

## Семинар #2: Массивы. Домашнее задание.

### Задача 1. Операции над массивом

Во всех подзадачах этой задачи вам нужно изменить массив **a** и, возможно, размер **n** между считыванием массива и его печатью. Каждая программа должна иметь такой вид:

```
#include <stdio.h>
int main()
{
    int a[1000];
    int n;
    scanf("%i", &n);
    for (int i = 0; i < n; ++i)
        scanf("%i", &a[i]);

    // ||||| Ваш код между считыванием и печатью массива |||||

    // |||||
    for (int i = 0; i < n; ++i)
        printf("%i ", a[i]);

    printf("\n");
}
```

Внутри вашего кода нужно считать дополнительные данные и изменить массив и переменную **n**.

1. **Удвоение массива:** Нужно увеличить массив **a** в 2 раза, заполнив новую часть копией массива **a**. Предполагается, что количество места в массиве (1000) больше чем  $2n$ , то есть места хватит. Не забудьте изменить переменную **n**.

ВХОД	ВЫХОД
4	0 1 2 3 0 1 2 3
0 1 2 3	
3	6 4 3 6 4 3
6 4 3	

2. **Вставка:** На вход подаётся массив, новый элемент массива и индекс – положение в массиве, после которого нужно вставить элемент. Чтобы освободить место в массиве нужно передвинуть часть элементов вправо. Предполагается, что количество места в массиве (1000) больше чем **n**, то есть места на 1 элемент хватит. Будьте осторожны, не перепишите элементы массива при их перемещении. Не забудьте изменить переменную **n**.

ВХОД	ВЫХОД
6	0 1 2 9 3 4 5
0 1 2 3 4 5	
9 2	
2	1 5 4
1 5	
4 1	

3. **Удаление:** На вход подаётся массив и индекс элемента, который нужно удалить. При этом понадобится передвинуть часть элементов влево.

ВХОД	ВЫХОД
6	0 1 2 4 5
0 1 2 3 4 5	
3	
2	5
1 5	
0	

4. **Удаление подмассива:** На вход подаётся массив и подмассив(2 индекса). Нужно удалить этот подмассив из массива. Постарайтесь написать как можно более эффективный код. Например, каждый элемент нужно переместить только 1 раз.

ВХОД	ВЫХОД
6	0 4 5
0 1 2 3 4 5	
1 4	
9	2 1
9 8 7 6 5 4 3 2 1	
0 7	

5. **Удаление отрицательных элементов:** Удалите все отрицательные элементы из массива. Постарайтесь написать как можно более эффективный код. Например, каждый элемент нужно переместить только 1 раз.

ВХОД	ВЫХОД
6	0 2 5
0 -1 2 -3 -4 5	
2	9
9 -5	

6. **Разделение на чётные/нечётные:** Переставьте элементы массива **a** так, чтобы сначала в нём шли нечётные элементы, а потом чётные. Причём порядок следования внутри чётной или нечётной части не важен. Эту задачу можно решить с использованием дополнительных массивов, а можно и без них.

ВХОД	ВЫХОД
7	1 3 5 0 4 2 6
0 1 2 3 4 5 6	
9	9 7 5 3 1 8 2 4 6
9 8 7 6 5 4 3 2 1	
2	1 2
2 1	

7. **Раздвоение:** Увеличьте массив в 2 раза, раздвоив каждый элемент. Постарайтесь написать более оптимальный код без использования дополнительного массива.

ВХОД	ВЫХОД
6	0 0 1 1 2 2 3 3 4 4 5 5
0 1 2 3 4 5	
1	1 1
1	

8. **Циклический сдвиг:** На вход подаётся массив и целое положительное число **k** нужно циклически сдвинуть массив на **k** элементов вправо.

ВХОД	ВЫХОД
6	4 5 0 1 2 3
0 1 2 3 4 5	
2	
6	1 2 3 4 5 0
0 1 2 3 4 5	
5	

*Подсказка:* Новое положение **i**-го элемента в массиве будет задаваться формулой  $(i + k) \% n$ . Эту задачу проще всего решить с использованием дополнительного массива, но можно и без него.

# Бинарный поиск на отсортированном массиве

Если известно, что массив уже отсортирован, то многие задачи на таком массиве можно решить гораздо проще и/или эффективней. Например, просто найти минимум, максимум и медианное значение. Одной из задач, которая быстрее решается на отсортированном массиве – это задача поиска элемента в массиве. Если массив отсортирован, то решить эту задачу можно гораздо быстрее чем простой обход всех элементов.

Предположим, что массив отсортирован по возрастанию и надо найти элемент  $x$  в этом массиве или понять, что такого элемента в массиве не существует. Для этого мы мысленно разделим массив на 2 части:

1. Элементы, которые меньше, чем  $x$
2. Элементы, которые больше или равны  $x$

Затем введём две переменные-индекса  $l$  и  $r$ . В начале работы алгоритма индекс  $l$  будет хранить индекс фиктивного элемента, находящегося до первого (то есть  $l = -1$ ), а индекс  $r$  будет хранить индекс фиктивного элемента, находящимся после последнего (то есть  $r = n$ ).

