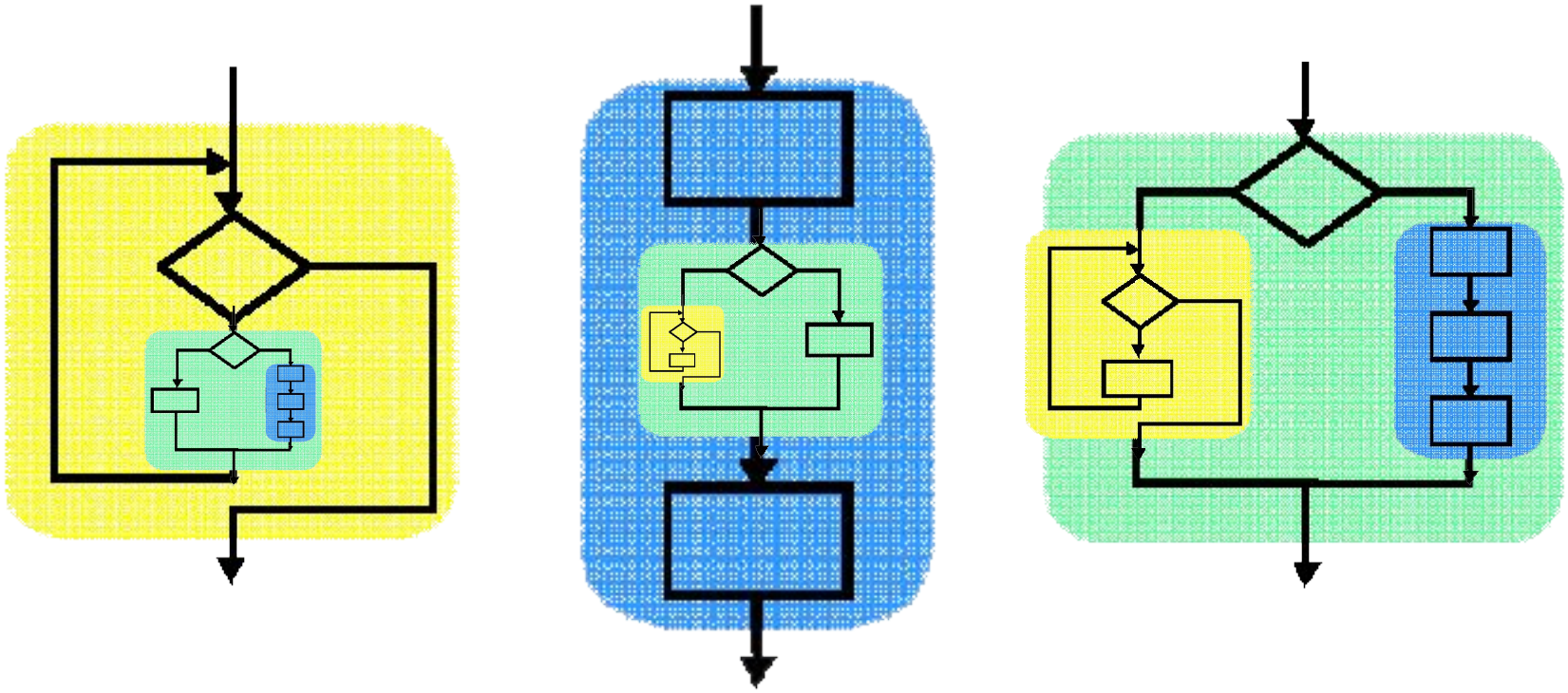
- Базовые алгоритмические структуры
  - Следование
  - Повторение
  - Ветвление



Подпрограмма – еще одна алгоритмическая структура

Алгоритмические структуры

- Алгоритмические структуры вкладываются друг в друга



Подпрограммы

- Часто при построении алгоритмов приходится использовать алгоритмы, составленные ранее
- Алгоритмы, целиком используемые в составе других алгоритмов, называются **вспомогательными** (или подчиненными)
- В алгоритмических языках вспомогательные алгоритмы оформляются в виде **подпрограмм** (процедур, функций)

Подпрограммы

Вернуться домой

1. Позвонить в звонок
2. Если никто не открывает
3. Открыть замок
4. Отворить дверь
5. Иначе
6. Приветствовать открыв
7. Переступить порог
8. Вытереть обувь о ковр
9. Сказать «Вот я и дома!»

