

Работа на ЭВМ и программирование (группа 114)

Занятие 3

Контактная информация

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

Электронная почта

- Тема письма

- 114 Фамилия Имя Отчество
- 114 Фамилия Имя

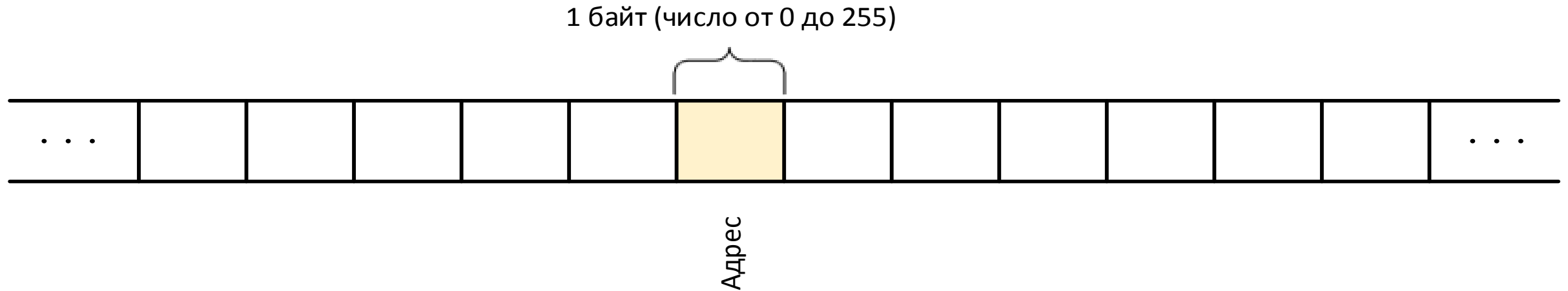
- Пример

- 114 Иванов Иван Иванович
- 114 Иванов Иван

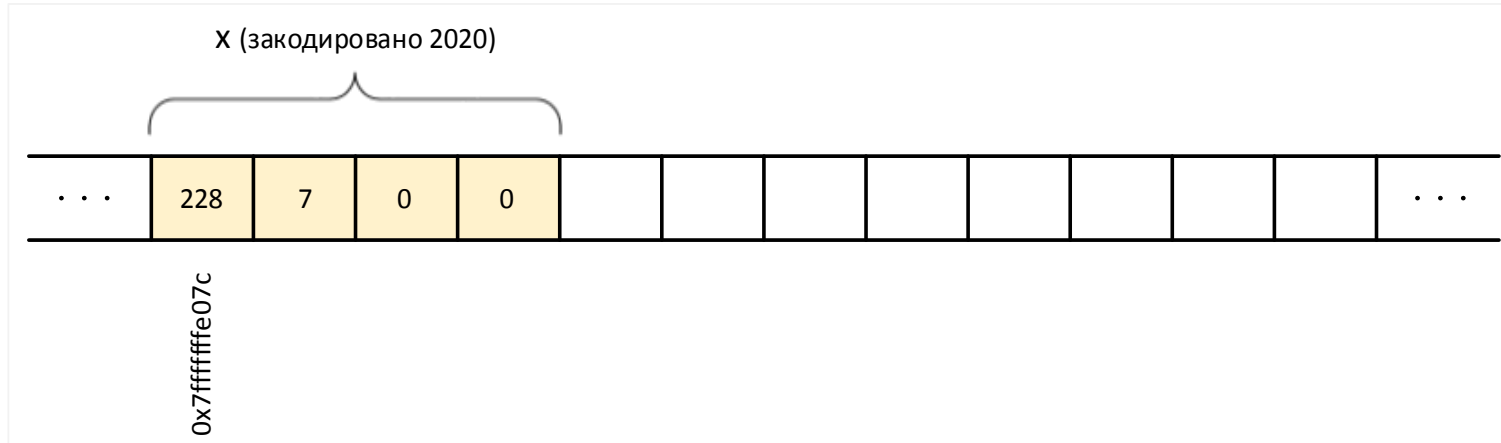
Модель памяти программы

(пользовательской программы)

Виртуальное адресное пространство



Виртуальное адресное пространство



`int x = 2020;`

Адрес переменной x:

`&x (0x7fffffffe07c)`

Значение переменной x:

2020

Указатели

С каждым типом **T** (например, `int` или `double`) связан беззнаковый целый тип указателей на **T**.

Указатель на **T**:

- 0 (специальное выделенное значение)
- адрес памяти, которая может быть выделена под хранение значения типа **T**.

