

Программирование на языке Си Часть II

1. [Массивы](#)
2. [Максимальный элемент массива](#)
3. [Обработка массивов](#)
4. [Сортировка массивов](#)
5. [Поиск в массиве](#)
6. [Массивы в процедурах и функциях](#)
7. [Практикум \(моделирование\)](#)
8. [Символьные строки](#)
9. [Рекурсивный перебор](#)
10. [Матрицы](#)
11. [Файлы](#)

Программирование на языке Си Часть II

Тема 1. Массивы

Массивы

Массив – это группа однотипных элементов, имеющих общее имя и расположенных в памяти рядом.

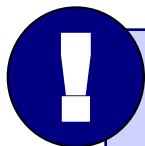
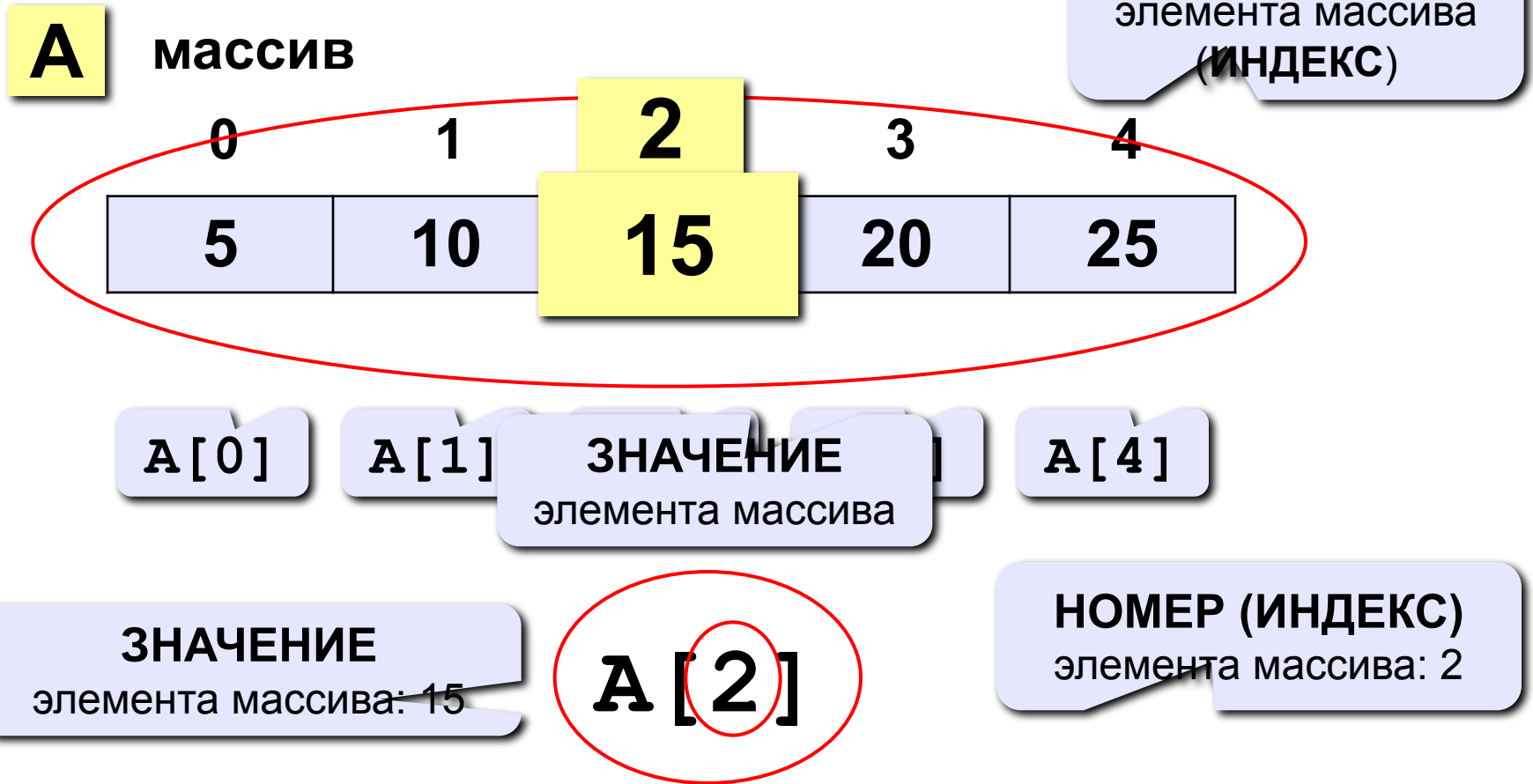
Особенности:

- все элементы имеют **один тип**
- весь массив имеет **одно имя**
- все элементы расположены в памяти **рядом**

Примеры:

- список учеников в классе
- квартиры в доме
- школы в городе
- данные о температуре воздуха за год

Массивы



Нумерация элементов массива в Си начинается
с **НУЛЯ!**

Объявление массивов

Зачем объявлять?

- определить **ИМЯ** массива
- определить **ТИП** массива
- определить **ЧИСЛО ЭЛЕМЕНТОВ**
- выделить **МЕСТО В ПАМЯТИ**

Пример:

ТИП
элементов

ИМЯ

размер массива
(количество
элементов)

```
int A [ 5 ] ;
```

Размер через константу:

```
const int N =  
5;  
int A [ N ] ;
```

Объявление массивов

Еще примеры:

```
int X[10], Y[10];  
float zz, A[20];  
char s[80];
```

С присвоением начальных значений:

```
int A[4] = { 8, -3, 4, 6 };  
float B[2] = { 1. };  
char C[3] = { 'A', '1', 'Ю' };
```

остальные
нулевые!



Если начальные значения не заданы, в ячейках находится «мусор»!

Что неправильно?

```
const int N = 10;
float A[N];
```

```
int X[4.5];
```

```
int A[10];
A[10] = 0;
```

**ВЫХОД ЗА ГРАНИЦЫ
МАССИВА**
(стираются данные
в памяти)

```
float X[5];
int n = 1;
X[n-2] = 4.5;
X[n+8] = 12.;
```

дробная часть
отбрасывается
(ошибки нет)

```
int X[4];
X[2] = 4.5;
```

```
float A[2] = { 1, 3.8 };
```

```
float B[2] = { 1., 3.8, 5.5 };
```

Массивы

Объявление:

```
const int N = 5;
int A[N], i;
```

Ввод с клавиатуры:

```
printf("Введите 5 элементов массива:\n");
for( i=0; i < N; i++ ) {
    printf ("A[%d] = ", i );
    scanf ("%d", &A[i] );
}
```

A[0] = 5
 A[1] = 12
 A[2] = 34
 A[3] = 56
 A[4] = 13

По

Вь

```
for( i=0; i < N; i++ ) A[i] = A[i]*2;
```

```
printf("Результат:\n");
for( i=0; i < N; i++ )
    printf("%4d", A[i]);
```

Результат:

10 24 68 112 26

Программа

Задача: ввести с клавиатуры массив из 5 элементов, умножить все элементы на 2 и вывести полученный массив на экран.

```
#include <stdio.h>
#include <conio.h>
main()
{
    const int N = 5;
    int A[N], i;
    // ввод элементов массива
    // обработка массива
    // вывод результата
    getch();
}
```

на предыдущих
слайдах

Задания

«4»: Ввести с клавиатуры массив из 5 элементов, найти среднее арифметическое всех элементов массива.

Пример:

Введите пять чисел:

4 15 3 10 14

среднее арифметическое 9.200

«5»: Ввести с клавиатуры массив из 5 элементов, найти минимальный из них.

Пример:

Введите пять чисел:

4 15 3 10 14

минимальный элемент 3



При изменении константы N остальная программа не должна изменяться!

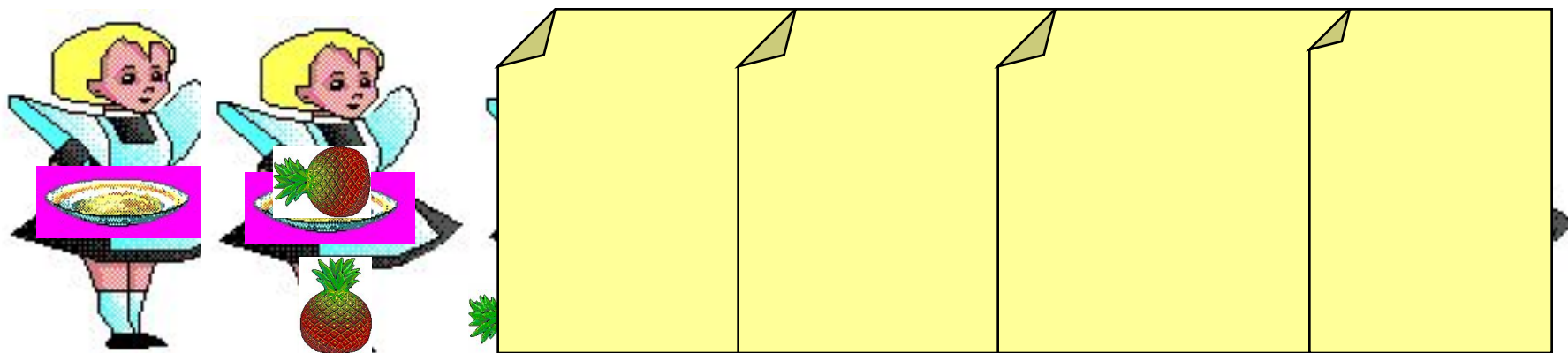
Программирование на языке Си Часть II

Тема 2. Максимальный элемент массива

Максимальный элемент

Задача: найти в массиве максимальный элемент.

Алгоритм:



Псевдокод:

```
// считаем, что элемент A[0] – максимальный
for ( i=1; i < N; i++ )
    if ( A[i] > максимального )
        // запомнить новый максимальный элемент A[i]
```



Почему цикл от $i=1$?

Максимальный элемент

Дополнение: как найти номер максимального элемента?

```

// пока A[0] – максимальный
iMax = 0;
for ( i=1; i < N; i++ ) // проверяем остальные
    if ( A[i] > A[iMax] ) { // нашли новый
        // запомнить A[i]
        iMax = i; // запомнить i
    }
```



Как упростить?

По номеру элемента **iMax** всегда можно найти его значение **A[iMax]**. Поэтому везде меняем **max** на **A[iMax]** и убираем переменную **max**.

Заполнение случайными числами

```
#include <stdlib.h> // случайные числа
```

RAND_MAX – максимальное случайное целое число
(обычно `RAND_MAX = 32767`)

Случайное целое число в интервале `[0,RAND_MAX]`

```
x = rand(); // первое число
```

```
x = rand(); // уже другое число
```

Установить начальное значение последовательности:

```
srand ( 345 ); // начнем с 345
```

Целые числа в заданном интервале

Целые числа в интервале $[0, N-1]$:

```
int random(int N) {  
    return rand() % N;  
}
```

Примеры:

```
x = random ( 100 ) ;    // интервал [0, 99]  
x = random ( z ) ;     // интервал [0, z-1]
```

Целые числа в интервале $[a, b]$:

```
x = random ( z ) + a ;    // интервал [a, z-1+a]  
x = random ( b - a + 1 ) + a ; // интервал [a, b]
```

Заполнение случайными числами

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

int random(int N)
{ return rand() % N; }

main()
{
  const int N = 10;
  int A[N], i;
  printf("Исходный массив:\n");
  for (i = 0; i < N; i++) {
    A[i] = random(100) + 50;
    printf("%4d", A[i]);
  }
  ...
}
```

функция выдает
случайное число
от 0 до N-1



Какой интервал?

Программа

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

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

```
main ()
```



Что дает `const`?