Позвонить в звонок

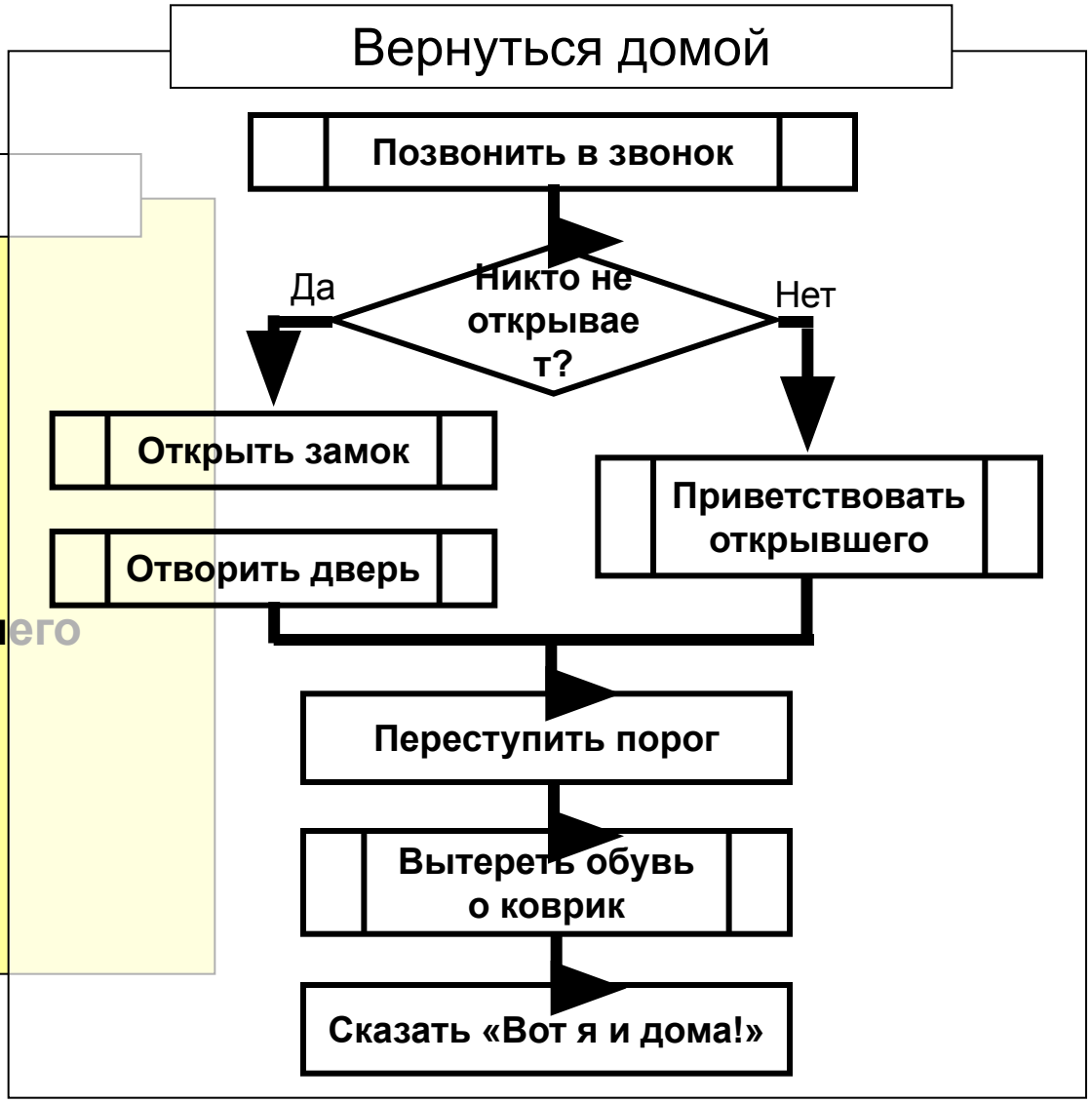
Приветствовать открывшего

1. Если открыла жена
2. Повторять 3 раза
3. Поцеловать в щечку
4. Иначе
5. Если открыла теща
6. Подумать «Опять 25!!!»
7. Сказать «Добрый вечер!»
8. Иначе
9. Сказать «А ты что тут делаешь?»

Подпрограммы

Вернуться домой

1. **Позвонить в звонок**
2. **Если никто не открывает**
3. **Открыть замок**
4. **Отворить дверь**
5. **Иначе**
6. **Приветствовать открывшего**
7. **Переступить порог**
8. **Вытереть обувь о коврик**
9. **Сказать «Вот я и дома!»**



Подпрограммы

Позвонить в звонок

1. **Повторять 3 раза**
2. **Нажать на кнопку звонка**

Вернуться домой

1. **Позвонить в звонок**
2. **Если никто не открывает**
3. **Открыть замок**
4. **Отворить дверь**
5. **Иначе**
6. **Приветствовать открывшего**
7. **Переступить порог**
8. **Вытереть обувь о коврик**
9. **Сказать «Вот я и дома!»**