Примеры определения переменных

```
double x;
```

```
double *p;
```

```
double y, z, *q, *r, u;
```


Примеры определения переменных

```
double x;
```

```
double *p;
```

```
double y, z, *q, *r, u;
```

Переменные типа double

Примеры определения переменных

```
double x;
```

```
double *p;
```

```
double y, z, *q, *r, u;
```

Переменные типа указатель double

Операции & и *

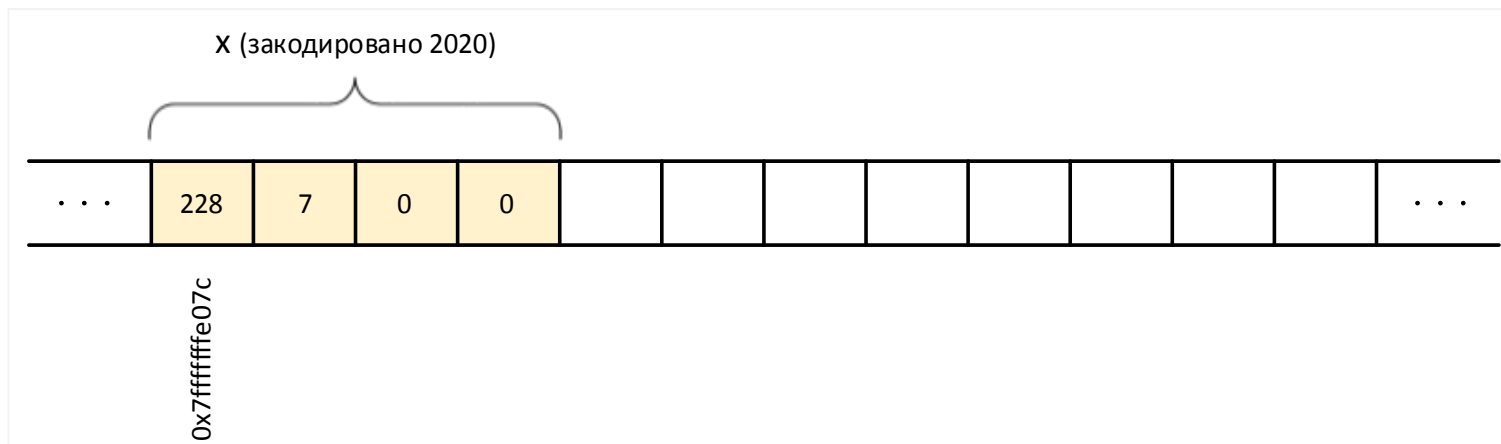
Операция &

& <переменная> результат: <указатель> (адрес переменной)

Обратная операция *

* <указатель> результат: "<переменная>"

Виртуальное адресное пространство



`int x = 2020;`

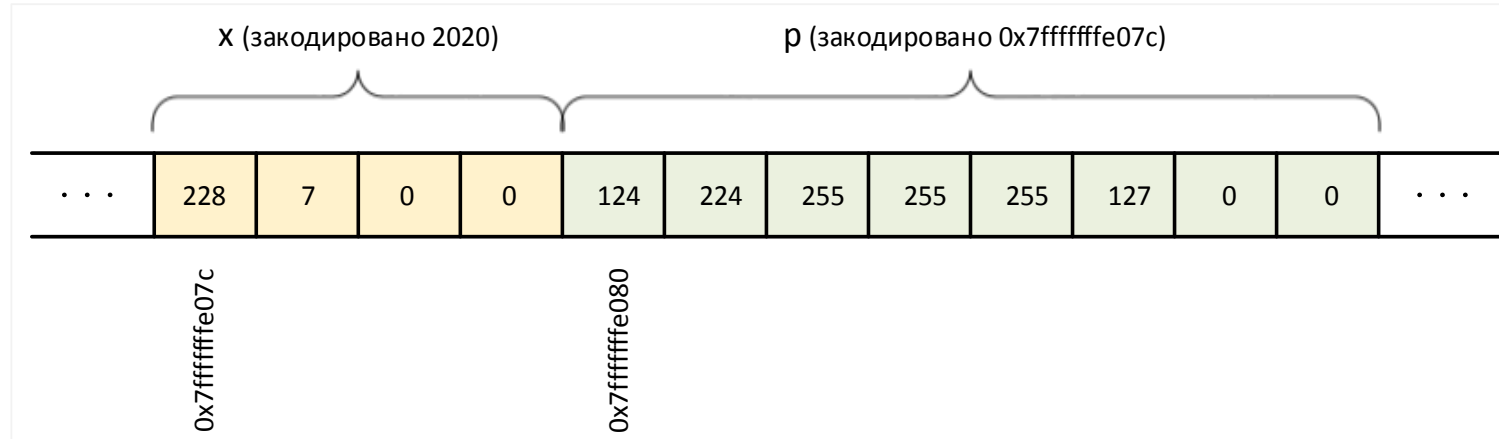
Адрес переменной x:

`&x (0x7fffffffe07c)`

Значение переменной x:

2020

Виртуальное адресное пространство



```
int x = 2020;
```

```
int *p = &x;
```

Адрес переменной x:

Значение переменной x:

Адрес переменной p:

Значение переменной p:

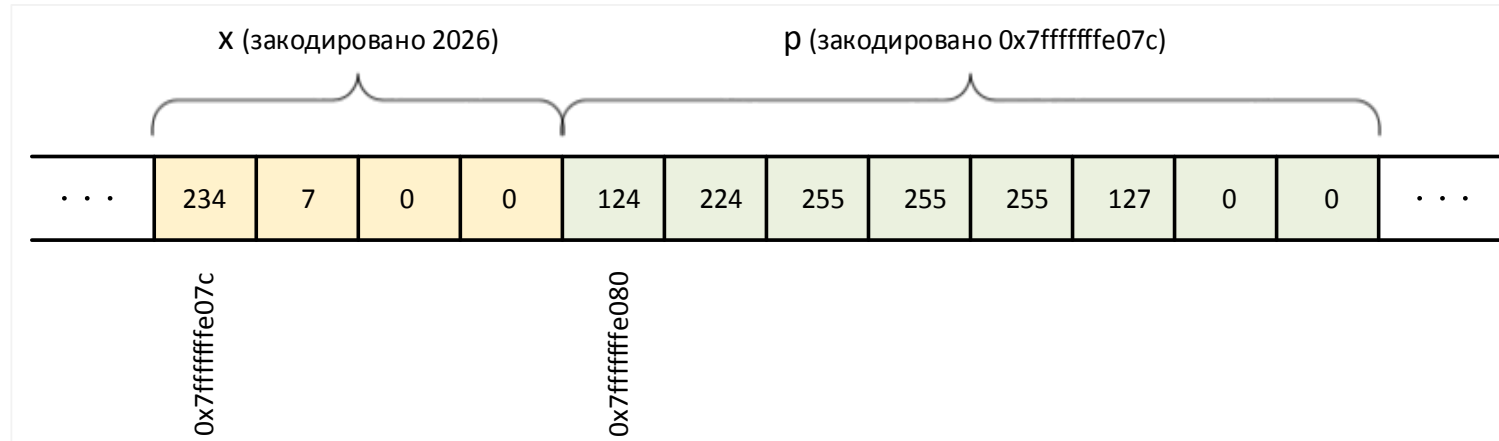
&x (0x7fffffffe07c)

2020

&p (0x7fffffffe080)

0x7fffffffe07c

Виртуальное адресное пространство



```
int x = 2020;
```

```
int *p = &x;
```

```
*p = 226
```

Адрес переменной x:

Значение переменной x:

Адрес переменной p:

Значение переменной p:

Значение переменной x:

&x (0x7fffffffe07c)

2020

&p (0x7fffffffe080)

0x7fffffffe07c

2026

Несколько выходных параметров

Пример

Требуется реализовать функцию, которая на вход получает два числа x , y и возвращает два числа $x+y$, $x-y$.

Внимание! У функции должно быть два выходных параметра.

Несколько выходных параметров

Решение

```
void f(int x, int y, int *p1, int *p2)
{
    *p1 = x + y;
    *p2 = x - y;
}
```


Несколько выходных параметров

Решение

```
void f(int x, int y, int *p1, int *p2)
{
    *p1 = x + y;
    *p2 = x - y;
}
```

Вариант использования

```
int z1, z2;
f(3, 2, &z1, &z2);

// z1 == 5
// z2 == 1
```

Числовые последовательности

Основные определения

Определения

Числовая последовательность длины N :

$$x_1, x_2, \dots, x_N \quad (*)$$

Пустая последовательность (не содержащая элементов) имеет длину 0

Числовая последовательность вида

$$x_i, x_{i+1}, \dots, x_{i+n}$$

называется подпоследовательностью $(*)$

Определения

i -й элемент (*) называется **локальным минимумом**:

если $1 < i < N$ и $x_i < x_{i-1}, x_i < x_{i+1}$;

если $i = 1$ и $x_1 < x_2$;

если $i = N$ и $x_N < x_{N-1}$.