```
{
```

```
const int N = 5;
```

```
int A[N], i, iMax;
```

```
    // заполнить случайными числами [100,150]
```

```
    // найти максимальный элемент и его номер
```

```
printf("\nМаксимальный элемент A[%d] = %d",  
        iMax, A[iMax]);
```

```
getch();
```

```
}
```

на предыдущих
слайдах

Задания

«4»: Заполнить массив из 10 элементов случайными числами в интервале $[-10..10]$ и найти в нем максимальный и минимальный элементы и их номера.

Пример:

Исходный массив:

4 -5 3 10 -4 -6 8 -10 1 0

максимальный $a[4]=10$

минимальный $a[8]=-10$

«5»: Заполнить массив из 10 элементов случайными числами в интервале $[-10..10]$ и найти в нем два максимальных элемента и их номера.

Пример:

Исходный массив:

4 -5 3 10 -4 -6 8 -10 1 0

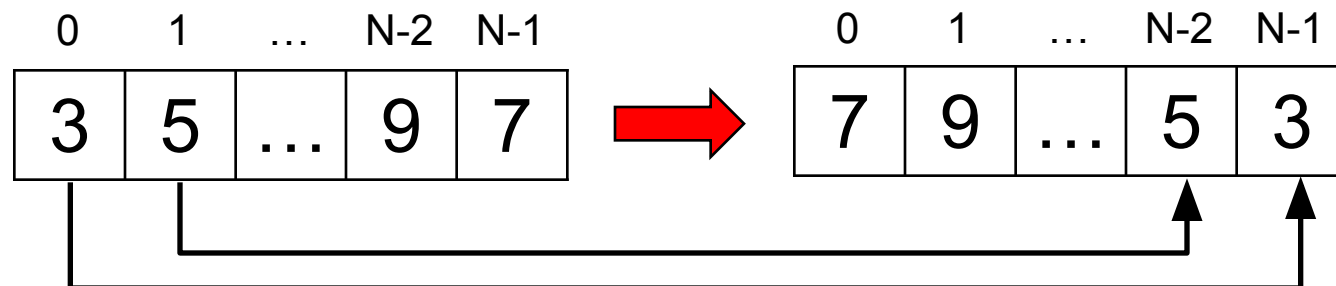
максимальные $a[4]=10, a[7]=8$

Программирование на языке Си Часть II

Тема 3. Обработка массивов

Реверс массива

Задача: переставить элементы массива в обратном порядке (выполнить инверсию).



Алгоритм:

поменять местами $A[0]$ и $A[N-1]$, $A[1]$ и $A[N-2]$, ...

Псевдокод:

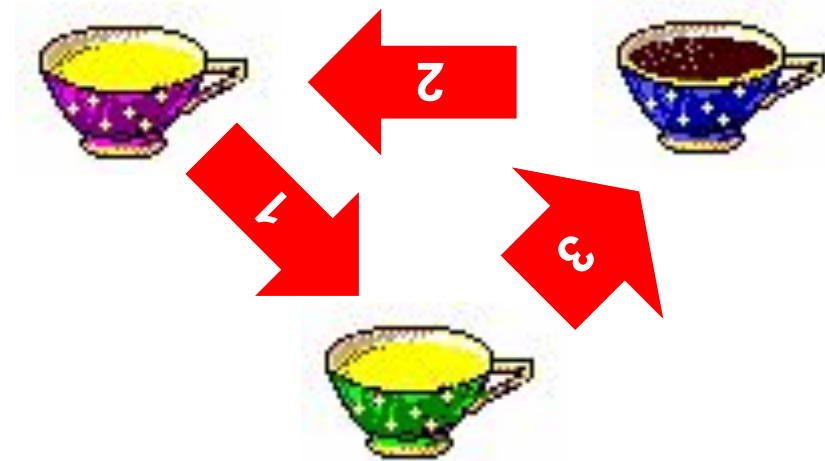
```
for ( i = 0; i < N / 2 ; i++ )
  // поменять местами A[i] и A[N-1-i]
```



Что неверно?

Как переставить элементы?

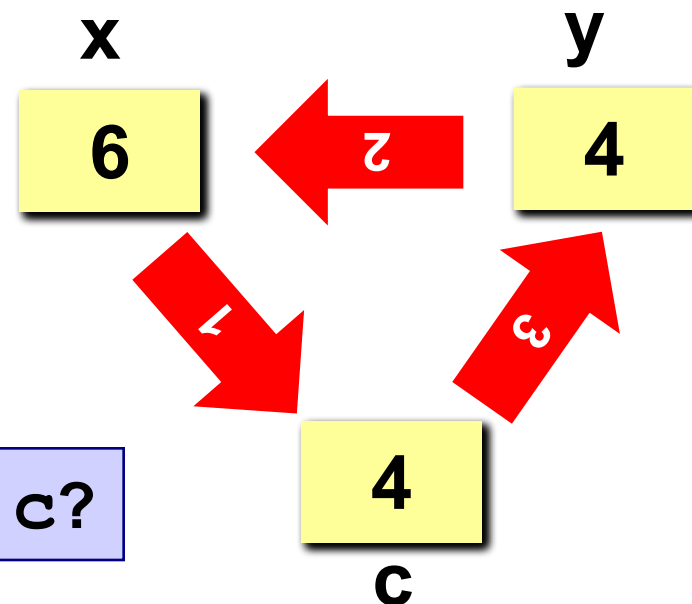
Задача: поменять местами содержимое двух чашек.



Задача: поменять местами содержимое двух ячеек памяти.

~~$x = y;$
 $y = x;$~~

$c = x;$
 $x = y;$
 $y = c;$



Можно ли обойтись без c ?

Программа

```
main ()
{
    const int N=10;
    int A[N], i, c;
    // заполнить массив
    // вывести исходный массив
    for ( i=0; i<N/2; i++) {
        c=A[i];
        A[i]=A[N-1-i];
        A[N-1-i]=c;
    }
    // вывести полученный массив
}
```

Задания

«4»: Заполнить массив из 10 элементов случайными числами в интервале $[-10..10]$ и выполнить инверсию отдельно для 1-ой и 2-ой половин массива.

Пример:

Исходный массив:

4	-5	3	10	-4	-6	8	-10	1	0
---	----	---	----	----	----	---	-----	---	---

Результат:

-4	10	3	-5	4	0	1	-10	8	-6
----	----	---	----	---	---	---	-----	---	----

«5»: Заполнить массив из 12 элементов случайными числами в интервале $[-12..12]$ и выполнить инверсию для каждой трети массива.

Пример:

Исходный массив:

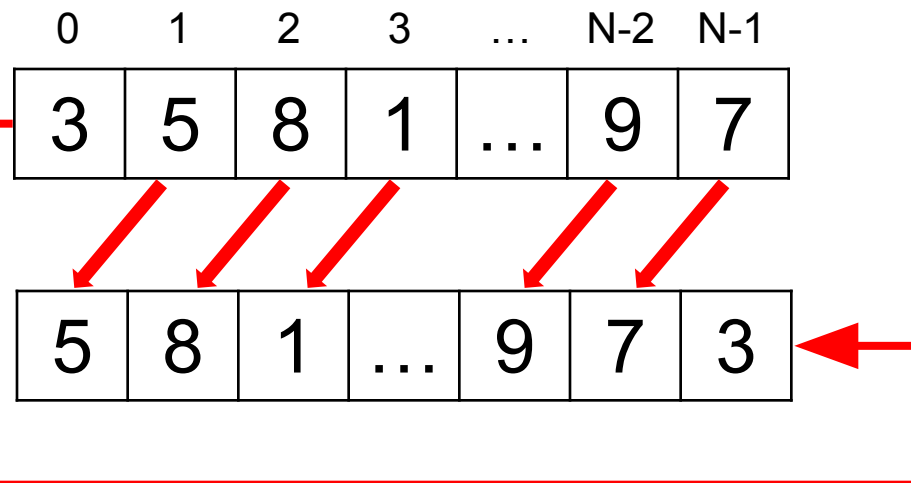
4	-5	3	10	-4	-6	8	-10	1	0	5	7
---	----	---	----	----	----	---	-----	---	---	---	---

Результат:

10	3	-5	4	-10	8	-6	-4	7	5	0	1
----	---	----	---	-----	---	----	----	---	---	---	---

Циклический сдвиг

Задача: сдвинуть элементы массива влево на 1 ячейку, первый элемент становится на место последнего.



Алгоритм:

$A[0]=A[1] ; A[1]=A[2] ; \dots A[N-2]=A[N-1] ;$

Цикл:

почему не N ?

```
for ( i = 0; i < N-1; i++)
    A[i] = A[i+1];
```



Что неверно?

Программа

```
main()  
{  
    const int N = 10;  
    int A[N], i, c;  
    // заполнить массив  
    // вывести исходный массив  
    c = A[0];  
    for ( i = 0; i < N-1; i ++ )  
        A[i] = A[i+1];  
    A[N-1] = c;  
    // вывести полученный массив  
}
```

Задания

«4»: Заполнить массив из 10 элементов случайными числами в интервале $[-10..10]$ и выполнить циклический сдвиг ВПРАВО.

Пример:

Исходный массив:

4 -5 3 10 -4 -6 8 -10 1 0

Результат:

0 4 -5 3 10 -4 -6 8 -10 1

«5»: Заполнить массив из 12 элементов случайными числами в интервале $[-12..12]$ и выполнить циклический сдвиг ВПРАВО на 4 элемента.

Пример:

Исходный массив:

4 -5 3 10 | -4 -6 8 -10 1 0 5 7

Результат:

1 0 5 7 4 -5 3 10 | -4 -6 8 -10

Программирование на языке Си Часть II

Тема 4. Сортировка массивов

Сортировка

Сортировка – это расстановка элементов массива в заданном порядке (по возрастанию, убыванию, последней цифре, сумме делителей, ...).

Задача: переставить элементы массива в порядке возрастания.

Алгоритмы:

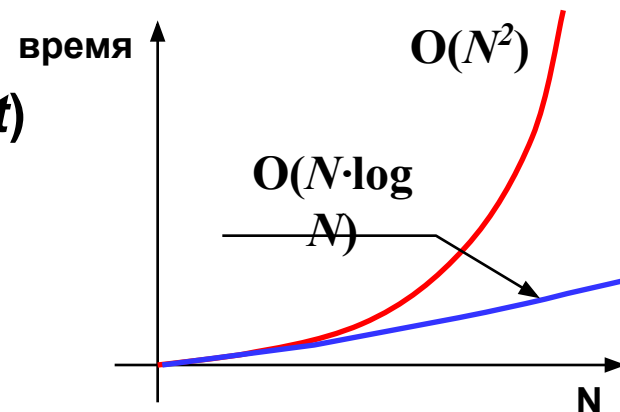
сложность $O(N^2)$

- простые и понятные, но неэффективные для больших массивов

- метод пузырька
- метод выбора

сложность $O(N \cdot \log N)$

- сложные, но эффективные
 - «быстрая сортировка» (*Quick Sort*)
 - сортировка «кучей» (*Heap Sort*)
 - сортировка слиянием
 - пирамидальная сортировка



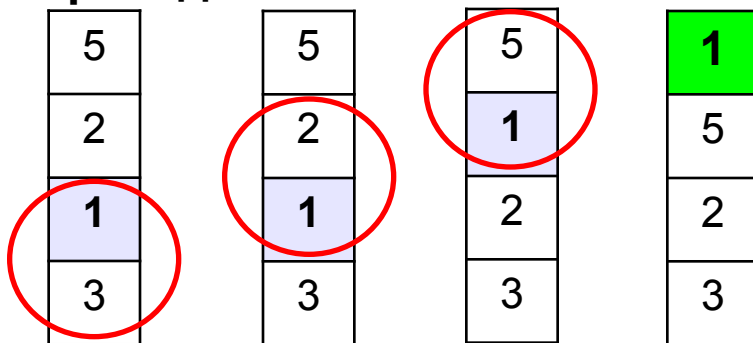
Метод пузырька

Идея – пузырек воздуха в стакане воды поднимается со дна вверх.

Для массивов – самый маленький («легкий») элемент перемещается вверх («всплывает»).

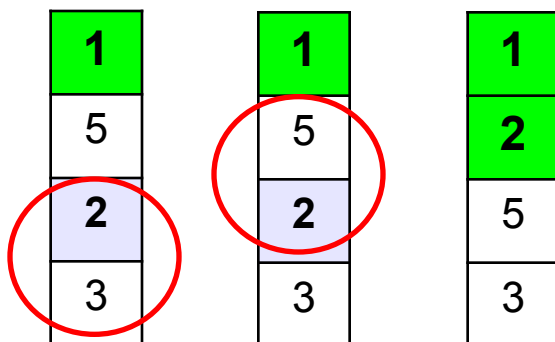
1-ый

проход

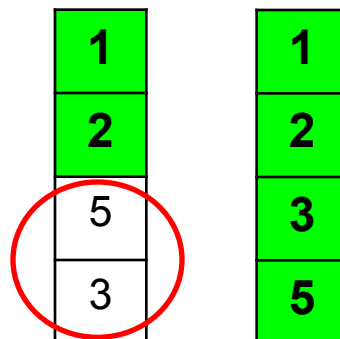


- начиная снизу, сравниваем два соседних элемента; если они стоят «неправильно», меняем их местами
- за 1 проход по массиву **один** элемент (самый маленький) становится на свое место

2-ой проход



3-ий проход



Для сортировки массива из N элементов нужен $N-1$ проход (достаточно поставить на свои места $N-1$ элементов).

Программа (1-ый проход)

0	5
1	2
...	...
N-2	6
N-1	3

сравниваются пары

$A[N-2]$ и $A[N-1]$,

$A[N-3]$ и $A[N-2]$

...

$A[0]$ и $A[1]$

$A[j]$ и $A[j+1]$

```

for ( j = N-2; j >= 0; j-- )
    if ( A[j] > A[j+1] ) {
        c = A[j];
        A[j] = A[j+1];
        A[j+1] = c;
    }

```

Программа (следующие проходы)

2-ой
проход

0	1
1	5
...	...
N-2	3
N-1	6

(i+1)-ый
проход



A[0] уже на своем месте!

```
for ( j = N-2; j >= 1; j-- )
    if ( A[j] > A[j+1] ) {
        c = A[j];
        A[j] = A[j+1];
        A[j+1] = c;
    }
```

```
for ( j = N-2; j >= i; j-- )
    ...
```

Программа

```

main ()
{
  const int N = 10;
  int A[N], i, j, c;
  // заполнить массив
  // вывести исходный массив
  for (i = 0; i < N-1; i++) {
    for (j = N-2; j >= i; j--)
      if (A[j] > A[j+1]) {
        c = A[j];
        A[j] = A[j+1];
        A[j+1] = c;
      }
  }
  // вывести полученный массив
}

```



Почему цикл для $i < N-1$,
а не $i < N$?

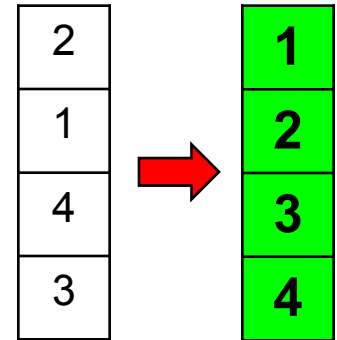
элементы выше
 $A[i]$ уже
поставлены

меняем $A[j]$
и $A[j+1]$

Метод пузырька с флажком

Идея – если при выполнении метода пузырька не было обменов, массив уже отсортирован и остальные проходы не нужны.

Реализация: переменная-флаг, показывающая, был ли обмен; если она равна **0**, то выход.



