

# Работа на ЭВМ и программирование (группа 112)

Занятие 4

# Контактная информация

- Шундеев Александр Сергеевич
- [alex.shundeev@gmail.com](mailto:alex.shundeev@gmail.com)
- <http://group112.github.io/sem1.html>

# Электронная почта

- Тема письма

- 112 Фамилия Имя Отчество
- 112 Фамилия Имя

- Пример

- 112 Иванов Иван Иванович
- 112 Иванов Иван

Стиль оформления программы

# Стиль Олмана и стиль Кернигана и Ритчи

```
int foo(int x, double y)
{
    double u, v;
    int z;

    z = bar(u, v, z);
    v ++;

    if(z)
    {
        u = abc(u, v);
        zoo();
    }

    return z + x;
}
```

```
int foo(int x, double y) {
    double u, v;
    int z;

    z = bar(u, v, z);
    v ++;

    if(z) {
        u = abc(u, v);
        zoo();
    }

    return z + x;
}
```

# Оформление блока

```
int foo(int x, double y)
{
    double u, v;
    int z;

    z = bar(u, v, z);
    v ++;

    if(z)
    {
        u = abc(u, v);
        zoo();
    }

    return z + x;
}
```

```
int foo(int x, double y) {
    double u, v;
    int z;

    z = bar(u, v, z);
    v ++;

    if(z) {
        u = abc(u, v);
        zoo();
    }

    return z + x;
}
```

# Открывающая скобка {

```
int foo(int x, double y)
{ <пусто>
    double u, v;
    int z;

    z = bar(u, v, z);
    v ++;

    if(z)
    { <пусто>
        u = abc(u, v);
        zoo();
    }

    return z + x;
}
```

```
int foo(int x, double y) { <пусто>
    double u, v;
    int z;

    z = bar(u, v, z);
    v ++;

    if(z) { <пусто>
        u = abc(u, v);
        zoo();
    }

    return z + x;
}
```

# Закрывающая скобка }

```
int foo(int x, double y)
{
    double u, v;
    int z;

    z = bar(u, v, z);
    v ++;

    if(z)
    {
        u = abc(u, v);
        zoo();
    }

    return z + x;
}
```

```
int foo(int x, double y) {
    double u, v;
    int z;

    z = bar(u, v, z);
    v ++;

    if(z) {
        u = abc(u, v);
        zoo();
    }

    return z + x;
}
```



# Закрывающая скобка } (неправильно)

```
int foo(int x, double y)
{
    double u, v;
    int z;

    z = bar(u, v, z);
    v ++;

    if(z)
    {
        u = abc(u, v);
        zoo();
    }

    return z + x;
}
```

```
int foo(int x, double y) {
    double u, v;
    int z;

    z = bar(u, v, z);
    v ++;

    if(z) {
        u = abc(u, v);
        zoo();
    }

    return z + x;
}
```

# Отдельные строки для операторов

```
int foo(int x, double y)
{
    double u, v;
    int z;

    z = bar(u, v, z);
    v ++;

    if(z)
    {
        u = abc(u, v);
        zoo();
    }

    return z + x;
}
```

```
int foo(int x, double y) {
    double u, v;
    int z;

    z = bar(u, v, z);
    v ++;

    if(z) {
        u = abc(u, v);
        zoo();
    }

    return z + x;
}
```

# Отдельные строки для операторов (неправильно)

```
int foo(int x, double y)
{
    double u, v;
    int z;

    z = bar(u, v, z); v ++;

    if(z)
    {
        u = abc(u, v); zoo();
    }

    return z + x;
}
```

```
int foo(int x, double y) {
    double u, v;
    int z;

    z = bar(u, v, z); v ++;

    if(z) {
        u = abc(u, v); zoo();
    }

    return z + x;
}
```

# Отдельные строки для операторов (неправильно)

```
int foo(int x, double y)
{
    double u, v;
    int z;

    z = bar(u, v, z);
    v ++;

    if(z)
    {
        u = abc(u, v);
        zoo();
    } return z + x;
}
```

```
int foo(int x, double y) {
    double u, v;
    int z;

    z = bar(u, v, z);
    v ++;

    if(z) {
        u = abc(u, v);
        zoo();
    } return z + x;
}
```