Заголовок подпрограммы

Тело подпрограммы

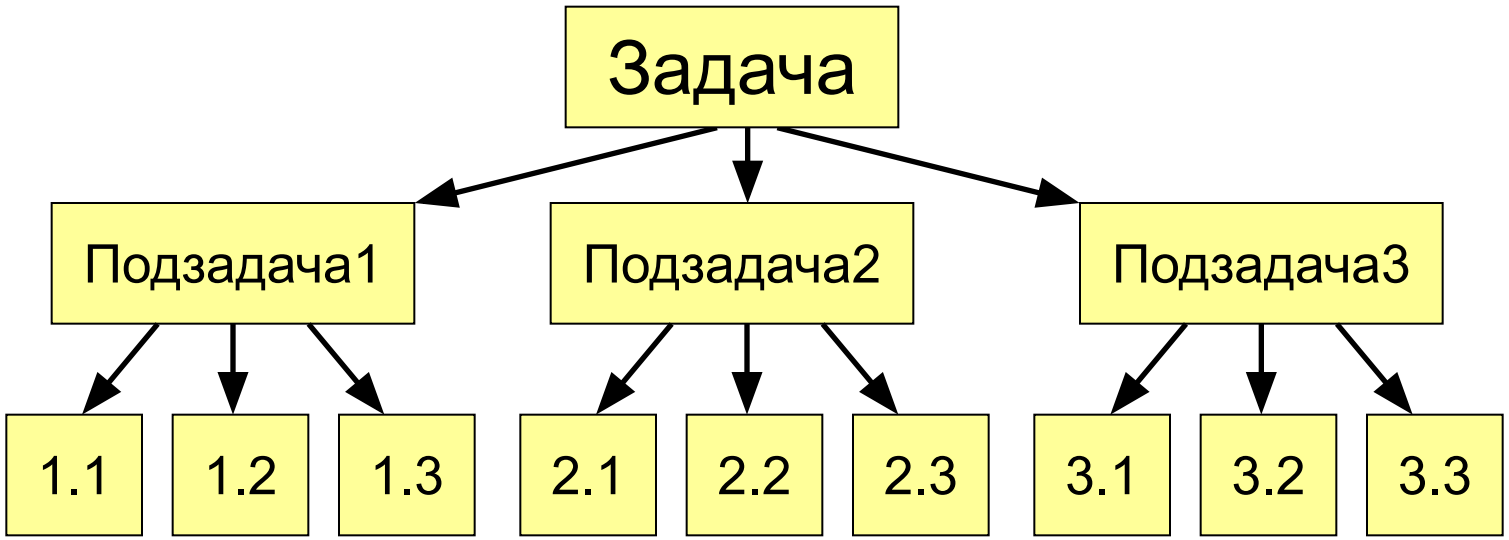
Вызов подпрограммы

Подпрограммы

- Вызов подпрограммы приводит к выполнению ее тела

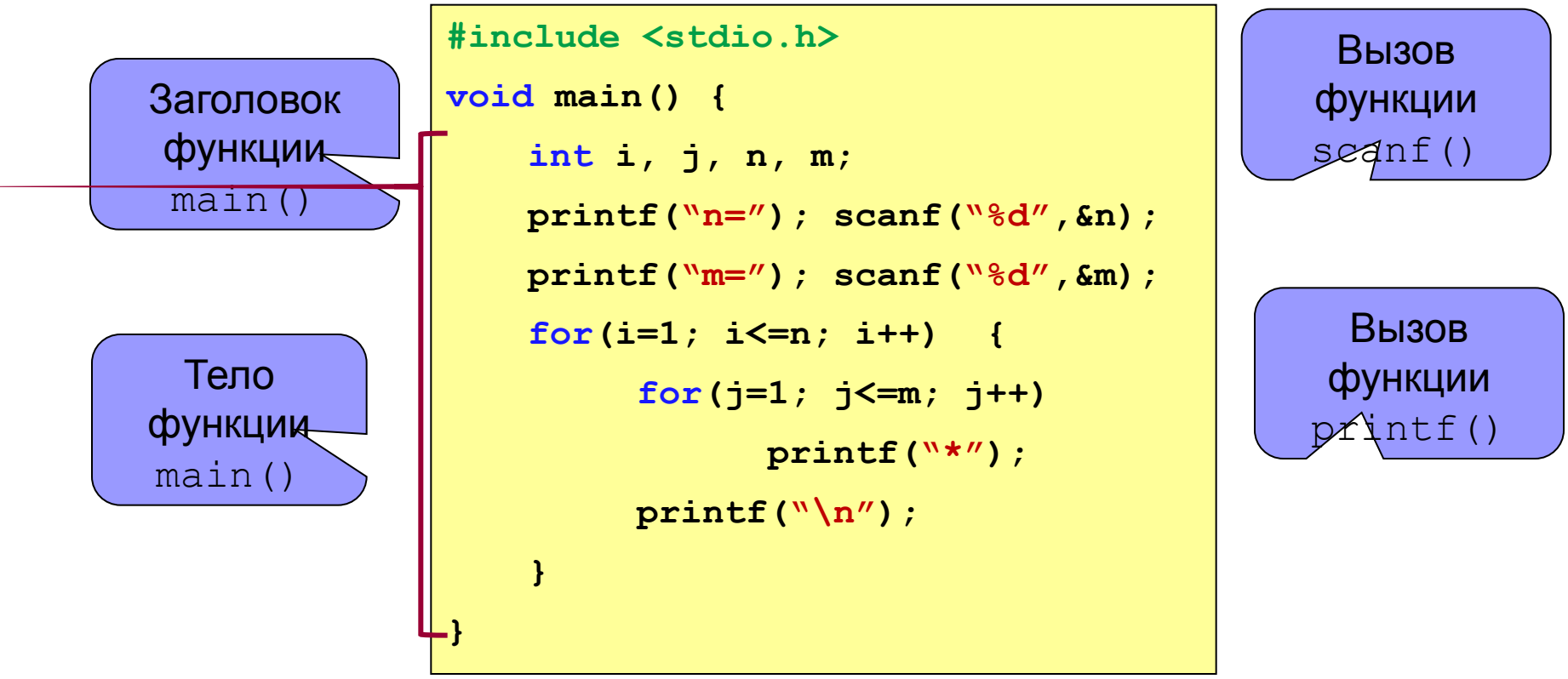
Подпрограммы

- Использование подпрограмм
  - сокращает описание алгоритма (выполнение одинаковых действий в разных местах программы)
  - структурирует описание алгоритма (разбивка программы (или другой подпрограммы) на подзадачи для лучшего восприятия)
  - позволяет реализовать на практике принципы **структурного программирования** при построении больших программ



Подпрограммы в Си

- В Си подпрограммы реализуются в виде функций
- Выполнение программы начинается с вызова функции main(), которая вызывает другие функции



Функции в Си

- Как описать собственную (нестандартную) функцию?
  - Вариант 1
  - Вариант 2

Описание функции должно располагаться **ранее** вызова этой функции

Вызов функции

```
#include <stdio.h>

void starbar() {
    …
}

void main() {
    …
    starbar();
    …
}
```

```
#include <stdio.h>

void starbar();

void main() {
    …
    starbar();
    …
}

void starbar() {
    …
}
```

Объявление функции – **до** использования

Вызов функции

Описание функции -- где угодно (может быть и в другом файле)

Функции в Си

- Как описать собственную (нестандартную) функцию?