Определения

i -й элемент (*) называется **локальным максимумом**:

если $1 < i < N$ и $x_i > x_{i-1}, x_i > x_{i+1}$;

если $i = 1$ и $x_1 > x_2$;

если $i = N$ и $x_N > x_{N-1}$.

Определения

Последовательность (*) **возрастающая**, если

$$N > 1 \text{ и } x_i > x_{i-1} \ (2 \leq i \leq N)$$

Последовательность (*) **неубывающая**, если

$$N > 1 \text{ и } x_i \geq x_{i-1} \ (2 \leq i \leq N)$$

Определения

Последовательность (*) **убывающая**, если

$$N > 1 \text{ и } x_i < x_{i-1} \ (2 \leq i \leq N)$$

Последовательность (*) **невозрастающая**, если

$$N > 1 \text{ и } x_i \leq x_{i-1} \ (2 \leq i \leq N)$$

Определения

Последовательность (*) **постоянная**, если

$$N > 1 \text{ и } x_i = x_1 \text{ (} 2 \leq i \leq N \text{)}$$

Аналогичные определения остаются в силе и для подпоследовательностей

Числовые последовательности

Постановка задачи

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
```

Постановка задачи

Ограничения

- числовая последовательность, записанная во входном файле `input.txt` обрабатывается за один проход
Нельзя, например, прочитать данные из файла, закрыть файл, а потом повторно его открыть и прочитать данные
- запрещается использовать массивы
Нельзя, например, сохранить все прочитанные элементы последовательности в массив, а потом работать с этим массивом

Постановка задачи

Ограничения

- Файлы открываются только внутри функции `main`.
- Перед завершением программы (выходом из функции `main`) должны быть явно закрыты все ранее открытые файлы.

Тестирование программы

Пример (программа подсчета среднего арифметического значения элементов последовательности)

Корректные входные данные

```
$ echo "1 2 3 4 5" > input.txt
```

```
$ ./prog
```

```
$ echo $?
```

```
0
```

```
$ cat output.txt
```

```
3.000000
```


Корректные входные данные

```
$ echo "1 2 3 4 5" > input.txt
```

Создаем (перезаписываем) входной файл.

```
$ ./prog
```

```
$ echo $?
```

```
0
```

```
$ cat output.txt
```

```
3.000000
```

Корректные входные данные

```
$ echo "1 2 3 4 5" > input.txt
```

```
$ ./prog
```

Запускаем программу обработки последовательности.

```
$ echo $?
```

```
0
```

```
$ cat output.txt
```

```
3.000000
```

Корректные входные данные

```
$ echo "1 2 3 4 5" > input.txt
```

```
$ ./prog
```

```
$ echo $?
```

```
0
```

```
$ cat output.txt
```

```
3.000000
```

Проверяем, что вернула функция `main` программы `prog` (случай `return 0;`).

Корректные входные данные

```
$ echo "1 2 3 4 5" > input.txt
```

```
$ ./prog
```

```
$ echo $?
```

```
0
```

```
$ cat output.txt
```

```
3.000000
```

Печатаем содержимое выходного файла.

Корректные входные данные

```
$ echo "1 2 3 4 5" > input.txt; ./prog; echo $?; cat output.txt
```

```
0
```

```
3.000000
```

Некорректные входные данные

(отсутствует input.txt)

```
$ rm -f input.txt; ./prog; echo $?
```

Can't open file 'input.txt'.

255

Некорректные входные данные

(отсутствует input.txt)

```
$ rm -f input.txt; ./prog; echo $?
```

```
Can't open file 'input.txt'.
```

```
255
```

Удаляем входной файл.

Некорректные входные данные

(отсутствует input.txt)

```
$ rm -f input.txt; ./prog; echo $?
```

 Запускаем программу обработки последовательности.

```
Can't open file 'input.txt'.
```

```
255
```


Некорректные входные данные

(отсутствует input.txt)

```
$ rm -f input.txt; ./prog; echo $?
```

```
Can't open file 'input.txt'.
```

```
255
```

Проверяем, что вернула функция `main` программы `prog` (случай `return -1;`).

Некорректные входные данные

(пустой input.txt)

```
$ rm -f input.txt; touch input.txt; ./prog; echo $?
```

The input file is empty.

255

Некорректные входные данные

(пустой input.txt)

```
$ rm -f input.txt; touch input.txt; ./prog; echo $?
```

Удаляем входной файл.

The input file is empty.

255

Некорректные входные данные

(пустой input.txt)