```

do {
    int flag;
    flag = 0; // сбросить флаг
    for (j = N-2; j >= 0; j --)
        if (A[j] > A[j+1]) {
            c = A[j];
            A[j] = A[j+1];
            A[j+1] = c;
            flag = 1; // поднять флаг
        }
    }
while (flag != 0); // выход при flag = 0
  
```

? Как улучшить?

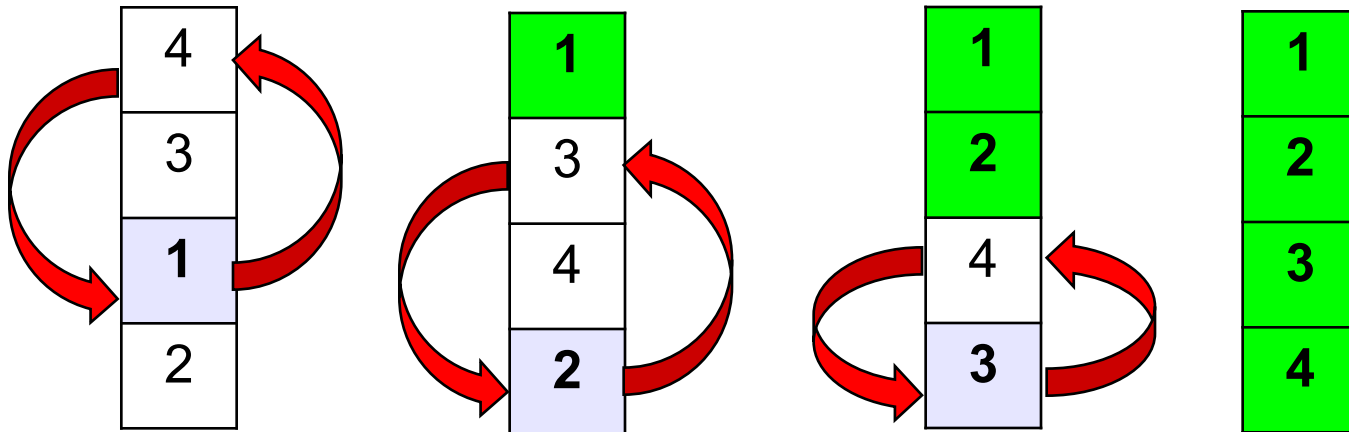
Метод пузырька с флажком

```
i = 0;
do {
    flag = 0; // сбросить флаг
    for ( j = N-2; j >= i; j -- )
        if ( A[j] > A[j+1] ) {
            c = A[j];
            A[j] = A[j+1];
            A[j+1] = c;
            flag = 1; // поднять флаг
        }
    i ++;
}
while ( flag ); // выход при flag = 0
```

Метод выбора

Идея:

- найти минимальный элемент и поставить на первое место (поменять местами с $A[0]$)
- **из оставшихся** найти минимальный элемент и поставить на второе место (поменять местами с $A[1]$), и т.д.



Метод выбора

нужно $N-1$ проходов

```

for ( i = 0; i <  $N-1$ ; i++ ) {
    nMin = i;
    for ( j =  $i+1$ ; j < N; j++ )
        if ( A[j] < A[nMin] ) nMin = j;
    if ( nMin != i ) {
        c = A[i];
        A[i] = A[nMin];
        A[nMin] = c;
    }
}

```

ПОИСК МИНИМАЛЬНОГО
ОТ $A[i]$ ДО $A[N-1]$

если нужно,
переставляем



Можно ли убрать if?

Задания

«4»: Заполнить массив из 10 элементов случайными числами в интервале [0..100] и отсортировать его по последней цифре.

Пример:

Исходный массив :

14 25 13 30 76 58 32 11 41 97

Результат :

30 11 41 32 13 14 25 76 97 58

«5»: Заполнить массив из 10 элементов случайными числами в интервале [0..100] и отсортировать первую половину по возрастанию, а вторую – по убыванию.

Пример:

Исходный массив :

14 25 13 30 76 | 58 32 11 41 97

Результат :

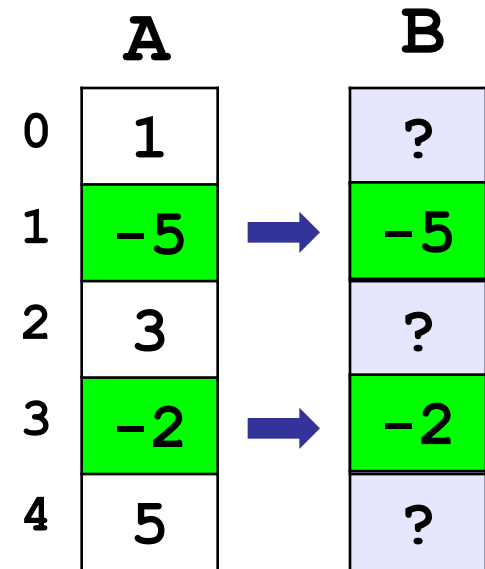
13 14 25 30 76 | 97 58 41 32 11

Формирование массива по условию

Задача – найти в массиве элементы, удовлетворяющие некоторому условию (например, отрицательные), и скопировать их в другой массив.

Примитивное решение:

```
const int N = 5;
int A[N], B[N];
// здесь заполнить массив A
for( i=0; i < N; i++ )
    if( A[i] < 0 ) B[i] = A[i];
```

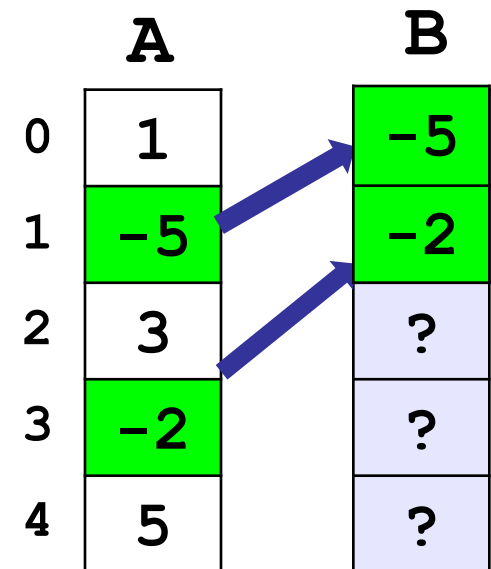


- выбранные элементы не рядом, не в начале массива
- непонятно, как с ними работать

Формирование массива по условию

Решение: ввести счетчик найденных элементов `count`, очередной элемент ставится на место `B[count]`.

```
int A[N], B[N], count = 0;
// здесь заполнить массив A
for( i=0; i < N; i++)
    if( A[i] < 0 ) {
        B[count] = A[i];
        count++;
    }
// вывод массива B
for( i=0; i < count; i++)
    printf("%d\n", B[i]);
```



Задания

«4»: Заполнить массив случайными числами и отобразить в другой массив все числа, у которых вторая с конца цифра (число десятков) – ноль.

Пример:

Исходный массив:

40 105 203 1 14

Результат:

105 203 1

«5»: Заполнить массив случайными числами и выделить в другой массив все числа, которые встречаются более одного раза.

Пример:

Исходный массив:

4 1 2 1 11 2 34

Результат:

1 2

Программирование на языке Си Часть II

Тема 5. Поиск в массиве

Поиск в массиве

Задача – найти в массиве элемент, равный **X**, или установить, что его нет.

Решение: для произвольного массива: **линейный поиск** (перебор)

недостаток: **низкая скорость**

Как ускорить? – заранее подготовить массив для поиска

- как именно подготовить?
- как использовать «подготовленный» массив?

Линейный поиск

nX – номер нужного элемента в массиве

```
nX = -1; // пока не нашли ...
for ( i = 0; i < N; i ++ ) // цикл по всем элементам
    if ( A[i] == X ) // если нашли, то ...
        nX = i; // ... запомнили номер

if ( nX < 0 ) printf("Не нашли...")
else          printf("A[%d]=%d", nX, X);
```

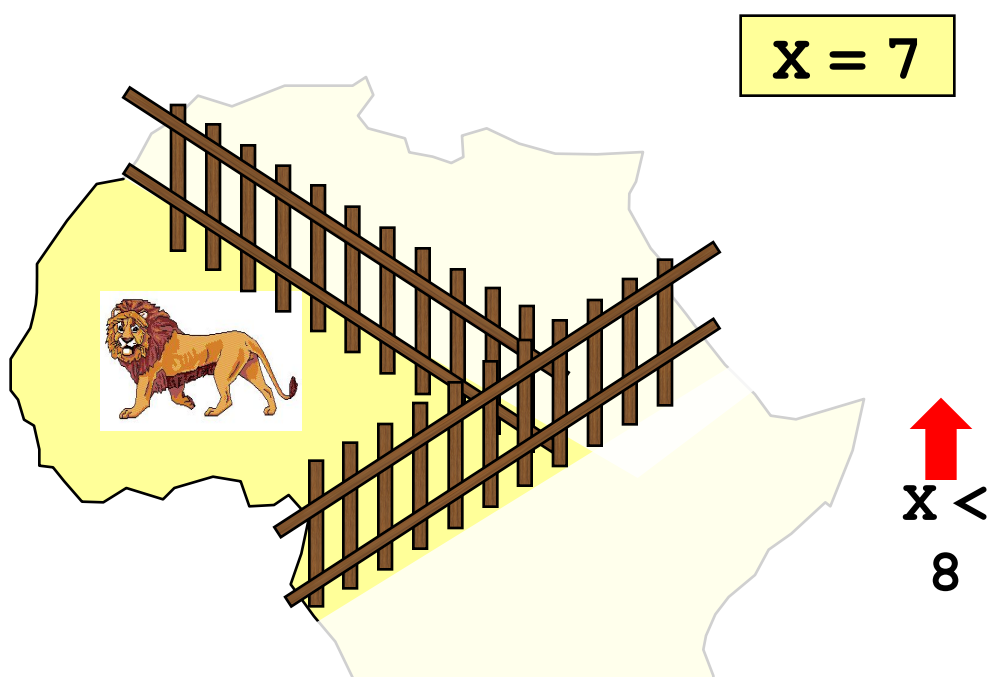


Что можно улучшить?

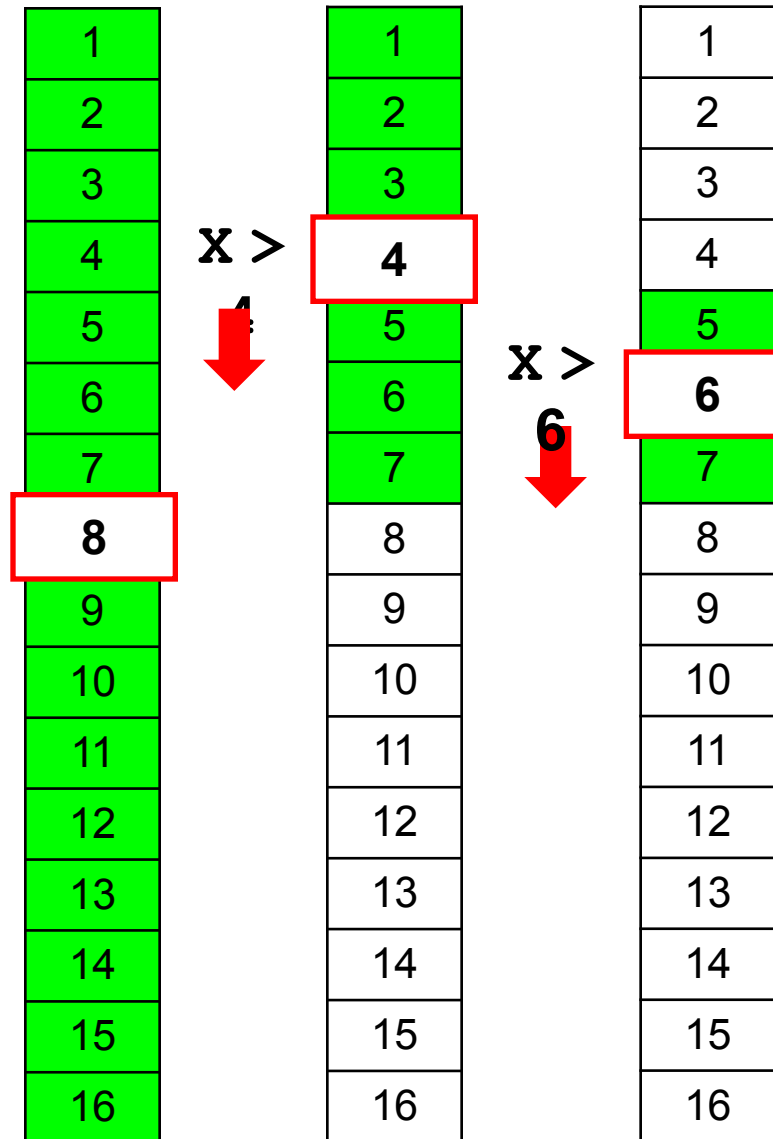
Улучшение: после того, как нашли X , выходим из цикла.