Функция, выводящая на экран строку из звездочек

```
#include <stdio.h>

void starbar() {
  int count;
  for (count=1; count <=60;
   count++)
      printf("*");
  printf("\n");
}

void main() {
   …
   starbar();
   …
}
```

Функции в Си

- Как описать собственную (нестандартную) функцию?

```
#include <stdio.h>

void starbar() {
  int count;
  for (count=1; count <=60;
   count++)
      printf("*");
  printf("\n");
}

void main() {
   starbar();
   printf("Привет!");
   starbar();
}
```

Вызов функции

Вызов функции

Функции в Си

- После выполнения программы:

```
**********************************************************************
Привет!
**********************************************************************
```

Функции в Си

- Как описать функцию с параметрами?

```
#include <stdio.h>

void starbar() {
   …
}

void spaces(int n) {
  int count;
  for (count=1; count <=n; count++)
        printf(" ");
  printf("\n");
}

void main() {
   starbar();
   spaces(27); printf("Привет!");
   starbar();
}
```

Функция с параметром

Формальный параметр

Фактический параметр – константа

Вызов функции с параметром

Функции в Си

- После выполнения программы:

```
**********************************************************************
                              Привет!
**********************************************************************
```

Функции в Си

- Как описать функцию с параметрами?

```
#include <stdio.h>

void starbar() {   …
}

void spaces(int n) {
  int count;
  for (count=1; count <=n; count++)
        printf(" ");
  printf("\n");
}

void main() {
   int space_num=27;
   starbar();
   spaces(space_num);
   printf("Привет!");
   starbar();
}
```

Формальный параметр

Фактический параметр – переменная

Вызов функции с параметром

Функции в Си

- Как описать функцию с параметрами?

```
#include <stdio.h>

void starbar() {   …
}

void spaces(int n) {
  int count;
  for (count=1; count <=n; count++)
        printf(" ");
  printf("\n");
}

void main() {
   starbar();
   spaces((60-7)/2);  printf("Привет!");
   starbar();
}
```

Формальный параметр

Фактический параметр – выражение

Вызов функции с параметром

Функции в Си

- Как описать функцию с несколькими параметрами?

```
#include <stdio.h>

void starbar(int n, int m) {
  int i, j;
  for (i=1; i <=n; i++) {
     for (j=1; j <=m; j++)
        printf("*");
     printf("\n");
  }
}

void main() {
   starbar(4,15);
}
```

Формальные параметры: перечисляются с указанием типа через запятую

Вызов функции с параметрами

Фактические параметры

Функции в Си

- После выполнения программы:

```
***************
***************
***************
***************
```

Функции в Си

Параметры функций

- в заголовке функции перечисляются **формальные** параметры, они обозначаются именами, поскольку могут меняться

```
void tr( int x, int y, int c )
```

- при вызове функции в скобках указывают **фактические** параметры (константы или выражения) **в том же порядке**

```
tr ( 200, 100, COLOR(255,0,0));
```

x y c

Функции в Си

Параметры функций

- для каждого формального параметра в заголовке функции указывают его **тип**

```
void A ( int x, float y, char z ) { ... }
```

- внутри функции параметры используются так же, как и прочие переменные
- в функции можно объявлять дополнительные **локальные переменные**, остальные функции не имеют к ним доступа

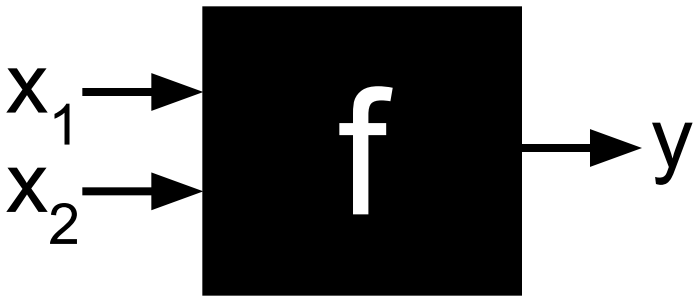
```
void A ( int x, float y, char z )
 {
    int a2, bbc =
            345;
   ...
 }
```

локальные переменные

Функции в Си

- Функция как «черный ящик»

$y = f(x_1, x_2)$



Вход

Функция: внутренний механизм скрыт

Выход

```
#include <stdio.h>

void starbar() {
  int count;
  …
}

void spaces(int n) {
  int count;
  …
}

void main() {
   starbar();
   spaces(27);   printf("Привет!");
   starbar();
}
```

Механизм скрыт от головной и прочих функций

Выход – последовательность пробелов

Вход

Функции в Си

- Возвращаемое значение функции

void указывает на отсутствие возвращаемого значения

```
#include <stdio.h>

void starbar(int n, int m) {
  int i, j;
  for (i=1; i <=n; i++) {
     for (j=1; j <=m; j++)
        printf("*");
     printf("\n");
  }
}

void main() {
   starbar(4,15);
}
```

Функции в Си

- Возвращаемое значение функции

При наличии возвращаемого значения нужно указать его тип

Выходная величина возвращается оператором **return**

```
#include <stdio.h>

double max(double a, double b) {
    if (a > b)
        return a;
    else
        return b;
}

void main() {
  double x=10.5, y=20;
   printf("max(%lf, %lf) = %lf\n",
            x, y, max(x,y));
}
```

После выполнения **return** функция прекращает свою работу

Функции в Си

- Возвращаемое значение функции

Функция без возвращаемого значения

Оператор **return** используется без аргумента

```
#include <stdio.h>

void WritePositive(int a) {
    if (a < 0)
        return;
    printf("%d", a);
}

void main() {
    int n= -10, p=100;
    WritePositive(n);
    WritePositive(p);
}
```