```
$ rm -f input.txt; touch input.txt; ./prog; echo $?
```

The input file is empty.

255

Создаем пустой входной файл.

Некорректные входные данные

(пустой input.txt)

```
$ rm -f input.txt; touch input.txt; ./prog; echo $?
```

The input file is empty.

255

Запускаем программу обработки последовательности.

Некорректные входные данные

(пустой input.txt)

```
$ rm -f input.txt; touch input.txt; ./prog; echo $?
```

The input file is empty.

255

Проверяем, что вернула функция `main` программы `prog` (случай `return -1;`).

Некорректные входные данные

(input.txt содержит «мусор»)

```
$ echo "abc" > input.txt; ./prog; echo $?
```

Wrong data in the input file.

255

Некорректные входные данные

(input.txt содержит «мусор»)

```
$ echo "1 2 3 abc" > input.txt; ./prog; echo $?
```

Wrong data in the input file.

255

Сдача программы

Должны быть присланы:

- файл с текстом программы;
- файл protocol.txt.

Файл protocol.txt должен содержать:

- команду компиляции программы;
- тесты с некорректными входными данными;
- тесты с корректными входными данными.

Пример оформления файла protocol.txt приведен на сайте.

Работа с файлами

stdio.h

Тип **FILE** для хранения информации об открытых файлах

Напрямую с этим типом не работают (работа только через указатели)

Стандартные функции

- fopen (открытие файла)
- fprintf (форматная печать в файл)
- fscanf (форматное чтение из файла)
- feof (проверка конца файла)
- fclose (заккрытие файла)

stdio.h

Три глобальных переменных переменных типа **FILE***

- `stdin` (стандартный поток ввода)
- `stdout` (стандартный поток вывода)
- `stderr` (стандартный поток сообщений об ошибках)

Вызов функции:

- | | | |
|----------------------------|--------------|-----------------------------------|
| ▪ <code>scanf(...)</code> | эквивалентен | <code>fscanf(stdin, ...)</code> |
| ▪ <code>printf(...)</code> | эквивалентен | <code>fprintf(stdout, ...)</code> |

stdio.h

По умолчанию печать сообщений в `stdout` и `stderr` приводит к их выводу на экран терминала

Сообщения об ошибках, диагностические сообщения следует печатать в `stderr`

Пример программы

Программа подсчета среднего арифметического значения элементов последовательности

Структура программы

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

```
int average(FILE *fi, double *avrp);
```

```
int main(void)
{
    ...
    s = average(fi, &avr);
    ...
}
```

```
int average(FILE *fi, double *avrp)
{
    ...
}
```

Функция обработки последовательности

Прототип:

```
int average(FILE *fi, double *avrp);
```

Входной параметр:

fi - указатель на открытый входной файл

Функция обработки последовательности

Прототип:

```
int average(FILE *fi, double *avrp);
```

Выходной параметр:

возвращаемый через оператор return

0 - успех

1 - ошибка: «мусор» во входном файле

2 - ошибка: пустой входной файл

Функция обработки последовательности

Прототип:

```
int average(FILE *fi, double *avrp);
```

Выходной параметр:

avrp - указатель, по которому будет записано
вычисленное среднее арифметическое значение

Функция main (начало)

```
int main(void)
{
    FILE *fi, *fo;
    double avr;
    int s;

    fi = fopen("input.txt", "r");
    if(!fi)
    {
        fprintf(stderr, "Can't open file 'input.txt'.\n");
        return -1;
    }

    s = average(fi, &avr);

    fclose(fi);
```

```
if(s)
{
    switch(s)
    {
        case 1:
            fprintf(stderr, "Wrong data in the input file.\n");
            break;

        case 2:
            fprintf(stderr, "The input file is empty.\n");
    }

    return -1;
}
```

Функция main (продолжение)

```
fo = fopen("output.txt", "w");  
if(!fo)  
{  
    fprintf(stderr, "Can't open file 'output.txt'.\n");  
    return -1;  
}  
  
fprintf(fo, "%f", avr);  
  
fclose(fo);  
  
return 0;  
}
```

Функция `main` (подробнее)

```
int main(void)
{
    FILE *fi, *fo;
    double avr;
    int s;

    fi = fopen("input.txt", "r");
    if(!fi)
    {
        fprintf(stderr, "Can't open file 'input.txt'.\n");
        return -1;
    }

    s = average(fi, &avr);

    fclose(fi);
```

```
    if(s)
    {
        switch(s)
        {
            case 1:
                fprintf(stderr, "Wrong data in the input file.\n");
                break;

            case 2:
                fprintf(stderr, "The input file is empty.\n");
        }

        return -1;
    }
```

Комментарий. Переменные `fi` и `fo` будут использоваться для хранения указателей на открытые файлы `input.txt` и `output.txt`.

Функция main (подробнее)