```
nX = -1;
for ( i = 0; i < N; i ++ )
    if ( A[i] == X ) {
        nX = i;
        break // выход из цикла
    }
```

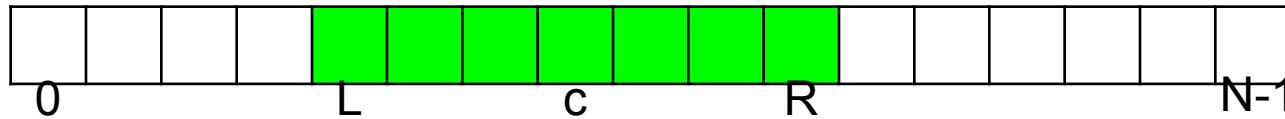
Двоичный поиск



1. Выбрать средний элемент $A[s]$ и сравнить с X .
2. Если $X = A[s]$, нашли (выход).
3. Если $X < A[s]$, искать дальше в первой половине.
4. Если $X > A[s]$, искать дальше во второй половине.



Двоичный поиск



```

nX = -1;
L = 0; R = N-1; // границы: ищем от A[0] до A[N-1]
while ( R >= L ){
    c = (R + L) / 2;
    if (X == A[c]) {
        nX = c;
        break;
    }
    if (X < A[c]) R = c - 1;
    if (X > A[c]) L = c + 1;
}
if (nX < 0) printf("Не нашли...");
else      printf("A[%d]=%d", nX, X);

```

номер среднего элемента

если нашли ...

ВЫЙТИ ИЗ ЦИКЛА

сдвигаем
границы



Почему нельзя `while (R > L) { ... } ?`

Сравнение методов поиска

	Линейный	Двоичный
подготовка	нет	отсортировать
	число шагов	
N = 2	2	2
N = 16	16	5
N = 1024	1024	11
N = 1048576	1048576	21
N	$\leq N$	$\leq \log_2 N + 1$

Задания

- «4»: Написать программу, которая сортирует массив **ПО УБЫВАНИЮ** и ищет в нем элемент, равный X (это число вводится с клавиатуры). Использовать **двоичный поиск**.
- «5»: Написать программу, которая считает **среднее число шагов в двоичном поиске** для массива из 32 элементов в интервале $[0, 100]$. Для поиска использовать 1000 случайных чисел в этом же интервале.

Программирование на языке Си Часть II

Тема 6. Массивы в процедурах и функциях

Массивы в процедурах

Задача: составить процедуру, которая переставляет элементы массива в обратном порядке.

параметр-
массив

размер
массива

```
void Reverse ( int A[] , int N )
{
  int i, c;
  for ( i = 0; i < N/2; i ++ ) {
    c = A[i];
    A[i] = A[N-1-i];
    A[N-1-i] = c;
  }
}
```

Массивы как параметры процедур

Особенности:

- при описании параметра-массива в заголовке функции его размер не указывается (функция работает с массивами **любого размера**)



Почему здесь размер не обязателен?

- размер массива надо передавать как отдельный параметр
- в процедура передается **адрес** исходного массива: все **изменения**, сделанные в процедуре **влияют** на массив в основной программе

Массивы в процедурах

```
void Reverse ( int A[], int N )
```

```
{
```

```
...
```

```
}
```

```
main()
```

```
{
```

```
int A[10];
```

A или &A[0]

```
// здесь надо заполнить массив
```

```
Reverse ( A, 10 ); // весь массив
```

```
// Reverse ( A, 5 ); // первая половина
```

```
// Reverse ( A+5, 5 ); // вторая половина
```

```
}
```

A+5 или &A[5]

Задания

- «4»: Написать процедуру, которая сортирует массив по возрастанию, и показать пример ее использования.
- «5»: Написать процедуру, которая ставит в начало массива все четные элементы, а конец – все нечетные.

Массивы в функциях

Задача: составить функцию, которая находит сумму элементов массива.

результат –
целое число

параметр-
массив

размер
массива

```
int Sum ( int A[] , int N )  
{  
    int i, sum = 0;  
    for ( i = 0; i < N; i ++ )  
        sum += A[i];  
    return sum;  
}
```

Массивы в процедурах и функциях

```
int Sum ( int A[], int N )
{
    ...
}
main()
{
    int A[10], sum, sum1, sum2;
    // заполнить массив
    sum = Sum ( A, 10 ); // весь массив
    sum1 = Sum ( A, 5 ); // первая половина
    sum2 = Sum ( A+5, 5 ); // вторая половина
    ...
}
```

Задания

«4»: Написать функцию, которая находит максимальный элемент в массиве.

«5»: Написать логическую функцию, которая определяет, верно ли, что среди элементов массива есть два одинаковых. Если ответ «да», функция возвращает 1; если ответ «нет», то 0.

Подсказка: для отладки удобно использовать массив из 5 элементов, задаваемых вручную:

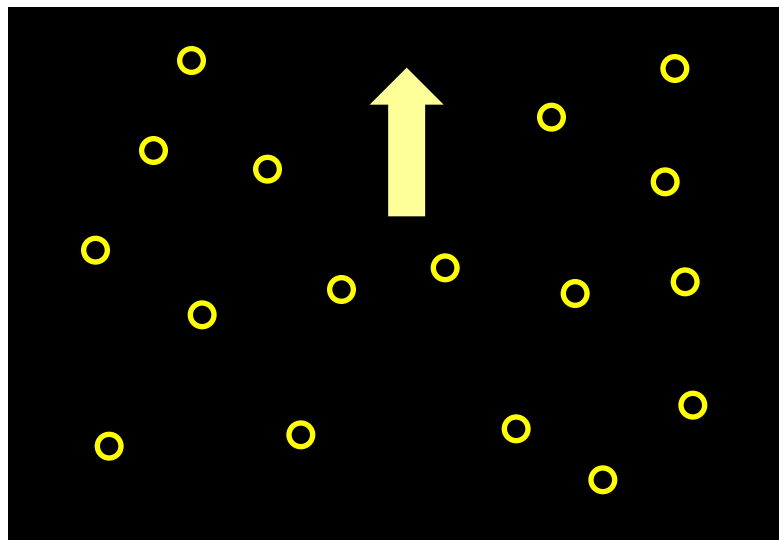
```
const int N = 5;  
int A[N] = { 1, 2, 3, 3, 4 };
```

Программирование на языке Си Часть II

Тема 7. Практикум (моделирование)

Моделирование кипения воды

Задача: Построить компьютерную модель кипения воды.



Хранение данных: координаты (центров) пузырьков хранятся в массивах **X** и **Y**:

X[i], Y[i] – координаты центра пузырька с номером **i**.

Структура программы

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

глобальные
константы и
переменные

```
const int N = 100;
int X[N], Y[N], r = 3;
```

объявления
процедур

```
void Init (); // начальное положение
void Draw ( int color ); // рисуем, стираем
void Sdvig ( int dy ); // летят вверх
void Zamena (); // ушли, пришли
```

```
main()
```

```
{
```

```
initwindow (600, 400);
... // основная часть программы
closegraph();
```

```
}
```

```
... // здесь сами процедуры
```

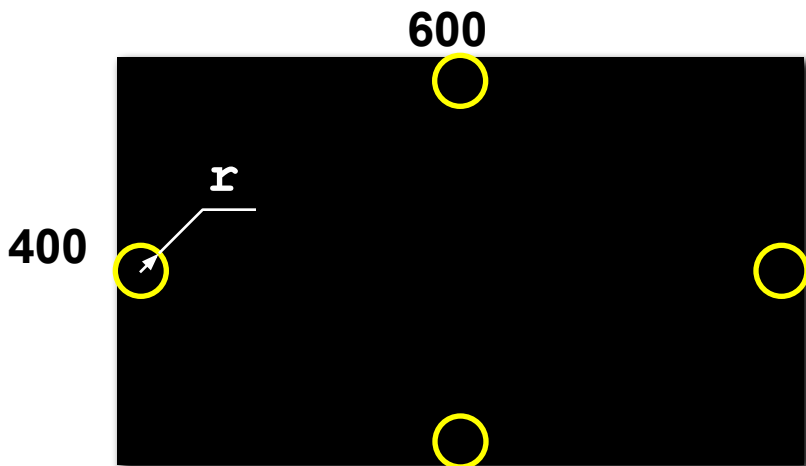
Основная программа

```
Init();           // начальная расстановка
while ( 1 )      // зацикливание ???
{
  if ( kbhit() )
    if ( getch() == 27 ) break;
  Draw ( YELLOW ); // рисуем все пузырьки
  delay ( 10 );    // ждем 10 мс
  Draw ( BLACK );  // стираем все пузырьки
  Sdvig ( 4 );     // вверх на 4 пикселя
  Zamena ();      // если за пределами экрана...
}
```

ВЫХОД ПО
Esc (код 27)

Процедура Init

Начальная случайная расстановка:



Интервал для x : $[r, 600-r]$

$X[i] = \text{random}(640 - 2*r) + r;$

Интервал для y : $[r, 400-r]$

$Y[i] = \text{random}(400 - 2*r) + r;$

```
void Init()
{
    int i;
    for ( i = 0; i < N; i ++ ) {
        X[i] = random(600 - 2*r) + r;
        Y[i] = random(400 - 2*r) + r;
    }
}
```

Процедуры Draw, Sdvig

Рисование и стирание:

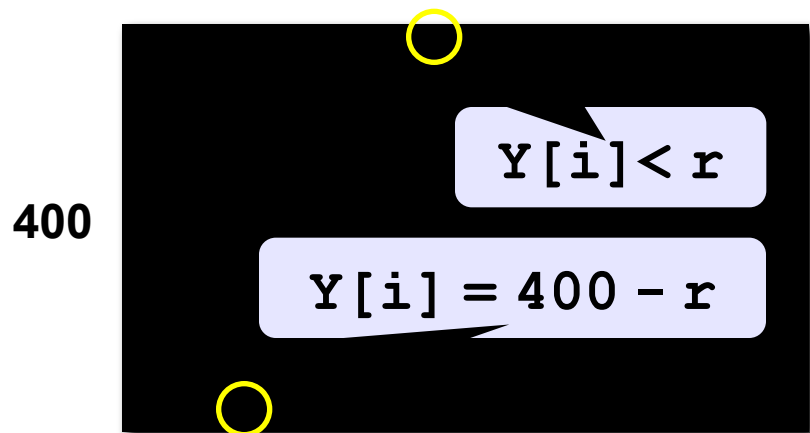
```
void Draw ( int color )
{
    int i;
    setcolor ( color );
    for ( i = 0; i < N; i ++ )
        circle ( X[i], Y[i], r );
}
```

Сдвиг вверх:

```
void Sdvig ( int dy )
{
    int i;
    for ( i = 0; i < N; i ++ )
        Y[i] -= dy;
}
```

Процедура Замена

Замена вышедших за границы экрана:



Условие выхода:

```
if ( Y[i] < r ) { ... }
```

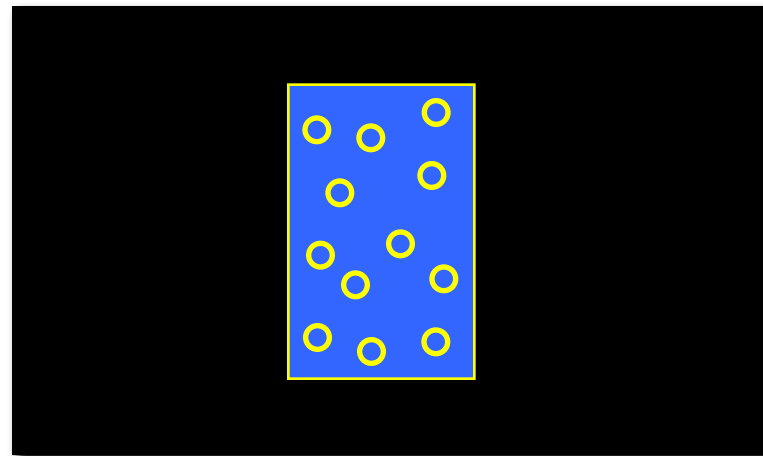
Перебросить вниз:

```
X[i] = random(600 - 2*r) + r;
Y[i] = 400 - r;
```

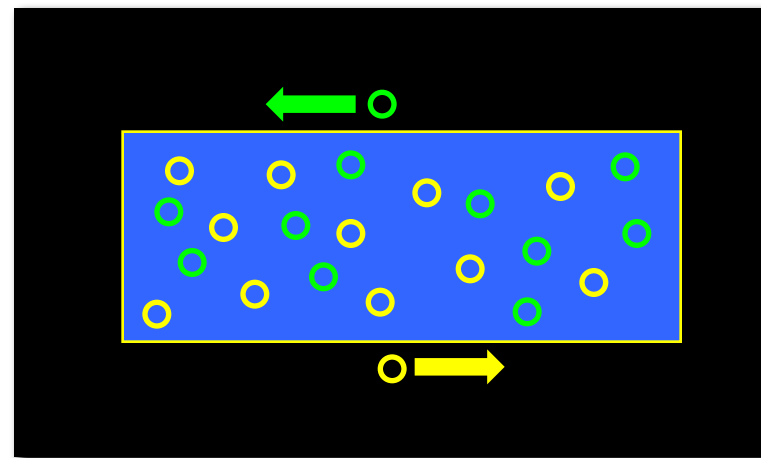
```
void Zamena ()
{
    int i;
    for ( i = 0; i < N; i ++ )
        if ( Y[i] < r ) {
            X[i] = random(600 - 2*r) + r;
            Y[i] = 400 - r;
        }
}
```

Задания

«4»: Моделирование кипения воды в стакане (синий фон, рамка):



«5»: Моделирование двустороннего потока: часть частиц двигаются влево, часть – вправо.



Программирование на языке Си Часть II

Тема 8. Символьные строки

Чем плох массив символов?

Это массивы символов:

```
char A[4] = { 'А', 'З', '[', 'Ж' };  
char B[10];
```

Для массива:

- каждый символ – отдельный объект;
- массив имеет длину N, которая задана при объявлении

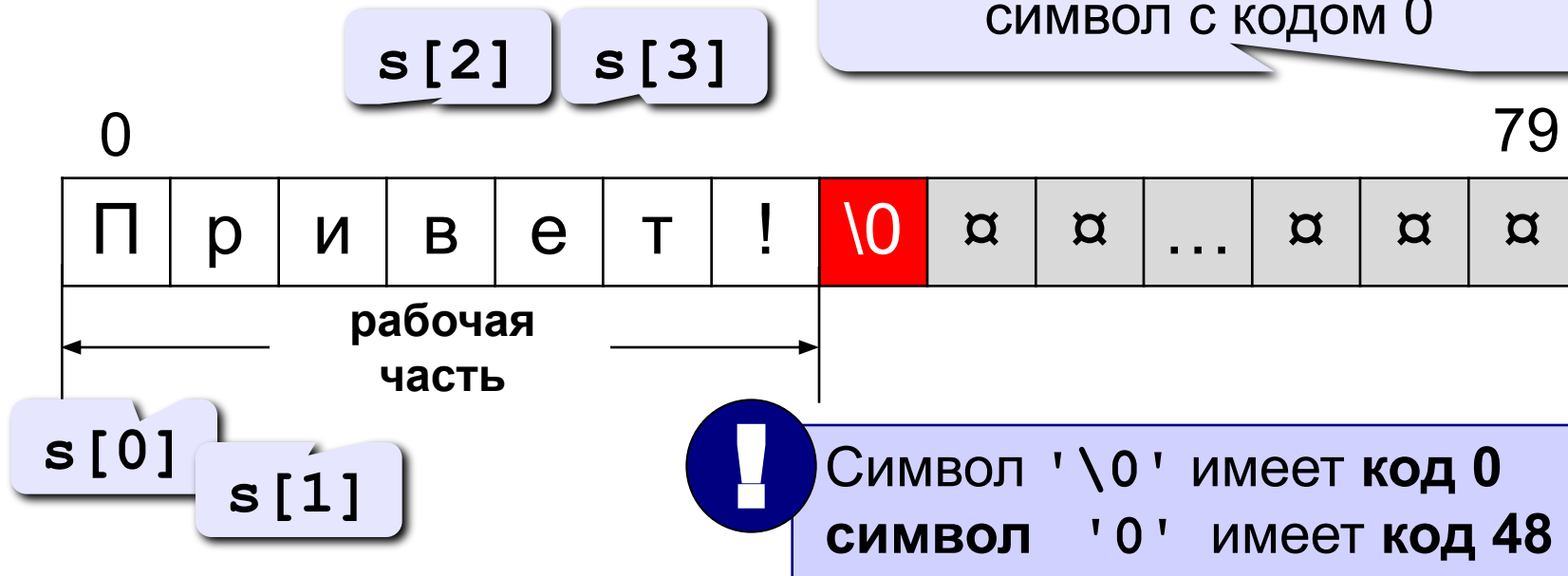
Что нужно:

- обрабатывать последовательность символов как единое целое
- строка должна иметь переменную длину

Символьные строки

```
char s[80];
```

признак окончания строки:
символ с кодом 0



Символьная строка – это последовательность символов, которая заканчивается символом `'\0'`.

Объявление символьных строк

Объявить строку = выделить ей место в памяти и присвоить имя.