Выполнение **return** прекращает работу функции

Функции в Си

- После выполнения программы:

Функции: резюме

- Шаблон описания функции

```
<тип-результата> <имя-функции> (<список параметров, если есть>) {
    <объявления>
    <инструкции>
}
```

- Шаблон определения функции

```
<тип-результата> <имя-функции> (<список параметров, если есть>);
```

- В определении можно опускать имена параметров (только типы)
- До вызова функции – ее описание или определение
- Если нет возвращаемого значения, *<тип-результата>* = **`void`**
- Возврат результата и прекращение функции:
  - **`return`** `<результат>;` или
  - **`return;`**
- Параметры передаются по значению
  - фактические параметры могут быть константами и выражениями
  - формальные параметры – локальные переменные

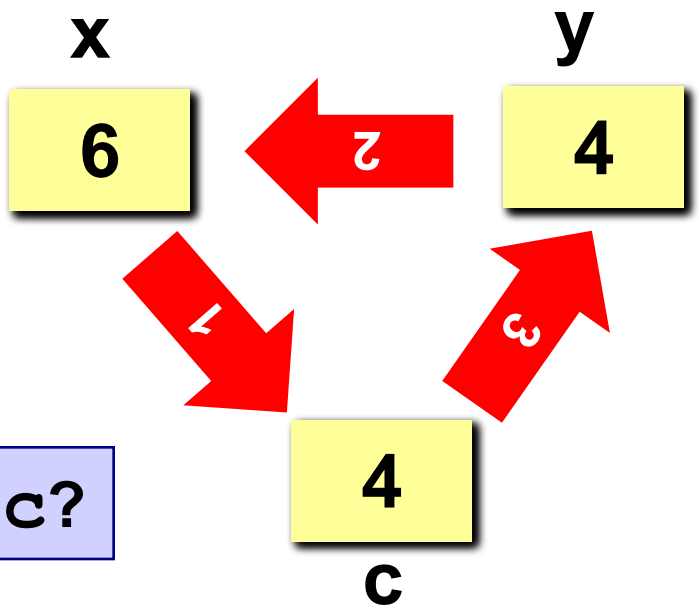
Как поменять местами?

Задача: поменять местами содержимое двух чашек.

Задача: поменять местами содержимое двух ячеек памяти.

```
x = y;
y = x;
```

```
c = x;
x = y;
y = c;
```



Можно ли обойтись без c?

Параметры, передаваемые по указателю

Задача: составить процедуру, которая меняет местами значения двух переменных.

Особенности: надо, чтобы изменения, сделанные в процедуре, стали известны вызывающей программе.

```
void swap ( int a, int b )
{
  int c;
  c = a; a = b; b = c;
}
main()
{
  int x = 1, y = 2;
  swap ( x, y );
  printf ( "x = %d,  y = %d", x, y );
}
```

эта процедура работает с **копиями** параметров

x = 1, y = 2

Параметры, передаваемые по указателю

```
void swap ( int * a, int * b )
{
  int c;
  c = *a; *a = *b; *b = c;
}
```

передаются не значения, а адреса переменных

Применение:

таким образом функция может возвращать несколько значений

Вызов:

```
swap ( 2, 3 );       // числа
swap ( x+z, y+2 ); // выражения
swap ( &x, &y );     // адреса переменных
```

Функции: пример 1

Задача: составить функцию, которая вычисляет наибольшее из двух значений, и привести пример ее использования

Функция:

тип результата

формальные параметры

```
int Max ( int a, int b )
{
  if ( a > b ) return a ;
  else         return b ;
}
```

return - вернуть результат функции

Функции: пример 2

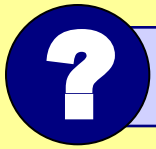
Задача: **составить функцию, которая определяет, верно ли, что заданное число – простое.**

Особенности:

- ответ – **логическое** значение: «да» (1) или «нет» (0)
- результат функции можно использовать как логическую величину **в условиях** (`if, while`)

Алгоритм: считаем число делителей в интервале от **2** до **N-1**, если оно не равно нулю – число составное.

```
count = 0;
for (i=2; i<N; i++)
  if (N % i == 0) count ++;
if ( count == 0 )
      // число N простое}
else // число N составное
```



Как улучшить?

Функция: пример 2

```
int Prime ( int N )
{
  int count = 0, i;
  for (i = 2; i*i <= N; i++)
    if (N % i == 0) count ++;
  return (count == 0);
}
```

перебор только до

```
if (count == 0) return 1;
else            return 0;
```

Функции: пример 2

```
#include <stdio.h>
int Prime ( int N )
{
...
}
main()
{
  int N;
  printf ( "Введите целое число\n" );
  scanf ( "%d", &N );
  if ( Prime( N ) )
       printf ("%d - простое число", N);
  else printf ("%d - составное число", N);
}
```

функция

Функции: что еще?

- Игнорирование возвращаемого значения
- Тип функции по умолчанию
- Неопределенное значение функции
- Тип и аргументы функции `main()`
- Функции с переменным числом параметров

Функции: что еще?

- При вызове функции можно игнорировать возвращаемое значение, вызывая ее как процедуру

```
#include <stdio.h>

int calc_and_print (int a, int b) {
    int c = a + b;
    printf("%d+%d=%d", a, b, c);
    return c;
}

void main() {
  int z;
  z=calc_and_print(4,15);
  calc_and_print(z,33);
}
```

Вызов с сохранением значения

Вызов, игнорирующий возвращаемое значение

Функции: что еще?