```
int main(void)
{
    FILE *fi, *fo;
    double avr;
    int s;

    fi = fopen("input.txt", "r");
    if(!fi)
    {
        fprintf(stderr, "Can't open file 'input.txt'.\n");
        return -1;
    }

    s = average(fi, &avr);

    fclose(fi);
```

```
if(s)
{
    switch(s)
    {
        case 1:
            fprintf(stderr, "Wrong data in the input file.\n");
            break;

        case 2:
            fprintf(stderr, "The input file is empty.\n");
    }

    return -1;
}
```

Комментарий. Переменная `avr` будет использоваться для хранения вычисленной характеристики последовательности, а переменная `s` для хранения статуса успешности обработки последовательности.

Функция main (подробнее)

```
int main(void)
{
    FILE *fi, *fo;
    double avr;
    int s;

    fi = fopen("input.txt", "r");
    if(!fi)
    {
        fprintf(stderr, "Can't open file 'input.txt'.\n");
        return -1;
    }

    s = average(fi, &avr);

    fclose(fi);
```

```
if(s)
{
    switch(s)
    {
        case 1:
            fprintf(stderr, "Wrong data in the input file.\n");
            break;

        case 2:
            fprintf(stderr, "The input file is empty.\n");
    }

    return -1;
}
```

Функция main (подробнее)

```
int main(void)
{
    FILE *fi, *fo;
    double avr;
    int s;

    fi = fopen("input.txt", "r");
    if(!fi)
    {
        fprintf(stderr, "Can't open file 'input.txt'.\n");
        return -1;
    }

    s = average(fi, &avr);

    fclose(fi);
```

```
if(s)
{
    switch(s)
    {
        case 1:
            fprintf(stderr, "Wrong data in the input file.\n");
            break;

        case 2:
            fprintf(stderr, "The input file is empty.\n");
    }

    return -1;
}
```

Комментарий. Проверка успешности открытия файла. В случае успеха `fopen` возвращает ненулевой указатель, а в случае ошибки - 0.

Функция `main` (подробнее)

```
int main(void)
{
    FILE *fi, *fo;
    double avr;
    int s;

    fi = fopen("input.txt", "r");
    if(!fi)
    {
        fprintf(stderr, "Can't open file 'input.txt'.\n");
        return -1;
    }

    s = average(fi, &avr);

    fclose(fi);
```

```
if(s)
{
    switch(s)
    {
        case 1:
            fprintf(stderr, "Wrong data in the input file.\n");
            break;

        case 2:
            fprintf(stderr, "The input file is empty.\n");
    }

    return -1;
}
```

Комментарий. В случае ошибки открытия файла печатается сообщение об ошибке в `stderr` и функция `main` завершается с кодом `-1`.

Функция main (подробнее)

```
int main(void)
{
    FILE *fi, *fo;
    double avr;
    int s;

    fi = fopen("input.txt", "r");
    if(!fi)
    {
        fprintf(stderr, "Can't open file 'input.txt'.\n");
        return -1;
    }

    s = average(fi, &avr);

    fclose(fi);
```

```
if(s)
{
    switch(s)
    {
        case 1:
            fprintf(stderr, "Wrong data in the input file.\n");
            break;

        case 2:
            fprintf(stderr, "The input file is empty.\n");
    }

    return -1;
}
```

Функция `main` (подробнее)

```
int main(void)
{
    FILE *fi, *fo;
    double avr;
    int s;

    fi = fopen("input.txt", "r");
    if(!fi)
    {
        fprintf(stderr, "Can't open file 'input.txt'.\n");
        return -1;
    }

    s = average(fi, &avr);

    fclose(fi);
```

```
if(s)
{
    switch(s)
    {
        case 1:
            fprintf(stderr, "Wrong data in the input file.\n");
            break;

        case 2:
            fprintf(stderr, "The input file is empty.\n");
    }

    return -1;
}
```

Комментарий. Заккрытие файла `input.txt`.

Функция main (подробнее)

```
int main(void)
{
    FILE *fi, *fo;
    double avr;
    int s;

    fi = fopen("input.txt", "r");
    if(!fi)
    {
        fprintf(stderr, "Can't open file 'input.txt'.\n");
        return -1;
    }

    s = average(fi, &avr);

    fclose(fi);
```

```
if(s)
{
    switch(s)
    {
        case 1:
            fprintf(stderr, "Wrong data in the input file.\n");
            break;

        case 2:
            fprintf(stderr, "The input file is empty.\n");
    }

    return -1;
}
```

Функция `main` (подробнее)