# Отступы внутри блока

```
{  
    определение1  
    определение2  
    оператор1  
    оператор2  
    оператор3  
}
```

```
... {  
    определение1  
    определение2  
    оператор1  
    оператор2  
    оператор3  
}
```

# Отступы внутри блока

```
{  
    определение1  
    определение2  
← оператор1  
    оператор2  
    оператор3  
}
```

```
... {  
    определение1  
    определение2  
← оператор1  
    оператор2  
    оператор3  
}
```

Отступ

2 пробела

4 пробела (рекомендую)

1 знак табуляции

**Внимание!** Запрещается смешивать пробелы и знаки табуляции

# Отступы внутри блока (неправильно)

```
{
    определение1
    определение2
    оператор1
    оператор2
    оператор3
}
```

```
... {
    определение1
    определение2
    оператор1
    оператор2
    оператор3
}
```



# Отступы внутри блока

```
int foo(int x, double y)
{
    double u, v;
    int z;

    z = bar(u, v, z);
    v ++;

    if(z)
    {
        u = abc(u, v);
        zoo();
    }

    return z + x;
}
```

```
int foo(int x, double y) {
    double u, v;
    int z;

    z = bar(u, v, z);
    v ++;

    if(z) {
        u = abc(u, v);
        zoo();
    }

    return z + x;
}
```

Определения переменных и операторы выделены разными цветами.

# Отступы внутри блока

```
int foo(int x, double y)
{
    double u, v;
    int z;
    z = bar(u, v, z);
    v ++;
    if(z)
    {
        u = abc(u, v);
        zoo();
    }
    return z + x;
}
```

```
int foo(int x, double y) {
    double u, v;
    int z;
    z = bar(u, v, z);
    v ++;
    if(z) {
        u = abc(u, v);
        zoo();
    }
    return z + x;
}
```

Определения переменных и операторы выделены разными цветами.

# Отступы внутри блока (неправильно)

```
int foo(int x, double y)
{
    double u, v;
    int z;
    z = bar(u, v, z);
    v ++;
    if(z)
    {
        u = abc(u, v);
        zoo();
    }
    return z + x;
}
```

```
int foo(int x, double y) {
    double u, v;
    int z;
    z = bar(u, v, z);
    v ++;
    if(z) {
        u = abc(u, v);
        zoo();
    }
    return z + x;
}
```

Определения переменных и операторы выделены разными цветами.

# Отступы внутри блока

```
int foo(int x, double y)
{
    double u, v;
    int z;

    z = bar(u, v, z);
    v ++;

    if(z)
    {
        u = abc(u, v);
        zoo();
    }

    return z + x;
}
```

```
int foo(int x, double y) {
    double u, v;
    int z;

    z = bar(u, v, z);
    v ++;

    if(z) {
        u = abc(u, v);
        zoo();
    }

    return z + x;
}
```

Определения переменных и операторы выделены разными цветами.

# Оператор if

```
if(...)
  оператор1
```

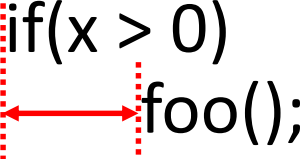
```
if(...)
{
  оператор1
  оператор2
  оператор3
}
```

```
if(...)
  оператор1
```

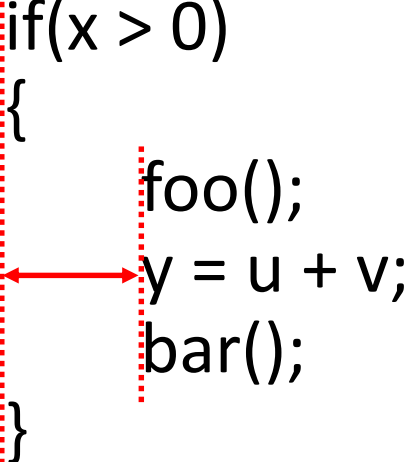
```
if(...) {
  оператор1
  оператор2
  оператор3
}
```