На каждом шаге алгоритма мы будем брать середину между индексами  $l$  и  $r$  и передвигать к этой середине или индекс  $l$  или индекс  $r$ . При этом при изменении индексов должны соблюдаться условия:

```
a[l] < x
a[r] >= x
```

Алгоритм закончится тогда, когда разница между индексами не станет равным 1, то есть не станет  $r == l + 1$ . И так как  $a[l] < x$  и  $a[r] >= x$ , то если элемент  $x$  в массиве существует, то его индекс равен  $r$ .

Код для поиска в отсортированном массиве бинарным поиском:

```
#include <stdio.h>

int main()
{
    int n;
    int a[1000];
    scanf("%i", &n);
    for (int i = 0; i < n; ++i)
        scanf("%i", &a[i]);

    int x;
    scanf("%i", &x);

    int l = -1, r = n;
    while (r > l + 1)
    {
        int mid = (l + r) / 2;

        if (a[mid] >= x)
            r = mid;
        else
            l = mid;
    }

    if (r < n && a[r] == x)
        printf("Element found! Index = %i\n", r);
    else
        printf("Element not found!");
}
```

## Задача 2. Нижняя граница

Пусть дан массив и некоторое число  $x$ . Нижняя граница – это индекс первого элемент, который больше или равен  $x$ . Напишите эффективную программу, которая ищет нижнюю границу на отсортированном массиве. Если такого элемента нет, то нужно вернуть  $n$ .

ВХОД	ВЫХОД
7 1 1 1 2 2 5 6 2	3
7 0 1 1 2 6 6 9 3	4

ВХОД	ВЫХОД
5 1 2 3 4 5 5	4
5 1 1 1 1 1 1	0

ВХОД	ВЫХОД
3 2 2 6 1	0
3 2 2 6 9	3

# Матрицы

## Задача 3. Сумма столбцов

На вход поступают размеры матрицы `n` и `m` и элементы матрицы. Нужно найти сумму элементов в каждом столбце. Для этой задачи не нужно использовать двумерный массив, достаточно будет одномерного.

ВХОД	ВЫХОД
3 4	14 15 16 12
1 2 3 6	
6 5 4 2	
7 8 9 4	

## Задача 4. Сортировка столбцов

На вход поступают размеры матрица `n` и `m` и элементы матрицы. Нужно отсортировать элементы в каждом столбце.

ВХОД	ВЫХОД
5 3	1 1 1
8 1 9	2 2 3
2 5 1	4 2 7
7 5 7	7 5 7
4 2 3	8 5 9
1 2 7	

ВХОД	ВЫХОД
2 6	5 2 1 3 1 2
6 2 8 3 2 4	6 4 8 5 2 4
5 4 1 5 1 2	

## Задача 5. Умножение матриц

На вход поступает число `n` и две квадратных матрицы размера `nхn`. Нужно перемножить эти матрицы и напечатать результат. Формула перемножения матриц:

$$C_{ij} = \sum_{k=0}^{n-1} A_{ik} \cdot B_{kj}$$

.

ВХОД	ВЫХОД
3	21 30 130
7 7 2	-6 21 82
1 8 3	48 12 -1
2 1 6	
5 2 9	
-4 2 11	
7 1 -5	

ВХОД	ВЫХОД
3	55 60 70
5 2 9	-4 -1 64
-4 2 11	40 52 -13
7 1 -5	
7 7 2	
1 8 3	
2 1 6	

## Работа с файлами

### Задача 6. Сортировка по сумме цифр

В файле `numbers.txt` хранится 10000 чисел. Читайте эти числа и отсортируйте по сумме цифр. То есть число, у которого сумма цифр минимальна должно идти первым. Сохраните результат в файле `sorted.txt`. Для считывания из файла и печати в файл используйте метод перенаправления потока.

### Задача 7. Умножение матриц из файла

В файлах `matA.txt` и `matB.txt` сохранены матрицы  $10 \times 10$ . Читайте эти матрицы, перемножьте их и сохраните в файле `matC.txt`. Читать их можно методом перенаправления потока из файла `combinedAB.txt`. В результате должна получиться такая матрица:

$$\begin{pmatrix} 259 & -15 & 237 & 257 & 231 & 67 & 237 & -64 & 152 & 363 \\ 555 & 233 & 539 & 188 & 356 & 325 & 423 & -47 & 123 & 387 \\ 497 & 512 & 572 & 95 & 619 & 155 & 414 & 207 & 203 & 217 \\ 455 & 280 & 675 & 354 & 664 & 346 & 483 & 177 & 168 & 404 \\ 264 & 182 & 272 & 290 & 474 & -33 & 234 & 99 & 379 & 156 \\ 272 & 180 & 469 & 286 & 326 & 282 & 325 & 215 & 195 & 231 \\ 421 & 363 & 475 & 506 & 359 & 481 & 468 & 101 & 325 & 328 \\ 384 & 218 & 567 & 395 & 475 & 488 & 361 & 168 & 291 & 298 \\ 387 & 297 & 480 & 170 & 318 & 423 & 483 & 10 & -17 & 406 \\ 193 & 241 & 486 & 38 & 403 & 146 & 286 & 326 & 212 & 172 \end{pmatrix}$$

Для считывания из файла и печати в файл используйте метод перенаправления потока.