```
int main(void)
{
    FILE *fi, *fo;
    double avr;
    int s;

    fi = fopen("input.txt", "r");
    if(!fi)
    {
        fprintf(stderr, "Can't open file 'input.txt'.\n");
        return -1;
    }

    s = average(fi, &avr);

    fclose(fi);
```

```
if(s)
{
    switch(s)
    {
        case 1:
            fprintf(stderr, "Wrong data in the input file.\n");
            break;

        case 2:
            fprintf(stderr, "The input file is empty.\n");
    }

    return -1;
}
```

Комментарий. Если при обработке последовательности возникла ошибка, то в `stderr` печатается соответствующее сообщение, и функция `main` завершается с кодом `-1`.

Функция main (подробнее)

```
fo = fopen("output.txt", "w");
if(!fo)
{
    fprintf(stderr, "Can't open file 'output.txt'.\n");
    return -1;
}

fprintf(fo, "%f", avr);

fclose(fo);

return 0;
}
```

Комментарий. Открытие файла [output.txt](#) на запись.

Функция `main` (подробнее)

```
fo = fopen("output.txt", "w");
if(!fo)
{
    fprintf(stderr, "Can't open file 'output.txt'.\n");
    return -1;
}

fprintf(fo, "%f", avr);

fclose(fo);

return 0;
}
```

Комментарий. Проверка успешности открытия файла. В случае успеха `fopen` возвращает ненулевой указатель, а в случае ошибки - **0**.

Функция `main` (подробнее)

```
fo = fopen("output.txt", "w");  
if(!fo)  
{  
    fprintf(stderr, "Can't open file 'output.txt'.\n");  
    return -1;  
}  
  
fprintf(fo, "%f", avr);  
  
fclose(fo);  
  
return 0;  
}
```

Комментарий. В случае ошибки открытия файла печатается сообщение об ошибке в `stderr` и функция `main` завершается с кодом `-1`.

Функция `main` (подробнее)

```
fo = fopen("output.txt", "w");  
if(!fo)  
{  
    fprintf(stderr, "Can't open file 'output.txt'.\n");  
    return -1;  
}  
  
fprintf(fo, "%f", avr);  
  
fclose(fo);  
  
return 0;  
}
```

Комментарий. Печать вычисленной характеристики в файл [output.txt](#).

Функция `main` (подробнее)

```
fo = fopen("output.txt", "w");  
if(!fo)  
{  
    fprintf(stderr, "Can't open file 'output.txt'.\n");  
    return -1;  
}  
  
fprintf(fo, "%f", avr);  
  
fclose(fo);  
  
return 0;  
}
```

Комментарий. Заккрытие файла `output.txt`.

Функция `main` (подробнее)

```
fo = fopen("output.txt", "w");
if(!fo)
{
    fprintf(stderr, "Can't open file 'output.txt'.\n");
    return -1;
}

fprintf(fo, "%f", avr);

fclose(fo);

return 0;
}
```

Комментарий. Функция `main` завершается с кодом `0`.

Функция обработки последовательности

```
int average(FILE *fi, double *avrp)
{
    double lst, sum;
    int len;

    sum = 0.;
    len = 0;

    for(;;)
        if(fscanf(fi, "%lf", &lst) != 1)
            break;
        else
        {
            sum += lst;
            len ++;
        }
}
```

```
    if(!feof(fi))
        return 1;

    if(!len)
        return 2;

    *avrp = sum / len;
    return 0;
}
```

Функция обработки последовательности

```
int average(FILE *fi, double *avrp)
{
    double lst, sum;
    int len;

    sum = 0.;
    len = 0;

    for(;;)
        if(fscanf(fi, "%lf", &lst) != 1)
            break;
        else
        {
            sum += lst;
            len ++;
        }
}
```

```
    if(!feof(fi))
        return 1;

    if(!len)
        return 2;

    *avrp = sum / len;
    return 0;
}
```

Комментарий. Переменная `lst` используется для хранения последнего элемента последовательности, переменная `sum` - для хранения суммы элементов последовательности, переменная `len` - для хранения длины последовательности.

Функция обработки последовательности

```
int average(FILE *fi, double *avrp)
{
    double lst, sum;
    int len;

    sum = 0.;
    len = 0;

    for(;;)
        if(fscanf(fi, "%lf", &lst) != 1)
            break;
        else
        {
            sum += lst;
            len ++;
        }
}
```