# Оператор if (пример)

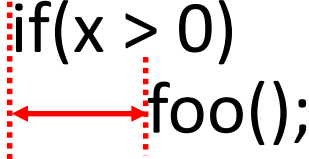
```
if(x > 0)
    foo();
```



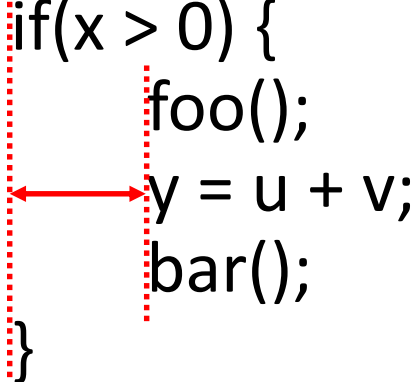
```
if(x > 0)
{
    foo();
    y = u + v;
    bar();
}
```



```
if(x > 0)
    foo();
```



```
if(x > 0) {
    foo();
    y = u + v;
    bar();
}
```



# Оператор if-else

```
if(...)
↔ оператор1
else
↔ оператор2
```

```
if(...)
{
↔ оператор1
↔ оператор2
}
else
{
↔ оператор3
↔ оператор4
}
```

```
if(...)
↔ оператор1
else
↔ оператор2
```

```
if(...) {
↔ оператор1
↔ оператор2
} else {
↔ оператор3
↔ оператор4
}
```

# Оператор for

```
for(...;...;...)  
↔ оператор1
```

```
for(...;...;...)  
{  
↔ оператор1  
↔ оператор2  
↔ оператор3  
}
```

```
for(...;...;...)  
↔ оператор1
```

```
for(...;...;...) {  
↔ оператор1  
↔ оператор2  
↔ оператор3  
}
```



# Структура программы

```
#include <stdio.h>
<пустая строка>
int foo(FILE *fi, double *p);
<пустая строка>
int main(void)
{
    ...
    foo(fi, &x);
    ...
    return 0;
}
<пустая строка>
int foo(FILE *fi, double *p)
{
    ...
}
```

```
#include <stdio.h>
<пустая строка>
int foo(FILE *fi, double *p);
<пустая строка>
int main(void) {
    ...
    foo(fi, &x);
    ...
    return 0;
}
<пустая строка>
int foo(FILE *fi, double *p) {
    ...
}
```

Корректные и некорректные данные

# Постановка задачи

Программа считывает входные данные из файла `input.txt`

Если входные данные корректны:

- программа записывает результат своей работы (вычисленная характеристика числовой последовательности) в файл `output.txt`
- функция `main` возвращает `0`

Если входные данные некорректны:

- функция `main` возвращает `-1`

# Постановка задачи

## Корректные входные данные

В файле `input.txt` через пробел записаны элементы последовательности.

Пример.

```
$ cat input.txt
```

```
1 2 3 4 5
```

В зависимости от условия задачи пустой файл `input.txt` (пустая входная последовательность) может трактоваться и как корректные и как некорректные входные данные

# Постановка задачи

## Некорректные входные данные

Файл `input.txt` отсутствует (не может быть открыт).

Файл `input.txt` содержит «мусор».

Пример.

```
$ cat input.txt  
abc
```

Пример.

```
$ cat input.txt  
1 2 3 4 5 abc
```

# Корректные и некорректные данные

Пустой входной файл и входной файл с числовой последовательностью

## Задачи 5, 6 (целочисленная последовательность)

В случае корректных данных файл `input.txt` должен содержать

`X x1 x2 x3 x4 x5 x6 ...`

Корректные данные

```
$ cat input.txt
```

```
3
```

```
$ cat input.txt
```

```
3 1
```

```
$ cat input.txt
```

```
3 1 2
```

## Задачи 5, 6 (целочисленная последовательность)

В случае корректных данных файл `input.txt` должен содержать

$X \ x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \dots$

Корректные данные

```
$ cat input.txt
```

3

```
$ cat input.txt
```

3 1

```
$ cat input.txt
```

3 1 2

Случай пустой последовательности.