```
char s[80];
```

выделяется 80 байт, в строке
– «мусор» (если она
глобальная, то нули '\0')

```
char s1[80] = "abc";
```

выделяется 80 байт,
занято 4 байта
(с учетом '\0')

```
char qqq[] = "Вася";
```

выделяется 5 байт
(с учетом '\0')



- При выделении памяти надо учитывать место для символа '\0'.
- В строку нельзя записывать больше символов, чем выделено памяти.

Ввод и вывод символьных строк

Задача: ввести слово с клавиатуры и заменить все буквы «а» на буквы «б».

```

main ()
{
    char q[80];
    printf ("Введите слово: ");
    scanf ( "%s", q );
    i = 0;
    while ( q[i] != '\0' ) {
        if ( q[i] == 'a' ) q[i] = 'б';
        i ++;
    }
    printf ( "Результат: %s", q );
}

```

%s – формат для ввода и вывода символьных строк (выводится только часть до '\0')

начали с q[0]

пока не дошли до конца строки

не надо ставить &: &q[0]

переход к следующему символу

Ввод символьных строк

Ввод одного слова:

```
char q[80];  
printf ("Введите текст:\n");  
scanf ( "%s" , q );  
printf ("Введено:\n%s" , q );
```

Введите текст:
Вася пошел гулять
Введено:
Вася

Ввод строки с пробелами:

```
char q[80];  
printf ("Введите текст:\n");  
gets ( q  
);  
printf ("Введено:\n%s" , q );
```

Введите текст:
Вася пошел гулять
Введено:
Вася пошел гулять

Вывод символьных строк

Универсальный способ:

```
printf ( "Результат: %s" , q ) ;
```

- можно выводить сразу и другую информацию: надписи, значения переменных, ...

Только для одной строки:

```
puts ( q ) ;
```



```
printf ( "%s\n" , q ) ;
```

- вывод только одной строки
- после вывода – переход на новую строку

Задания

«4»: Ввести символьную строку и заменить все буквы "а" на буквы "б" и наоборот, как заглавные, так и строчные.

Пример:

Введите строку:

ааббссААББСС

Результат:

ббаассББАСС

«5»: Ввести символьную строку и проверить, является ли она **палиндромом** (палиндром читается одинаково в обоих направлениях).

Пример:

Введите строку:

АБВГДЕ

Результат:

Не палиндром.

Пример:

Введите строку:

КАЗАК

Результат:

Палиндром.

Функции для работы со строками

Подключение библиотеки:

```
#include <string.h>
```

Длина строки: **strlen** (*string length*)

```
char q[80] = "qwerty";  
int n;  
n = strlen ( q );
```

n = 6

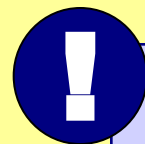


При определении длины символ ' \0 ' не учитывается!

Сравнение строк

strcmp (string comparison):

```
char q1[80], q2[80];  
int n;  
gets ( q1 );  
gets ( q2 );  
n = strcmp ( q1, q2 );
```



Функция вычисляет разность между кодами первых двух отличающихся символов!

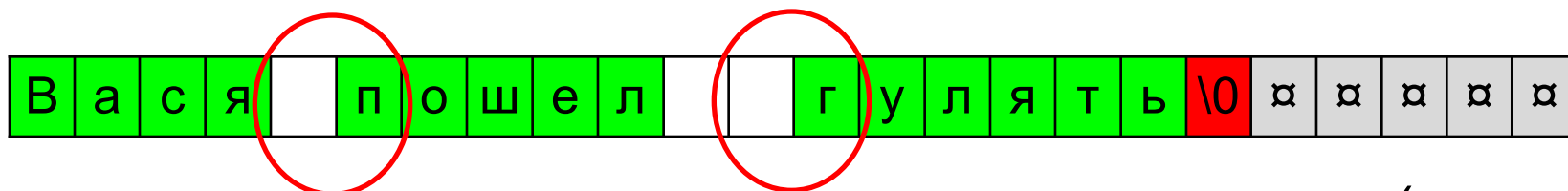
q1	q2	n
"AA"	"AA"	

Пример решения задачи

Задача: ввести строку и определить, сколько в ней слов.
Программа должна работать только при вводе правильного пароля.

Идея решения:

- проверка пароля – через *strcmp*
- количество слов = количеству первых букв слова
- первая буква: пробел и за ним «не пробел»



- исключение: предложение начинается со слова (а не с пробела)

Проверка пароля

```
#include <string.h>
main()
{
    char secret[] = "123", pass[20];
    printf ( "Введите пароль\n" );
    gets ( pass );
    if ( strcmp ( pass, secret ) != 0 )
    {
        printf ( "Пароль неверный" );
        getch ();
        return 1;
    }
    ...
}
```

если пароль
неверный...

сообщить об
ошибке и выйти
из программы

аварийное
завершение,
код ошибки 1

Основная часть программы

```
#include <stdio.h>
#include <string.h>
main()
{
    char q[80];
    int i, len, count = 0;
    ... // проверка пароля
    printf ("Введите предложение\n");
    gets ( q );
    len = strlen ( q );
    if ( q[0] != ' ') count++;
    for ( i = 0; i < len - 1; i ++ )
        if ( q[i] == ' ' && q[i+1] != ' ' )
            count ++;
    printf ( "Найдено %d слов", count );
}
```

предыдущий слайд

особый случай

если нашли
пробел, а за ним
не пробел...

Задания (везде – с паролем!)

«4»: Ввести предложение и определить, сколько слов заканчиваются на букву 'а'.

Пример:

Введите предложение:

Мама мыла раму

Найдено слов: 2

Введите предложение:

Декан пропил бутан

Нет таких слов

«5»: Ввести предложение и разобрать его на отдельные слова:

Пример:

Введите предложение:

Мама мыла раму

Результат:

Мама

мыла

раму

Подсказка: для вывода одного символа используйте функцию `putchar(символ)`.

Например:

```
putchar(q[i]);  
putchar('\n'); // переход на новую строку
```

Копирование строк

strcpy (*string copy*)

```
char q1[10] = "qwerty", q2[10] = "01234";
```

```
strcpy ( q1, q2 );
```



Старое значение q1
стирается!

куда

откуда

копирование «хвоста» строки

```
char q1[10] = "qwerty", q2[10] = "01234";
```

```
strcpy ( q1, q2+2 );
```

q2 = &q2[0]

q2+2 = &q2[2]

q1

2	3	4	\0	t	y	\0	␣	␣	␣
---	---	---	----	---	---	----	---	---	---

q2

0	1	2	3	4	\0	␣	␣	␣	␣
---	---	---	---	---	----	---	---	---	---



Копирование строк

копирование в середину строки

```
char q1[10] = "qwerty", q2[10] = "01234";
strcpy ( q1+2, q2 );
```

$q1+2 = \&q1[2]$



```
char q1[10] = "qwerty", q2[10] = "01234";
strcpy ( q1+2, q2+3 );
```

$q1+2 = \&q1[2]$

$q2+3 = \&q2[3]$



Копирование строк

strcpy – копирование нескольких символов

```
char q1[10] = "qwerty", q2[10] = "01234";  
strcpy ( q1+2, q2, 2 );
```

$q1+2 = \&q1[2]$

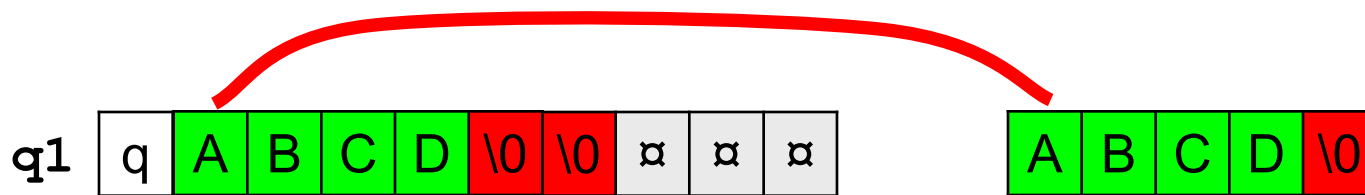


Функция **strcpy** не добавляет символ ' \0 ' в конце строки!

Копирование строк

копирование строки-константы

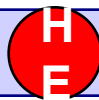
```
char q1[10] = "qwerty";
strcpy ( q1+1, "ABCD" );
```



```
char q1[10] = "qwerty";
strcpy ( "ABCD", q1+2 );
```



Первым параметром

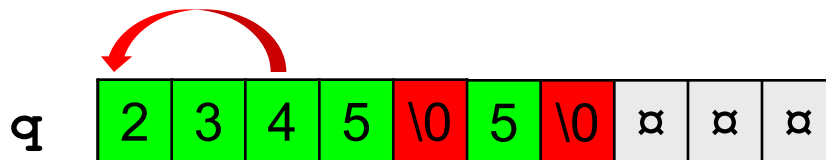


может быть константа!

Копирование строк

копирование внутри одной строки

```
char q[10] = "012345";
strcpy ( q, q+2 );
```



```
char q[10] = "012345";
strcpy ( q+2, q );
```

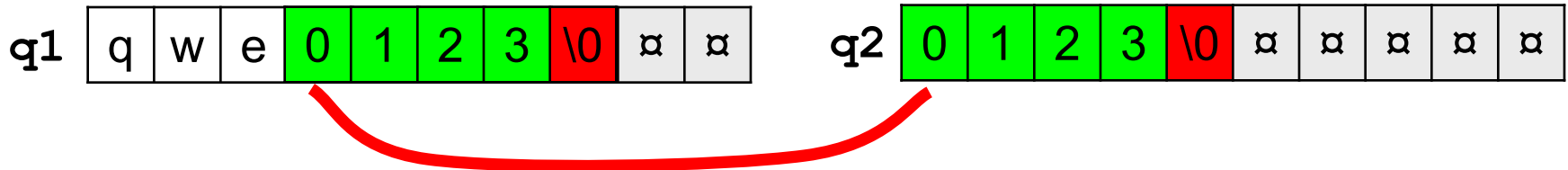


Заикливание и зависание компьютера!

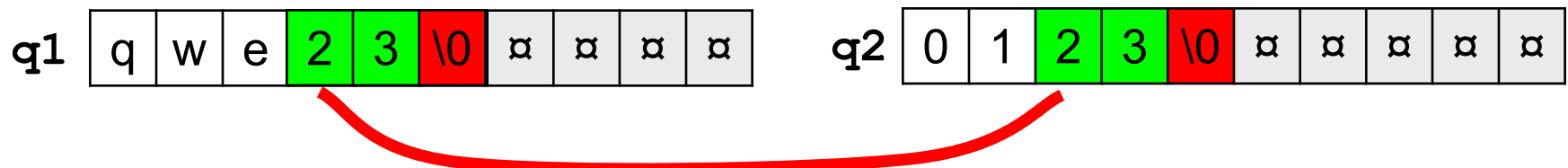
Объединение строк

***strcat* (string concatenation)** = копирование второй строки в конец первой

```
char q1[10] = "qwe", q2[10] = "0123";
strcat ( q1, q2 );
```



```
char q1[10] = "qwe", q2[10] = "0123";
strcat ( q1, q2+2 );
```



Проблемы при копировании строк

- не хватает места для строки-результата

```
char q1[] = "qwer", q2[10] = "01234";
strcpy ( q1+2, q2 );
```



- зацикливание при копировании в ту же строку «слева направо»

```
char q[10] = "01234";
strcpy ( q+2, q );
```



Транслятор не сообщает об этих ошибках!

Пример решения задачи

Задача: ввести имя файла (без пути) и поменять его расширение на `".exe"`.

Пример:

Введите имя файла:

`vasya.html`

Результат:

`vasya.exe`

Введите имя файла:

`vasya`

Результат:

`vasya.exe`

Алгоритм:

- найти точку в имени файла
- если она есть, скопировать в это место строку-константу `".exe"`
- если точки нет, добавить в конец строки `".exe"`

Программа

```
main()  
{  
char fName[80];  
int i;  
printf("Введите имя файла\n");  
gets ( fName );
```

```
i = 0;  
while ( fName[i] != '.' ) {  
    if ( fName[i] == '\0' ) break;  
    i ++;  
}
```

```
if ( fName[i] == '.' )  
    strcpy ( fName+i, ".exe" );  
else strcat ( fName, ".exe" );  
puts ( "Результат:" );  
puts ( fName );  
}
```

ПОИСК
ТОЧКИ

ДОШЛИ ДО
КОНЦА СТРОКИ

МЕНЯЕМ ИЛИ
ДОБАВЛЯЕМ
РАСШИРЕНИЕ

Задания

«4»: Ввести полный адрес файла (возможно, без расширения) и изменить его расширение на «.exe».

Пример:

Введите имя файла:

C:\DOC.TXT\qqq

Результат:

C:\DOC.TXT\qqq.exe

Введите имя файла:

C:\DOC.TXT\qqq.com

Результат:

C:\DOC.TXT\qqq.exe

«5»: Ввести в одной строке фамилию, имя и отчество. Вывести приветствие, где останутся имя и фамилия (см. пример).

Пример:

Введите ФИО:

Пупкин Василий Иванович

Результат:

Привет, Василий Пупкин!

Поиск в символьных строках

Задача: найти заданный символ или сочетание символов (**подстроку**) в символьной строке.



Функции поиска в Си возвращают адрес найденного символа или подстроки!

Если образец не найден, возвращается **NULL** (нулевой адрес).

Указатель – это переменная в которую можно записать адрес другой переменной заданного типа.

Указатели

pointer – указатель

Объявление:

```
char *p;    // адрес любого символа или строки
int *pI;   // адрес целого числа
float *pF;  // адрес вещественного числа
```

Целые переменные и массивы:

```
int n = 6, A[5] = {0, 1, 2, 3, 4};
int *p;    // указатель на целое
p = &n;    // записать адрес n
*p = 20;   // n = 20
p = A + 2; // записать адрес A[2] (&A[2])
*p = 99;   // изменить A[2]
p++;      // перейти к A[3]
printf("Адрес: %p, число %d", p, *p);
```

Адрес: 6BCD:000C, значение 3

Указатели и символьные строки

```
char str[10] = "0123456" ;
char *p;           // указатель на символ
p = str;          // или &str[0]
*p = 'A' ;        // "A12345"
p ++;             // перейти к str[1]
*p = 'B' ;        // "AB2345"
p ++;             // перейти к str[2]
strcpy ( p, "CD" ) ; // "ABCD"
strcat ( p, "qqq" ) ; // "ABCDqqq"
puts ( p ) ;
```

Поиск символа

strchr: найти первый заданный символ с начала строки

```
char q[10] = "abcdabcd";
```

```
char *p;
```

```
int nomer;
```

```
p = strchr(q, 'b');
```

```
if (p == NULL)
```

```
    printf("Не нашли...");
```

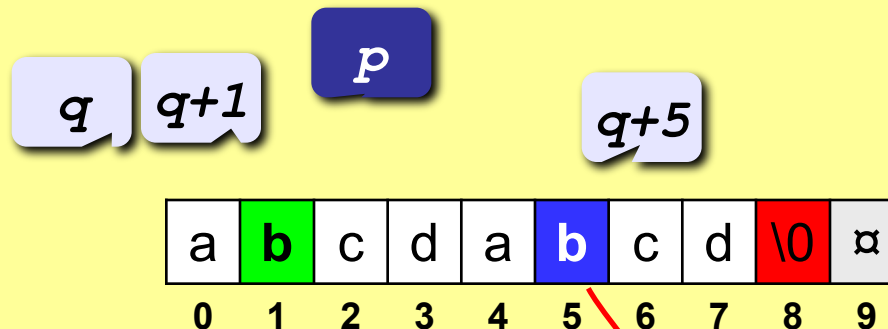
```
else {
```

```
    nomer = p - q;
```

```
    printf("Номер символа %d", nomer);
```

```
}
```

reverse



strrchr: найти последний заданный символ в строке

Поиск подстроки

strstr: найти первую подстроку с начала строки

```
char q[10] = "abcdabcd";
```

```
char *p;
```

```
int nomer;
```

```
p = strstr(q, "bcd");
```

```
if (p == NULL)
```

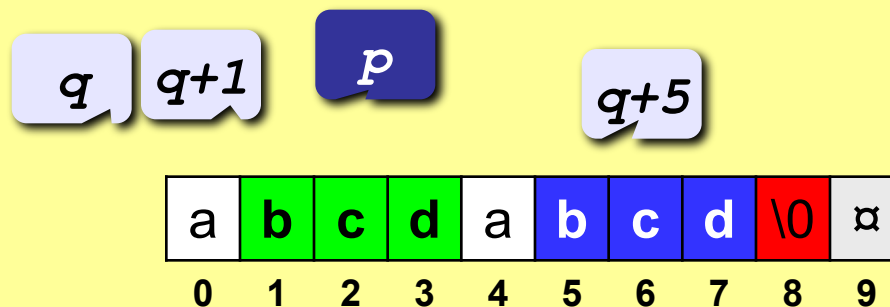
```
    printf("Не нашли...");
```

```
else {
```

```
    nomer = p - q;
```

```
    printf("Номер первого символа %d", nomer);
```

```
}
```



Пример решения задачи

Задача: ввести предложение и определить, сколько раз в нем встречается имя «Вася».

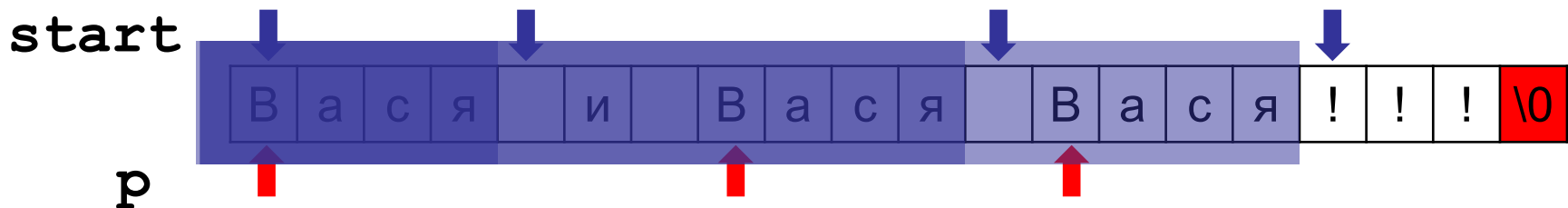
Проблема: функция *strstr* ищет только с начала строки.

Алгоритм:

1. Записать адрес начала строки в указатель **start**.
2. Искать подстроку «Вася», начиная с адреса **start**.

```
p = strstr( start, "Вася" );
```

3. Если не нашли, выход из цикла.
4. Увеличить счетчик найденных слов.
5. Переставить **start** на адрес после найденного слова.
6. Перейти к шагу 2.



Программа

```
main ()
{
    char q[80], *start, *p;
    int count = 0;
    puts ( "Введите предложение" );
    gets ( q );
    start = q; // ищем с начала строки

    while ( 1 ) {
        p = strstr ( start, "Вася" );
        if ( p == NULL ) break;
        count ++;
        start = p + 4; // отсюда ищем следующее слово
    }

    printf ( "Имя 'Вася' встречается %d раз", count );
}
```

начало поиска

адрес
найденного
слова

Задания

«4»: Ввести предложение и заменить все имена «Вася» на «Юра».

Пример:

Введите предложение:

Вася, Вася, Вася и Вася!!!

Результат:

Юра, Юра, Юра и Юра!!!

«5»: Ввести предложение и заменить все имена «Юра» на «Вася».

Пример:

Введите предложение:

Юра, Юра, Юра и Юра!!!

Результат:

Вася, Вася, Вася и Вася!!!

Строки в процедурах и функциях



- строки передаются в функции и процедуры так же, как и массивы;
- функции и процедуры могут изменять строки – параметры.

Задача: составить процедуру, которая переставляет символы строки в обратном порядке.

Алгоритм:

- определить длину строки **len**;
- все символы первой половины переставить с соответствующими символами второй половины:

`s[i] ↔ s[len-1-i]`

```
c = s[i];  
s[i] = s[len-i-1];  
s[len-1-i] = c;
```


Программа

```
void Reverse ( char s[] )
{
    int len = strlen(s);
    char c;
    for ( i = 0; i < len/2; i++ ) {
        c = s[i];
        s[i] = s[len-i-1];
        s[len-1-i] = c;
    }
}

main()
{
    char s[] = "1234567890";
    Reverse ( s );
    puts ( s );
    Reverse ( s + 5 );
    puts ( s );
}
```

длину строки
определяем на месте



Как сделать
инверсию **любой**
части строки?

0987654321

0987612345

Задания

«4»: Разработать процедуру, которая переставляет пары соседних символов.

Пример:

Введите предложение :

Вася пошел гулять!

Результат :

аВясп шoлeг лутя!ь

«5»: Разработать процедуру, которая удаляет все лишние пробелы (в начале предложения и сдвоенные пробелы).

Пример:

Введите предложение :

Вася пошел гулять!

Результат :

Вася пошел гулять!

Символьные строки в функциях

Задача: составить функцию, которая находит количество цифр в строке.

```
int NumDigits ( char s[] )
{
    int i, count = 0;
    for ( i = 0; i < strlen(s); i++ )
        if( strchr ( "0123456789", s[i] ) )
            count++;
    return count;
}
```

```
if ( strchr ( "0123456789", s[i] ) != NULL )
```

ИЛИ

```
if ( '0' <= s[i] && s[i] <= '9' )
```

Символьные строки в функциях

Основная программа

```
int NumDigits ( char s[] )
{
    ...
}
main()
{
    char s[80];
    int n;
    printf ( "Введите строку\n" );
    gets ( s );
    n = NumDigits ( s );
    printf ( "Нашли %d цифр.", s );
}
```

Задания

«4»: Разработать функцию, которая определяет, верно ли, что слово – палиндром.

Пример:

Введите слово: Введите слово:

казак кунак

Результат: Результат:

Это палиндром. Не палиндром.

«5»: Разработать функцию, которая определяет, верно ли, что *предложение* (с пробелами) – палиндром.

Пример:

Введите предложение:

а роза упала на лапу азора

Результат:

Это палиндром.

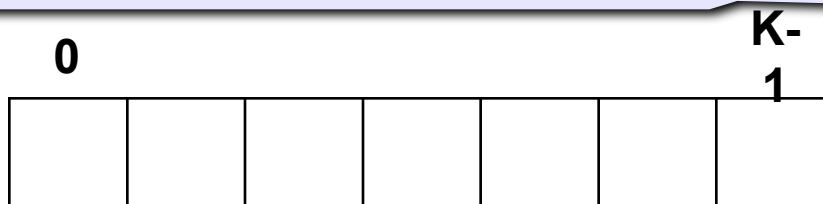
Программирование на языке Си Часть II

Тема 9. Рекурсивный перебор

Рекурсивный перебор

Задача: Алфавит языка племени «тумба-юмба» состоит из букв **Ы**, **Ц**, **Щ** и **О**. Вывести на экран все слова из **K** букв, которые можно составить в этом языке, и подсчитать их количество. Число **K** вводится с клавиатуры.

в каждой ячейке может быть любая из 4-х букв



4 вари

4 вариант

4 варианта

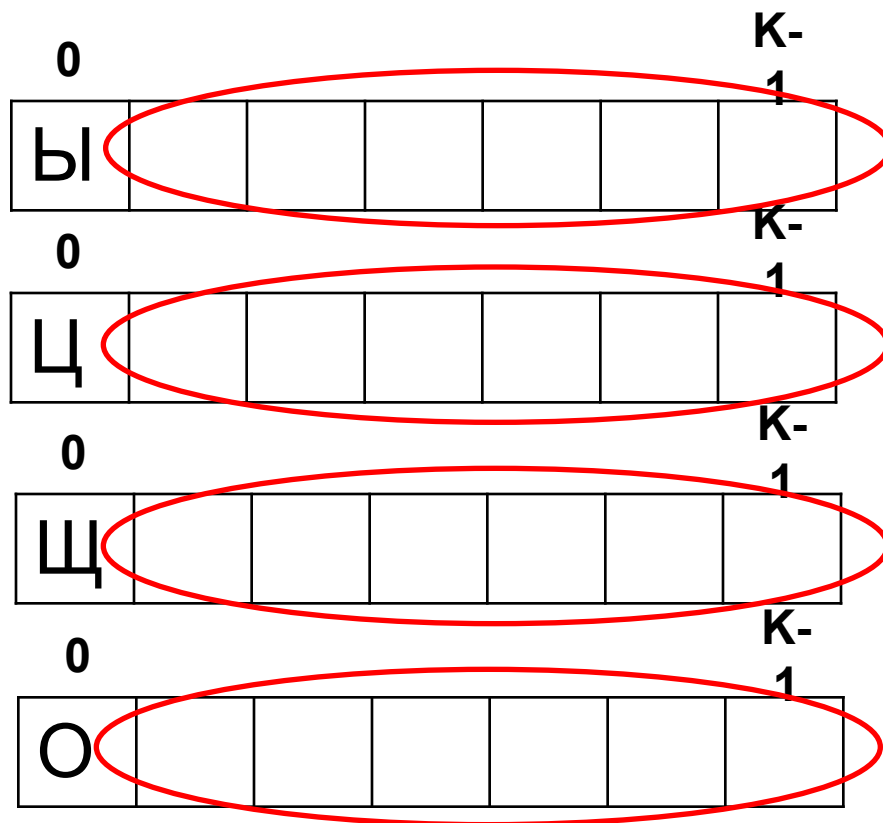
4 варианта

Количество вариантов:

$$N = 4 \cdot 4 \cdot 4 \cdot \dots \cdot 4 = 4^K$$

Рекурсивный перебор

Рекурсия: Решения задачи для слов из K букв сводится к 4-м задачам для слов из $K-1$ букв.



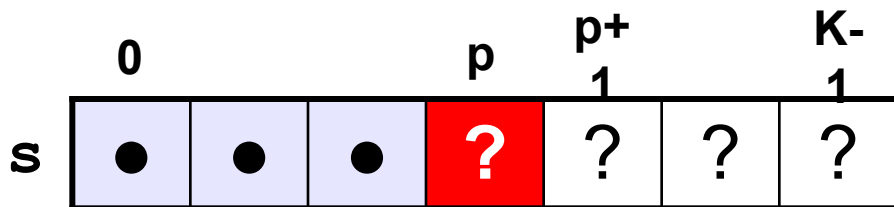
перебрать все варианты

перебрать все варианты

перебрать все варианты

перебрать все варианты

Процедура



Глобальные переменные:
`char s[80];`
`int count, K;`