- Если при описании функции не указан тип возвращаемого значения, то подразумевается **int**

```
int calc_and_print (int, int)
```

```
#include <stdio.h>

calc_and_print (int a, int b) {
    int c = a + b;
    printf("%d+%d=%d", a, b, c);
    return c;
}

void main() {
  int z;
  z = calc_and_print(4,15) % 7;
}
```

Функции: что еще?

- Если при описании функции не использован **`return`** для возврата значения, то значение функции **не определено**

```
#include <stdio.h>

int calc_and_print (int a, int b) {
    int c = a + b;
    printf("%d+%d=%d", a, b, c);
}

void main() {
  int z;
  z = calc_and_print(4,15);
  printf("%d", z);
}
```

Нет `return`

В результате будет получен «мусор»

Функции: что еще?

- Функция **main()** может иметь тип **int** или **void**
- Возвращаемое значение в **main()** – код завершения процесса в ОС
  - 0 – нет ошибок
  - >0 – код ошибки

```
#include <stdio.h>

void main() {
  int z=0xCAFE;
  printf("%d", z);
}
```

Код ошибки – 255

```
#include <stdio.h>

int main() {
  int z=0xCAFE;
  scanf("%d",&z);
  if(z) {
    printf("%d", z);
    return 0;
  }
  else
   return 255;
}
```

Функции: что еще?

- Функция **main()** может иметь параметры
  - **int argc** количество параметров командной строки
  - **char *argv[]** массив строк – значений параметров

```
#include <stdio.h>

void main(int argc, char *argv[]) {
  for(int i=0; i<argc; i++)
     printf("%d: %s\n", i, argv[i]);
}
```

```
c:\> my_prog.exe a.txt 10
0: C:\TEMP\my_prog.exe
1: a.txt
2: 10
```

argv[0] – всегда полный путь к исполняемому файлу

Функции: что еще?

- Функции могут иметь переменное количество параметров

```
#include <va_list.h>

int sred_znach(int x,...) {
   int i=0, j=0, sum=0;
   va_list uk_arg;
   va_start(uk_arg,x); /*установка указателя uk_arg на    */
                       /*  первый необязательный параметр*/
   if (x!=-1) sum=x;   /*проверка на пустоту списка */
   else return (0);
   j++;
   while ( (i=va_arg(uk_arg,int))!=-1) /* выборка очередного */
   {                                   /* параметра и проверка*/
      sum+=i;                          /* на конец списка */
      j++;
   }
   va_end(uk_arg); /* закрытие списка параметров*/
   return (sum/j);
}

int main() {
   int n;
   n=sred_znach(2,3,4,-1); /* вызов с четырьмя параметрами*/
   printf("n=%d",n);
   n=sred_znach(5,6,7,8,9,-1); /* вызов с шестью параметрами */
   printf("n=%d",n); return (0);
}
```

Функции и структура программы

- Программа из одного файла
- Программа из многих файлов
- Области видимости переменных

Функции и структура программы

- Функции в одном файле

first.c

```
#include <stdio.h>

int add(int a, int b) {
  return a+b;
}

int sub(int,int);

void main() {
  int x=1, y=10, z=0;
  z = add(x,y) + sub(y,x);
  printf("%d", z);
}

int sub(int a, int b) {
  return a-b;
}
```

Функции и структура программы

- Программа из нескольких файлов

first.c

```c
#include <stdio.h>

int add(int a, int b) {
  return a+b;
}

int sub(int,int);
int mul(int,int);
int div(int,int);

void main() {
  int x=1, y=10, z=0;
  z = add(x,y) + sub(y,x);
  z += mul(x,y);
  z += div(y,x);
  printf("%d", z);
}

int sub(int a, int b) {
  return a-b;
}
```

second.c

```c
int mul(int a, int b) {
  return a+b;
}
```

third.c

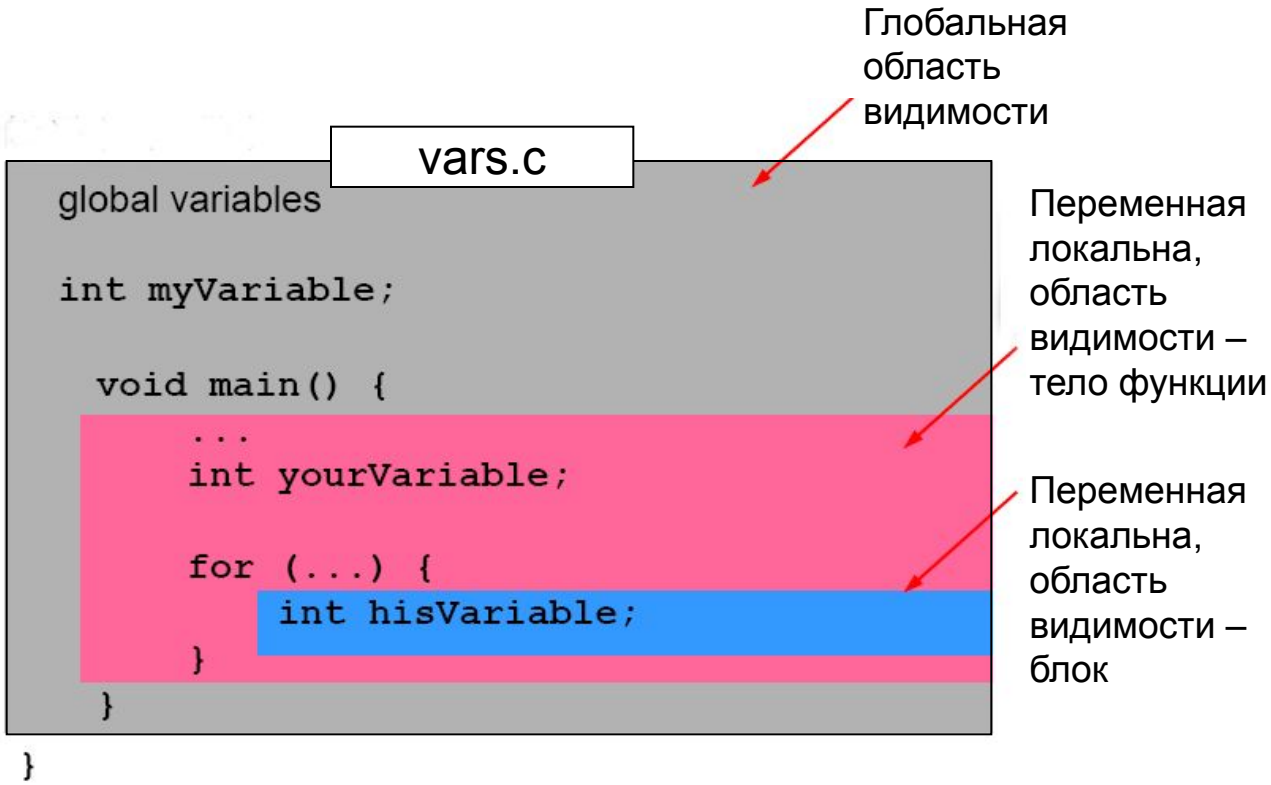
```c
#include <stdlib.h>

int div(int a, int b) {
  return a+b;
}

int adiv(int a, int b) {
  return abs(a) + abs(b);
}
```