## Задачи 5, 6 (целочисленная последовательность)

В случае корректных данных файл `input.txt` должен содержать

`X x1 x2 x3 x4 x5 x6 ...`

**Внимание!** Пустой файл - это некорректные данные. Должен быть задан `X`

## Задачи 5, 6 (вещественная последовательность)

В случае корректных данных файл `input.txt` должен содержать

$X \ \varepsilon \ x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \dots$

Корректные данные

```
$ cat input.txt
```

```
3.1 1e-16
```

```
$ cat input.txt
```

```
3.1 1e-16 1.1
```

```
$ cat input.txt
```

```
3.1 1e-16 1.1 2.1
```

## Задачи 5, 6 (вещественная последовательность)

В случае корректных данных файл `input.txt` должен содержать

$X \ \varepsilon \ x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \dots$

### Корректные данные

```
$ cat input.txt
```

```
3.1 1e-16
```

```
$ cat input.txt
```

```
3.1 1e-16 1.1
```

```
$ cat input.txt
```

```
3.1 1e-16 1.1 2.1
```

Случай пустой последовательности.

## Задачи 5, 6 (вещественная последовательность)

В случае корректных данных файл `input.txt` должен содержать

$X \ \varepsilon \ x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \dots$

**Внимание!** Некорректные данные:

- Пустой файл
- Файл содержит только одно число ( $X$ )
- Задана отрицательная точность

```
$ cat input.txt
```

```
3.1
```

```
$ cat input.txt
```

```
3.1 -1e-16 1.1 2.1
```

## Задачи 5, 6 (вещественная последовательность)

В случае корректных данных файл `input.txt` должен содержать

$X \ \varepsilon \ x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \dots$

**Внимание!** Некорректные данные:

- Пустой файл
- Файл содержит только одно число ( $X$ )
- Задана отрицательная точность

```
$ cat input.txt
```

3.1

Не задана точность  $\varepsilon$ .

```
$ cat input.txt
```

3.1 -1e-16 1.1 2.1

Задана отрицательная точность  $\varepsilon$ .

## Задача 9 (целочисленная последовательность)

В случае корректных данных файл `input.txt` должен содержать

$c_1$   $c_2$   $c_3$   $d$   $x_1$   $x_2$   $x_3$   $x_4$   $x_5$   $x_6$  ...

Корректные данные (должно быть минимум 4 числа)

```
$ cat input.txt
```

1 1 1 3

```
$ cat input.txt
```

1 1 1 3 1

```
$ cat input.txt
```

1 1 1 3 1 0 2 1

## Задача 9 (целочисленная последовательность)

В случае корректных данных файл `input.txt` должен содержать

$c_1$   $c_2$   $c_3$   $d$   $x_1$   $x_2$   $x_3$   $x_4$   $x_5$   $x_6$  ...

Корректные данные (должно быть минимум 4 числа)

```
$ cat input.txt
```

1 1 1 3

```
$ cat input.txt
```

1 1 1 3 1

```
$ cat input.txt
```

1 1 1 3 1 0 2 1

Случай пустой последовательности.

## Задача 9 (целочисленная последовательность)

В случае корректных данных файл `input.txt` должен содержать

$c_1$   $c_2$   $c_3$   $d$   $x_1$   $x_2$   $x_3$   $x_4$   $x_5$   $x_6$  ...

**Внимание!** Некорректные данные

- Пустой файл
- Файл содержит меньше **четырёх** чисел



## Задача 9 (вещественная последовательность)

В случае корректных данных файл `input.txt` должен содержать

$c_1$   $c_2$   $c_3$   $d$   $\varepsilon$   $x_1$   $x_2$   $x_3$   $x_4$   $x_5$   $x_6$  ...