```
void Rec( int p )
```

```
{
```

```
    if ( p >= K ) {
```

```
        puts ( s );
```

```
        count ++;
```

```
        return;
```

```
    }
```

```
    s[p] = 'Ы' ; Rec ( p + 1 );
```

```
    s[p] = 'Ц' ; Rec ( p + 1 );
```

```
    s[p] = 'Щ' ; Rec ( p + 1 );
```

```
    s[p] = 'О' ; Rec ( p + 1 );
```

```
}
```

`p` СИМВОЛОВ УЖЕ НА
СВОИХ МЕСТАХ

окончание рекурсии

РЕКУРСИВНЫЕ ВЫЗОВЫ



А если букв много?

Процедура

```
void Rec ( int p )
```

```
{
```

```
    const char letters[] = 'ЫЦЩО';
```

все буквы

```
    int i;
```

локальная переменная

```
    if ( p >= K ) {
```

```
        puts ( s );
```

```
        count ++;
```

```
        return;
```

```
    }
```

цикл по всем буквам

```
    for ( i = 0; i < strlen( letters ); i ++ ) {
```

```
        s[ p ] = letters[ i ];
```

```
        Rec ( p + 1 );
```

```
    }
```

```
}
```

Программа

```
char s[80];
```

```
int K, count;
```

глобальные переменные
(обнуляются)

```
void Rec ( int p )
```

```
{
```

```
...
```

```
}
```

процедура

```
main ( )
```

```
{
```

```
printf ( "Введите длину слов:\n" );
```

```
scanf ( "%d", &K );
```

```
Rec ( 0 );
```

```
printf ( "Всего %d слов.", count );
```

```
}
```



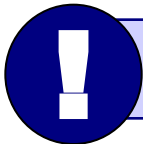
1. Как определяется конец строки?

2. Как обойтись без глобальных переменных?

Задания

Алфавит языка племени "тумба-юмба" состоит из букв **Ы**, **Ц**, **Щ** и **О**. Число **К** вводится с клавиатуры.

- «4»: Вывести на экран все слова из **К** букв, в которых буква **Ы** встречается более 1 раза, и подсчитать их количество.
- «5»: Вывести на экран все слова из **К** букв, в которых есть одинаковые буквы, стоящие рядом (например, **ЫЩЩО**), и подсчитать их количество.



Без глобальных переменных!

Программирование на языке Си Часть II

Тема 10. Матрицы

Матрицы

Матрица – это прямоугольная таблица однотипных элементов.

Матрица – это массив, в котором каждый элемент имеет два индекса (номер строки и номер столбца).

A

	0	1	2	3	4
0	1	4	7	3	6
1	2	-5	0	15	10
2	8	9	11	12	20

столбец 2

строка 1

ячейка **A**[2][3]

Матрицы

Объявление:

```
const int N = 3, M = 4;
int A[N][M];
float a[2][2] = {{3.2, 4.3}, {1.1, 2.2}};
char sym[2][2] = { 'a', 'b', 'c', 'd' };
```

Ввод с клавиатуры:

```
for ( j=0; j<M; j++ )
    for ( i=0; i<N; i++ ) {
        printf ( "A[%d][%d]=", i, j );
        scanf ( "%d", &A[i][j] );
    }
```

<i>i</i>	<i>j</i>		
		A[0][0]	2
		A[0][1]	5
		A[0][2]	4
]=	4
		A[2][3]	5
]=	4



Если переставить циклы?

Матрицы

Заполнение случайными числами

```
for ( i = 0; i < N; i ++ )
    for ( j = 0; j < M; j ++ )
        A[i][j] = random(25) - 10;
```

цикл по строкам

интервал?

цикл по столбцам

Вывод на экран

```
for ( i = 0; i < N; i ++ ) {
    for ( j = 0; j < M; j ++ )
        printf("%5d", A[i][j]);
    printf("\n");
}
```

ВЫВОД строки

12	25	1	13
156	1	12	447
1	456	222	23

в той же строке

перейти на
новую строку



Если переставить циклы?

Обработка всех элементов матрицы

Задача: заполнить матрицу из 3 строк и 4 столбцов случайными числами и вывести ее на экран. Найти сумму элементов матрицы.

```
main()  
{  
    const int N=3, M=4;  
    int A[N][M], i, j, S=0;  
    ... // заполнение матрицы и вывод на экран  
    for ( i = 0; i < N; i ++ )  
        for ( j = 0; j < M; j ++ )  
            S += A[i][j];  
    printf("Сумма элементов матрицы S=%d", S);  
}
```

Задания

Заполнить матрицу из 8 строк и 5 столбцов случайными числами в интервале $[-10, 10]$ и вывести ее на экран.

«4»: Найти минимальный и максимальный элементы в матрице их номера. Формат вывода:

Минимальный элемент $A[3][4] = -6$

Максимальный элемент $A[2][2] = 10$

«5»: Вывести на экран строку, сумма элементов которой максимальна. Формат вывода:

Строка 2: 3 5 8 9 8

Операции с матрицами

Задача 1. Вывести на экран главную диагональ квадратной матрицы из N строк и N столбцов.

Diagram illustrating the main diagonal of a square matrix. The diagonal elements are highlighted in green. Callouts point to the elements $A[0][0]$, $A[1][1]$, $A[2][2]$, and $A[N-1][N-1]$.

```
for ( i = 0; i < N; i ++ )
    printf ( "%5d", A[i][i] );
```

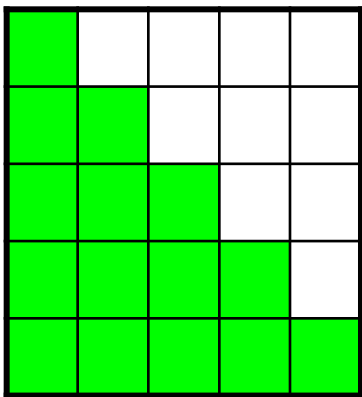
Задача 2. Вывести на экран вторую диагональ.

Diagram illustrating the second diagonal of a square matrix. The diagonal elements are highlighted in green. Callouts point to the elements $A[0][N-1]$, $A[1][N-2]$, $A[N-2][1]$, and $A[N-1][0]$. The text "сумма номеров строки и столбца $N-1$ " is associated with the first callout.

```
for ( i = 0; i < N; i ++ )
    printf ( "%5d", A[i][N-1-i] );
```

Операции с матрицами

Задача 3. Найти сумму элементов, стоящих на главной диагонали и ниже ее.



Одиночный цикл или вложенный?

строка 0: $A[0][0]$

строка 1: $A[1][0] + A[1][1]$

...

строка i : $A[i][0] + A[i][2] + \dots + A[i][i]$

цикл по всем строкам

```

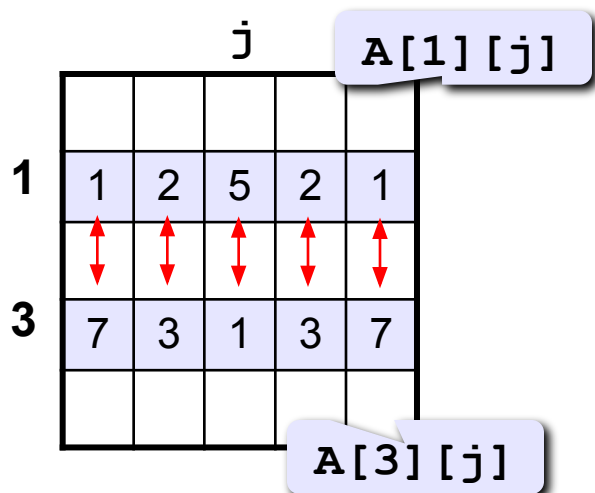
s = 0;
for ( i = 0; i < N; i ++ )
    for ( j = 0; j <= i; j ++ )
        s += A[i][j];

```

складываем нужные
элементы строки i

Операции с матрицами

Задача 4. Перестановка строк или столбцов. В матрице из N строк и M столбцов переставить 1-ую и 3-ю строки.



```
for ( j = 0; j <= M; j ++ ) {
    c = A[1][j];
    A[1][j] = A[3][j];
    A[3][j] = c;
}
```

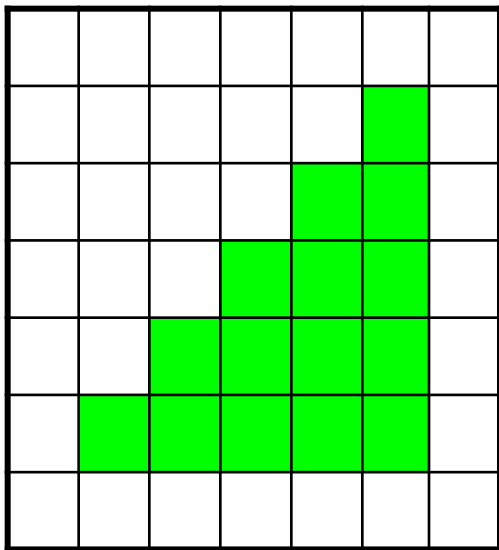
Задача 5. К третьему столбцу добавить шестой.

```
for ( i = 0; i < N; i ++ )
    A[i][3] += A[i][6];
```

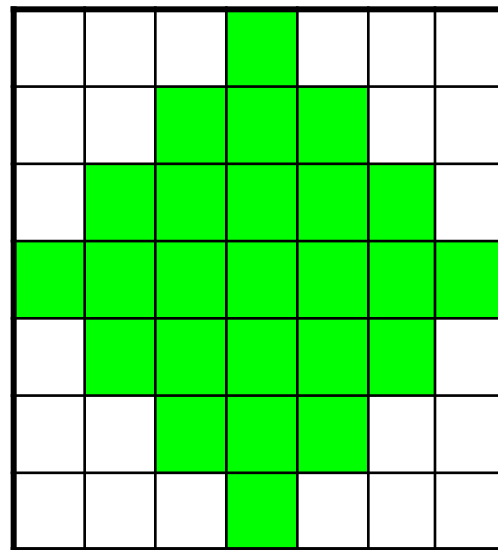
Задания

Заполнить матрицу из 7 строк и 7 столбцов случайными числами в интервале $[-10, 10]$ и вывести ее на экран. Обнулить элементы, отмеченные зеленым фоном, и вывести полученную матрицу на экран.

«4»:



«5»:



Программирование на языке Си Часть II

Тема 11. Файлы

Файлы

Файл – это область на диске, имеющая имя.

Файлы

ы

Текстовые

е

только текст без оформления,
не содержат управляющих
символов (с кодами < 32),
кроме перевода строки

ASCII (1 байт на символ)

UNICODE (2 байта на символ)

*.txt, *.log,

*.htm, *.html

Двоичные

могут содержать любые
символы кодовой таблицы

*.doc, *.exe,

*.bmp, *.jpg,

*.wav, *.mp3,

*.avi, *.mpg

Папки (каталоги)

Принцип сэндвича

Переменная типа
«указатель на файл»:

```
FILE *f;
```

I этап. открыть файл (
активным, приготовить к работе)

для чтения ("r", англ. *read*)

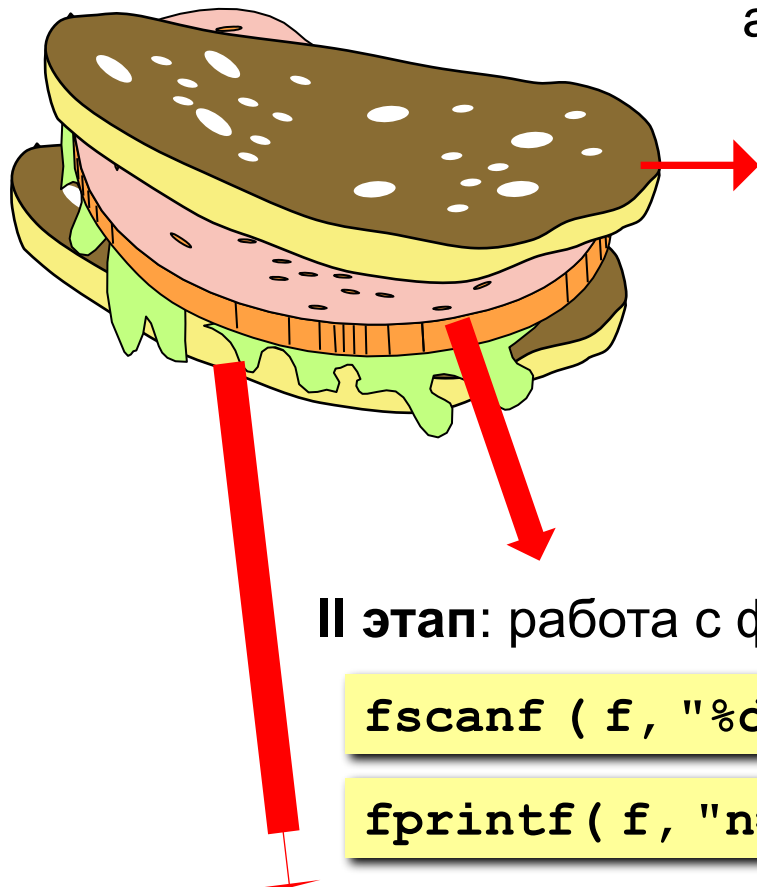
```
f = fopen("qq.dat", "r");
```

для записи ("w", англ. *write*)

```
f = fopen("qq.dat", "w");
```

для добавления ("a", англ. *append*)

```
f = fopen("qq.dat", "a");
```



II этап: работа с файлом

```
fscanf ( f, "%d", &n ); // ввести значение n
```

```
fprintf( f, "n=%d", n ); // записать значение n
```

III этап: закрыть (освободить) файл

```
fclose ( f );
```

Работа с файлами

Особенности:

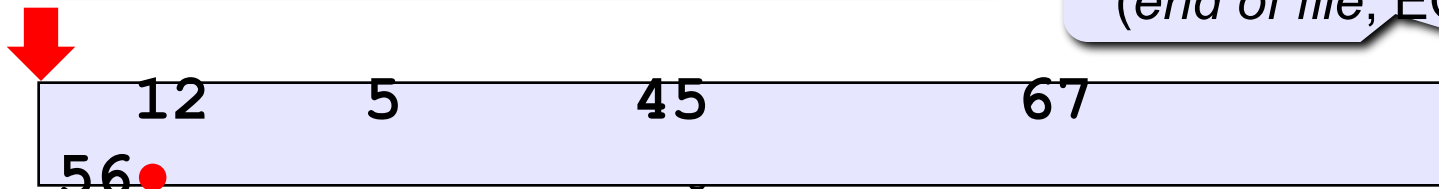
- имя файла упоминается только в команде **foren**, обращение к файлу идет через указатель **f**;
- файл, который открывается на чтение, должен **существовать**
- если файл, который открывается на запись, существует, старое содержимое **уничтожается**
- данные (*этим способом*) записываются в файл в текстовом виде
- когда программа заканчивает работу, все файлы закрываются автоматически
- после закрытия файла переменную **f** можно использовать еще раз для работы с другим файлом

Последовательный доступ

- при открытии файла курсор устанавливается в начало

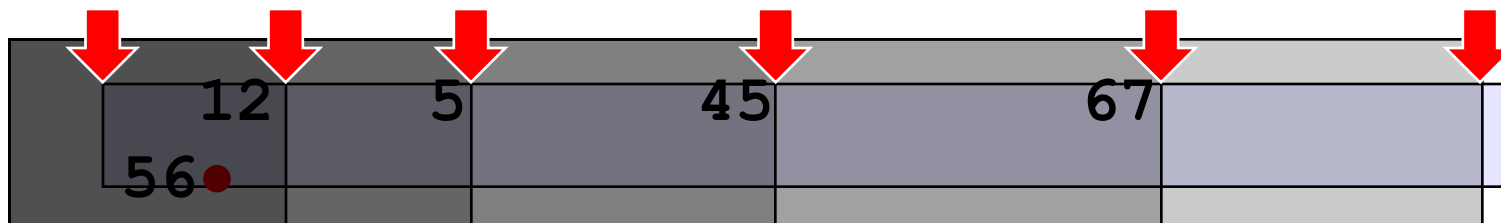
```
f = fopen("qq.dat", "r");
```

конец файла
(end of file, EOF)




- чтение выполняется с той позиции, где стоит курсор
- после чтения курсор сдвигается на первый непрочитанный символ

```
fscanf ( f, "%d", &x );
```



Как вернуться назад?

Ошибки при открытии файла

 Если файл открыть не удалось, функция `fopen` возвращает **NULL** (нулевое значение)!