```
    if(!feof(fi))
        return 1;

    if(!len)
        return 2;

    *avrp = sum / len;
    return 0;
}
```

Комментарий. Пустая последовательность имеет нулевую длину и сумму элементов. Значение последнего элемента пустой последовательности не определено.

Функция обработки последовательности

```
int average(FILE *fi, double *avrp)
{
    double lst, sum;
    int len;

    sum = 0.;
    len = 0;

    for(;;)
        if(fscanf(fi, "%lf", &lst) != 1)
            break;
        else
        {
            sum += lst;
            len ++;
        }
```

```
    if(!feof(fi))
        return 1;

    if(!len)
        return 2;

    *avrp = sum / len;
    return 0;
}
```

Функция обработки последовательности

```
int average(FILE *fi, double *avrp)
{
    double lst, sum;
    int len;

    sum = 0.;
    len = 0;

    for(;;)
        if(fscanf(fi, "%lf", &lst) != 1)
            break;
        else
        {
            sum += lst;
            len ++;
        }
```

```
    if(!feof(fi))
        return 1;

    if(!len)
        return 2;

    *avrp = sum / len;
    return 0;
}
```

Комментарий. Телом цикла является оператор **if-else**.

Функция обработки последовательности

```
int average(FILE *fi, double *avrp)
{
    double lst, sum;
    int len;

    sum = 0.;
    len = 0;

    for(;;)
        if(fscanf(fi, "%lf", &lst) != 1)
            break;
        else
        {
            sum += lst;
            len ++;
        }
}
```

```
    if(!feof(fi))
        return 1;

    if(!len)
        return 2;

    *avrp = sum / len;
    return 0;
}
```

Комментарий. На каждой итерации цикла осуществляется попытка считать очередной элемент последовательности и записать его в переменную `lst`.

Функция обработки последовательности

```
int average(FILE *fi, double *avrp)
{
    double lst, sum;
    int len;

    sum = 0.;
    len = 0;

    for(;;)
        if(fscanf(fi, "%lf", &lst) != 1)
            break;
        else
        {
            sum += lst;
            len ++;
        }
}
```

```
    if(!feof(fi))
        return 1;

    if(!len)
        return 2;

    *avrp = sum / len;
    return 0;
}
```

Комментарий. Если не удалось прочитать очередной элемент последовательности, то происходит выход из цикла.

Функция обработки последовательности

```
int average(FILE *fi, double *avrp)
{
    double lst, sum;
    int len;

    sum = 0.;
    len = 0;

    for(;;)
        if(fscanf(fi, "%lf", &lst) != 1)
            break;
        else
        {
            sum += lst;
            len ++;
        }
}
```

```
    if(!feof(fi))
        return 1;

    if(!len)
        return 2;

    *avrp = sum / len;
    return 0;
}
```

Комментарий. Иначе обновляются характеристики последовательности.

Функция обработки последовательности

```
int average(FILE *fi, double *avrp)
{
    double lst, sum;
    int len;

    sum = 0.;
    len = 0;

    for(;;)
        if(fscanf(fi, "%lf", &lst) != 1)
            break;
        else
        {
            sum += lst;
            len ++;
        }
}
```

```
if(!feof(fi))
    return 1;

if(!len)
    return 2;

*avrp = sum / len;
return 0;
}
```

Комментарий. Если не был достигнут конец файла, то это означает, что файл содержит «мусор». Функция завершается с кодом ошибки **1**.

Функция обработки последовательности

```
int average(FILE *fi, double *avrp)
{
    double lst, sum;
    int len;

    sum = 0.;
    len = 0;

    for(;;)
        if(fscanf(fi, "%lf", &lst) != 1)
            break;
        else
        {
            sum += lst;
            len ++;
        }
}
```

```
    if(!feof(fi))
        return 1;

    if(!len)
        return 2;

    *avrp = sum / len;
    return 0;
}
```

Комментарий. В случае пустой последовательности функция завершается с кодом ошибки 2.

Функция обработки последовательности

```
int average(FILE *fi, double *avrp)
{
    double lst, sum;
    int len;

    sum = 0.;
    len = 0;

    for(;;)
        if(fscanf(fi, "%lf", &lst) != 1)
            break;
        else
        {
            sum += lst;
            len ++;
        }
}
```

```
    if(!feof(fi))
        return 1;

    if(!len)
        return 2;

    *avrp = sum / len;
    return 0;
}
```

Комментарий. Последовательность была успешно обработана. Вычисленная характеристика последовательности записывается по адресу, переданному через входной параметр `avrp`. Функция завершается с успешным статусом `0`.