Корректные данные (должно быть минимум 5 чисел)

```
$ cat input.txt
```

Случай пустой последовательности.

```
1.1 1.1 1.1 3.1 1e-16
```

```
$ cat input.txt
```

```
1.1 1.1 1.1 3.1 1e-16 1.1
```

```
$ cat input.txt
```

```
1.1 1.1 1.1 3.1 1e-16 1.1 0.1 2.1 1.1
```

## Задача 9 (вещественная последовательность)

В случае корректных данных файл `input.txt` должен содержать

$c_1$   $c_2$   $c_3$   $d$   $\varepsilon$   $x_1$   $x_2$   $x_3$   $x_4$   $x_5$   $x_6$  ...

**Внимание!** Некорректные данные

- Пустой файл
- Файл содержит меньше **пяти** чисел
- Задана отрицательная точность

## Задача 9 (целочисленная последовательность)

В случае корректных данных файл `input.txt` должен содержать

$c_1$   $c_2$   $c_3$   $d$   $x_1$   $x_2$   $x_3$   $x_4$   $x_5$   $x_6$  ...

Первый вызов функции `scanf`

```
s = scanf("%d%d%d%d%d%d", &c1, &c2, &c3, &d, &x1, &x2, &x3);
```

## Задача 9 (целочисленная последовательность)

В случае корректных данных файл `input.txt` должен содержать

$c_1$   $c_2$   $c_3$   $d$   $x_1$   $x_2$   $x_3$   $x_4$   $x_5$   $x_6$  ...

Первый вызов функции `scanf`

```
s = scanf("%d%d%d%d%d%d%d", &c1, &c2, &c3, &d, &x1, &x2, &x3);
```

Если  $s < 4$  (не были считаны  $c_1$   $c_2$   $c_3$   $d$ ), то это некорректные данные.

## Задача 9 (целочисленная последовательность)

В случае корректных данных файл `input.txt` должен содержать

$c_1$   $c_2$   $c_3$   $d$   $x_1$   $x_2$   $x_3$   $x_4$   $x_5$   $x_6$  ...

Первый вызов функции `scanf`

```
s = scanf("%d%d%d%d%d%d%d", &c1, &c2, &c3, &d, &x1, &x2, &x3);
```

Если  $4 \leq s < 7$ , то это корректные данные, но ответ **NO**.

# Массивы

Начало

Входной файл input.txt

Вариант 1 (по умолчанию)

$n \ a_0 \ a_1 \ \dots \ a_{n-1}$

Вариант 2

$a_0 \ a_1 \ \dots \ a_{n-1}$

Входной файл input.txt

Вариант 1 (по умолчанию)

$n \ a_0 \ a_1 \ \dots \ a_{n-1}$

размер  
массива

элементы  
массива

Вариант 2

$a_0 \ a_1 \ \dots \ a_{n-1}$

элементы  
массива



# Работа с массивами

Массивы создаются динамически

- Функция `malloc` (выделение памяти)
- Функция `free` (освобождение памяти)

Выделение / освобождение памяти осуществляется внутри функции `main`

Перед выходом из функции `main` (завершением программы) должна быть явно освобождена ранее выделенная память

# Функция обработки массива

```
void fun(double *a, int n);
```

- `a` - адрес памяти, выделенной под хранение массива
- `n` - размер массива
- Могут быть и дополнительные параметры


# Структура программы (1 файл)

```
#include <stdlib.h>
#include <stdio.h>

void fun(double *a, int n);

int main(void)
{
    ...
    fun(a, n);
    ...
}

void fun(double *a, int n)
{
    ...
}
```



# Структура программы (2 файла)

// f1.c

```
#include <stdlib.h>
```

```
#include <stdio.h>
```

```
void fun(double *a, int n);
```

```
int main(void)
```

```
{
```

```
    ...
```

```
    fun(a, n);
```

```
    ...
```

```
}
```

// f2.c

```
#include <stdio.h>
```

```
void fun(double *a, int n);
```

```
void fun(double *a, int n)
```

```
{
```

```
    ...
```

```
}
```

# Команда компиляции

```
gcc -Wall -Wextra -Wfloat-equal -Werror -pedantic -std=c99 f1.c f2.c -o prog
```