```
FILE *f;  
f = fopen("qq.dat", "r");  
if ( f == NULL ) {  
    puts("Файл не найден.");  
    return;  
}
```

- неверное имя файла
- нет файла
- файл заблокирован другой программой

```
FILE *f;  
f = fopen("qq.dat", "w");  
if ( f == NULL ) {  
    puts("Не удалось открыть файл.");  
    return;  
}
```

- неверное имя файла
- файл «только для чтения»
- файл заблокирован другой программой

Пример

Задача: в файле `input.txt` записаны числа (в столбик), сколько их – неизвестно. Записать в файл `output.txt` их сумму.



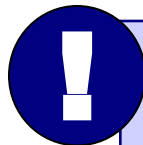
Можно ли обойтись без массива?

Алгоритм:

1. Открыть файл `input.txt` для чтения.
2. `S = 0;`
3. Прочитать очередное число в переменную `x`.
4. Если не удалось, перейти к шагу 7.
5. `S += x;`
6. Перейти к шагу 3.
7. Закрыть файл `input.txt`.
8. Открыть файл `output.txt` для записи.
9. Записать в файл значение `S`.
10. Закрыть файл `output.txt`.

цикл с условием
«пока есть данные»

Как определить, что числа кончились?



Функция `fscanf` возвращает количество удачно прочитанных чисел;
0, если была ошибка при чтении;
– 1, если достигли конца файла.

```
FILE *f;  
int n, x;  
f = fopen("input.txt", "r")  
...  
n = fscanf ( f, "%d", &x );  
if ( n != 1 )  
    puts ( "Не удалось прочитать число" );
```

- дошли до конца файла
- встретили «не число»

Программа

```
main()
{
FILE *f;
int n, x, S = 0;
f = fopen ( "input.txt", "r" );
if ( f == NULL ) {
    printf("Файл не найден.");
    return;
}
while ( 1 ) {
    n = fscanf ( f, "%d", &x );
    if ( n != 1 ) break;
    S += x;
}
fclose ( f );
f = fopen ( "output.txt", "w" );
fprintf ( f, "S = %d", S );
fclose ( f );
}
```

ошибка при
открытии
файла

ЦИКЛ ЧТЕНИЯ ДАННЫХ:
выход при $n \neq 1$.

запись
результата

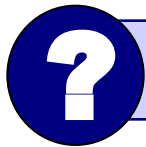
Задания

В файле `input.txt` записаны числа, сколько их – неизвестно.

- «4»: Найти среднее арифметическое всех чисел и записать его в файл `output.txt`.
- «5»: Найти минимальное и максимальное числа и записать их в файл `output.txt`.

Обработка массивов

Задача: в файле `input.txt` записаны числа (в столбик), сколько их – неизвестно, но не более 100. Переставить их в порядке возрастания и записать в файл `output.txt`.



Можно ли обойтись без массива?

Проблемы:

- для сортировки надо удерживать в памяти все числа сразу (массив);
- сколько чисел – неизвестно.

Решение:

- 1) выделяем в памяти массив из 100 элементов;
- 2) записываем прочитанные числа в массив и считаем их в переменной **N**;
- 3) сортируем первые **N** элементов массива;
- 4) записываем их в файл.

Чтение данных в массив

Функция, которая читает массив из файла, возвращает число прочитанных элементов (не более MAX):

```
int ReadArray ( int A[], char fName[], int MAX )
{
    int N=0, k;
    FILE *f;
    f = fopen ( fName, "r" );
    while ( 1 ) {
        k = fscanf ( f, "%d", &A[N] );
        if ( k != 1 ) break;
        N ++;
        if ( N >= MAX ) break;
    }
    fclose(f);
    return N;
}
```

массив

имя файла

предел

заканчиваем цикл
если не удалось
прочитать ...

... или заполнили
весь массив

Программа

```
int ReadArray(int A[], char fName[], int MAX)
{
    ...
}
```

```
main()
```

```
{
    int A[100], N, i;
    FILE *f;
    N = ReadArray ( A, "input.txt", 100 );
    ... // сортировка первых N элементов
    f = fopen("output.txt", "w");
    for ( i = 0; i < N; i ++ )
        fprintf ( f, "%d\n", A[i] );
    fclose ( f );
}
```

Вывод отсортированного массива в файл

Задания

В файле `input.txt` записаны числа (в столбик), известно, что их не более 100.

- «4»: Отсортировать массив по убыванию последней цифры и записать его в файл `output.txt`.
- «5»: Отсортировать массив по возрастанию суммы цифр и записать его в файл `output.txt`.

Обработка текстовых данных

Задача: в файле `input.txt` записаны строки, в которых есть слово-паразит "*короче*". Очистить текст от мусора и записать в файл `output.txt`.

Файл `input.txt` :

Мама, короче, мыла, короче, раму.

Декан, короче, пропил, короче, бутан.

А роза, короче, упала на лапу, короче, Азора.

Каждый, короче, охотник желает, короче, знать, где ...

Результат – файл `output.txt` :

Мама мыла раму.

Декан пропил бутан.

А роза упала на лапу Азора.

Каждый охотник желает знать, где сидит фазан.

Обработка текстовых данных

Особенность:

надо одновременно держать открытыми два файла (один в режиме чтения, второй – в режиме записи).

Алгоритм:

1. Открыть оба файла.
2. Прочитать строку.
3. Удалить все сочетания "*короче*".
4. Записать строку во второй файл.
5. Перейти к шагу 2.
6. Закрыть оба файла.

пока не кончились
данные

Работа с файлами

```
main()  
{  
    char s[80], *p;  
    int i;  
    FILE *fIn, *fOut;  
    fIn = fopen("input.txt", "r");  
    fOut = fopen("output.txt", "w");  
    ... // обработать файл  
    fclose(fIn);  
    fclose(fOut);  
}
```

указатель
для поиска

файловые
указатели

открыть файл для чтения

открыть файл
для записи

закрывать
файлы

Обработка текстовых данных

Чтение строки s:

```
char s[80], *p;  
FILE *fIn;  
... // здесь надо открыть файл  
      строка   длина   файл  
p = fgets ( s, 80, fIn );  
if ( p == NULL )  
    printf("Файл закончился.");  
else printf("Прочитана строка:\n%s", s);
```

Обработка строки s:

```
while ( 1 ) {  
    p = strstr ( s, ", короче," );  
    if ( p == NULL ) break;  
    strcpy ( p, p + 9 );  
}
```

искать ", короче,"

Выйти из цикла,
если не нашли

удалить 9 символов

Полный цикл обработки файла

```
#include <string.h>
```

читаем
строку

```
while ( 1 ) {  
    p = fgets ( s, 80, fIn );  
    if ( p == NULL ) break;
```

если нет больше
строк, выйти из
цикла

```
    while ( 1 ) {  
        p = strstr ( s, ", короче, " );  
        if ( p == NULL ) break;  
        strcpy ( p, p + 9 );  
    }
```

обработка
строки

```
    fputs ( s, fOut );  
}
```

запись "очищенной"
строки

Задания

В файле `input.txt` записаны строки, сколько их – неизвестно.

«4»: Заменить во всем тексте «в общем» на «короче» и записать результат в файл `output.txt`.

«5»: Заменить во всем тексте «короче» на «в общем» и записать результат в файл `output.txt`.

Двоичные файлы

Особенности:

- данные хранятся во внутреннем **машинном формате** (в текстовом редакторе не прочитать)
- можно читать и записывать любой кусок памяти (просто биты...)
- принцип сэндвича (открыть – работать – закрыть)
- обращение к файлу через указатель

Файловые указатели

```
FILE *fp;
```

Открытие и закрытие двоичных файлов

Открытие файла

```
fp = fopen ( "input.dat", "rb" );
```

"rb" = *read binary* (чтение)

"wb" = *write binary* (запись)

"ab" = *append binary* (добавление)

Ошибки при открытии

```
if ( fp == NULL ) {  
    printf ("Файл открыть не удалось.");  
}
```

Закрытие файла

```
fclose ( fp );
```

Чтение по блокам

Чтение в начало массива

размер одного
блока

указатель
на файл

```
int A[100];
n = fread ( A, sizeof(int), 100, fp );
```

прочитано
фактически

адрес области
памяти («куда»):
A ⇔ &A[0]

размер
переменной
целого типа

количество
блоков

Чтение в середину массива

```
int A[100];
n = fread ( A+5, sizeof(int), 2, fp );
```

читается 2 целых числа:
A[5], A[6]

Запись по блокам

Запись с начала массива

размер одного
блока

указатель
на файл

```
int A[100];
n = fwrite( A, sizeof(int), 100, fp );
```

записано
фактически

адрес области
памяти («откуда»):
A ⇔ &A[0]

размер
переменной
целого типа

количество
блоков

Запись отдельных элементов массива

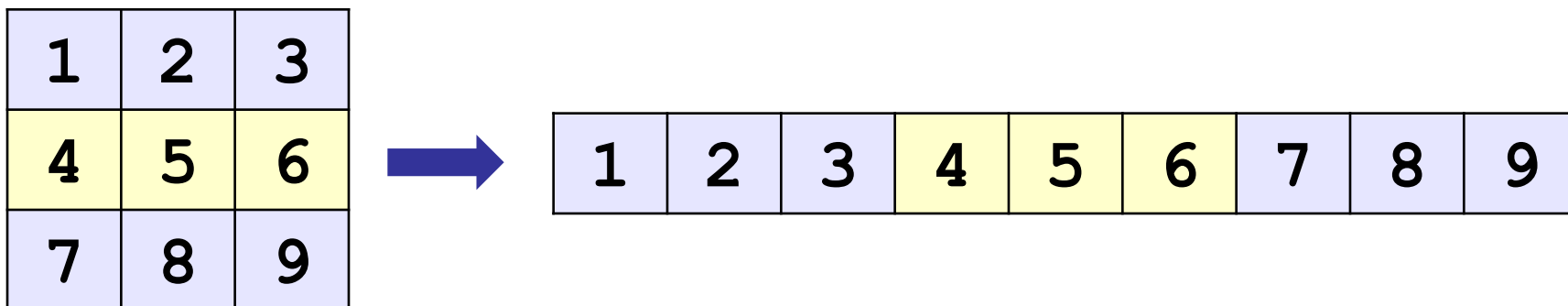
```
int A[100];
n = fwrite( A+5, sizeof(int), 2, fp );
```

записывается 2 целых числа:

A[5], A[6]

Работа с матрицами

Хранение в памяти: по строкам (Си, Паскаль)



Запись матрицы

```
int A[3][3];  
FILE *fp = fopen("output.dat", "wb");  
... // здесь заполняем матрицу  
n = fwrite(A, sizeof(int), 9, fp);
```


Пример

Задача: прочитать массив из файла `input.dat`, умножить все элементы на 2 и вывести в файл `output.dat`.

Структура программы:

```
#include <stdio.h>
main()
{
  const int N = 10;
  int i, A[N], n;
  FILE *fp;
  // чтение данных и файла input.dat
  for ( i = 0; i < n; i ++ )
    A[i] = A[i] * 2;
  // запись данных в файл output.dat
}
```

прочитано
фактически

Работа с файлами

Чтение данных:

```
fp = fopen( "input.dat", "rb" );  
if ( fp == NULL ) {  
    printf("Файл открыть не удалось.");  
    return;  
}  
n = fread ( A, sizeof(int), N, fp );  
if ( n < N ) printf("Не хватает данных в файле");  
fclose ( fp );
```

критическая
ошибка

некритическая
ошибка

Запись данных:

```
fp = fopen( "output.dat", "wb" );  
fwrite ( A, sizeof(int), n, fp );  
fclose ( fp );
```

СКОЛЬКО
прочитали

Задания

- «4»:** В текстовом файле `input.txt` записан массив целых чисел. Отсортировать его и записать в **двоичный** файл `output.dat`.
- «5»:** В текстовых файлах `input1.txt` и `input2.txt` записаны два массива. Объединить их в один массив, отсортировать и записать результат в **двоичный** файл `output.dat`.

Конец фильма