Области видимости переменных

- Переменные доступны только в той области видимости, где они описаны (за исключением описанных как **static**)

Переменная `yourVariable` доступна внутри функции `main()`

Переменная `hisVariable` доступна только внутри блока { … }



Рекурсивные функции

- Рекурсия: в математике и программировании
- Общий вид рекурсии
- Задача о ханойских башнях
- Цена рекурсии

Рекурсия в математике

- Рекурсия – метод определения множества объектов через себя, с использованием ранее заданных частных определений.
  - Факториал $n! = n \times (n-1)!$ при $n>0$ и $n! = 1$ при $n=0$
  - Числа Фибоначчи: $F_1 = F_2 = 1$, $F_n = F_{n-1} + F_{n-2}$, при $n>2$

Рекурсия в программировании

- Рекурсия – вызов функции из нее самой напрямую или через другие функции

```
#include <stdio.h>

long fact(long n) {
  if( n == 0 ) return 1;
  else return n*fact(n-1);
}

void main() {
  long N=5;
  printf("%d! = %d\n", N, fact(N));
}
```

Функция вычисления факториала

В теле функции вызывается она сама

Общий вид рекурсии

Если *(простейший случай)* ***тогда***

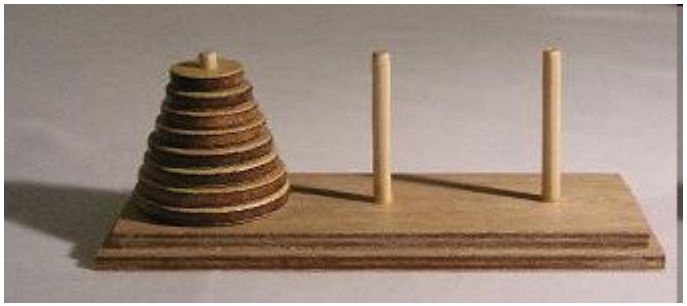
Решить напрямую

Иначе

Делать рекурсивный вызов до появления простейшего случая

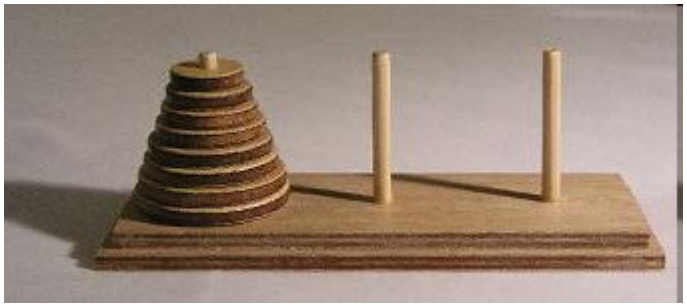
Задача о ханойских башнях

- В одном буддийском монастыре монахи уже тысячу лет занимаются перекладыванием колец. Они располагают тремя пирамидами, на которых надеты кольца разных размеров. В начальном состоянии 64 кольца были надеты на первую пирамиду и упорядочены по размеру. Монахи должны переложить все кольца с первой пирамиды на вторую, выполняя единственное условие — кольцо нельзя положить на кольцо меньшего размера. При перекладывании можно использовать все три пирамиды. Монахи перекладывают одно кольцо за одну секунду. Как только они закончат свою работу, наступит конец света.



Задача о ханойских башнях

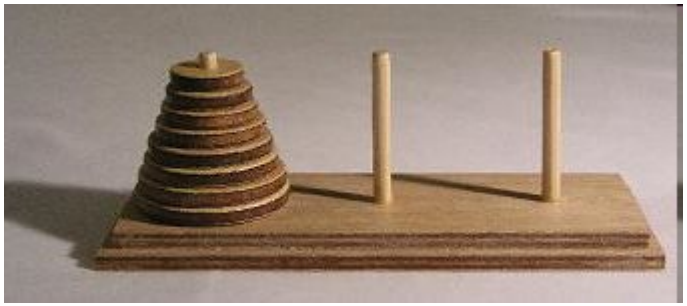
- Рекурсивное решение
  - Итак, нам необходимо перенести *n* дисков со стержня (a) на стержень (c).
  - Если есть функция перенесения *n* –1 диска, тогда задача легко разрешима.
  - Вначале перенесем *n* –1 диск со стержня (a) на стержень (b)
  - Применяя рекурсивный вызов той же функции, затем перенесем *n*-ый диск со стержня (a) на стержень (c)
  - И, наконец, перенесем *n* –1 диск со стержня (b) на стержень (c).
  - Конец света



Задача о ханойских башнях

- Рекурсивное решение

```
void Step(int n, char a, char b, char c)
    // n - количество колец;
    // a, b, c - башни;
    {
        // т. к. на каждом шаге количество колец
        // будет уменьшаться на один,
        // это условие будет условием выхода из рекурсии
        if (n <= 0) return;
        Step(n-1, a, c, b);
        printf("диск %d с %c на %c \n", n, a, b);
        Step(n-1, c, b, a);
    }
```



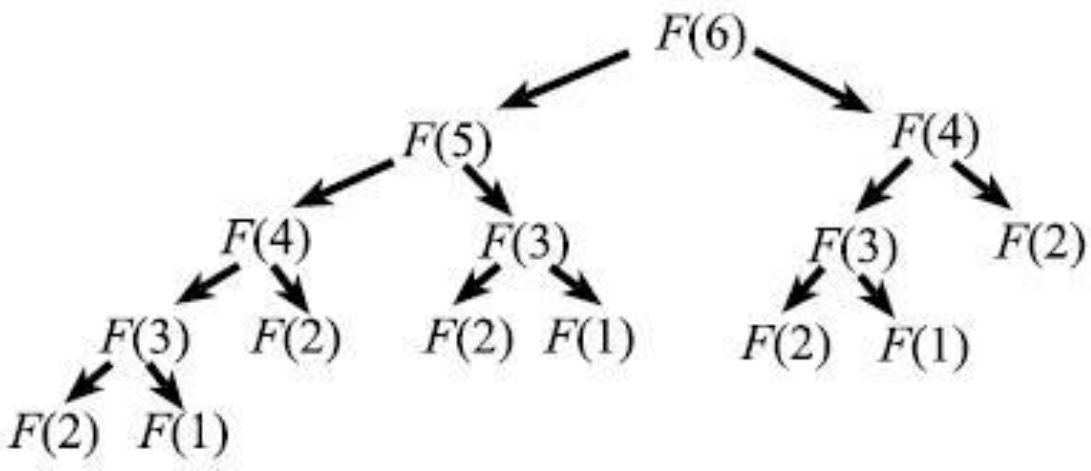
Цена рекурсии

- Использование рекурсии может сократить размер исходного кода программы и сделать код более элегантным и понятным. Однако рекурсия имеет и свои недостатки…

Цена рекурсии

- Пример – вычисление чисел Фибоначчи

```
long F( int n )
{
  if( n <= 1 )
       return n;
  else
       return F( n - 1 ) + F( n - 2 );
}
```



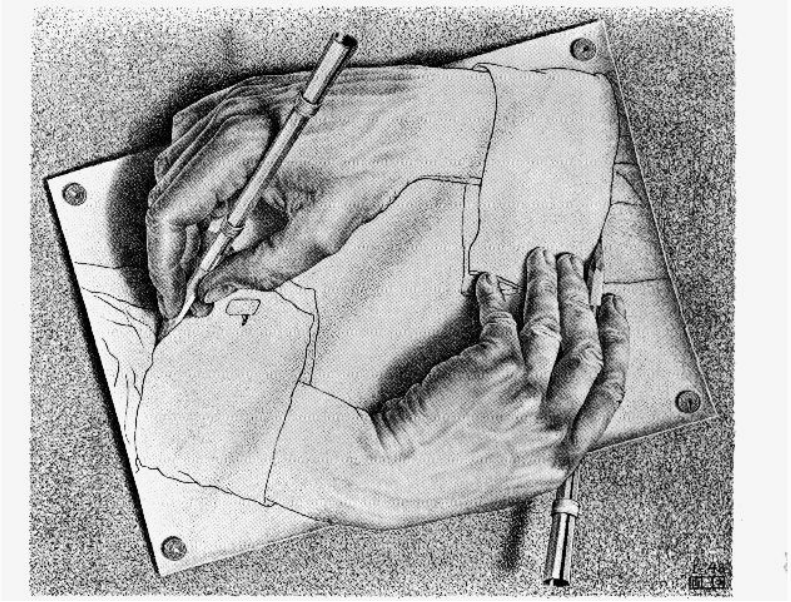
F(3) вычисляется трижды!

Цена рекурсии

- При рекурсивном вызове функции запоминается ее состояние, чтобы после окончания рекурсивного вызова можно было продолжить ее вычисление
- Состояние функции – совокупность значений всех локальных переменных функции
- Значения локальных переменных запоминаются в стэке (специальной области памяти)
- Стэк имеет ограниченный размер и не позволяет глубокие рекурсии

Рекурсия

- Рекурсия всегда(!) может быть заменена итеративным алгоритмом
- При использовании итеративного алгоритма, как правило, необходимо самостоятельно имитировать работу стэка



Вопросы?

- Функции
  - Подпрограмма как алгоритмическая структура
  - Функции в языке Си
  - Передача параметров
  - Возврат значений
  - Примеры функций
- Функции: что еще?
  - Игнорирование возвращаемого значения
  - Тип функции по умолчанию
  - Неопределенное значение функции
  - Тип и аргументы функции `main()`
  - Функции с переменным числом параметров
- Функции и структура программы
  - Программа из одного файла
  - Программа из многих файлов
  - Области видимости переменных