

Технология программирования на ЭВМ

Массивы

Баев А.Ж.

Казахстанский филиал МГУ

02 ноября 2018

Массив из 5 чисел типа `int`.

```
int a[5];
```

`a[0]` — первый элемент

`a[1]` — второй элемент

`a[2]` — третий элемент

`a[3]` — четвертый элемент

`a[4]` — пятый элемент (последний).

Операции с элементами массива

Изначально массив не инициализирован — там лежат «мусорные» значения.

```
a[0] = 20;  
a[1] = 18;  
a[2] = 0;  
a[3] = a[0];  
a[4] = a[3] + 1;
```

Имя	a[0]	a[1]	a[2]	a[3]	a[4]
Значение	20	18	0	20	19

Инициализация

С явным указанием размера:

```
int a[5] = {20, 18, 0, 20, 19};
```

Без явного указания размера (размер будет вычислен автоматически);

```
int a[] = {20, 16, 0, 20, 17};
```

Инициализация

Что будет, если размер больше, чем значений?

```
int a[5] = {20, 18};
```

Что будет, если размер меньше, чем значений?

```
int a[5] = {20, 18, 0, 20, 19, 0};
```

Читать: stackoverflow.com

Тип статического массива

Даны 2 массива

```
int a[5];  
char s[5];
```

Их тип соответственно:

```
(int [5])  
(char [5])
```

Размер статического массива

Даны 2 массива

```
int a[5];  
char s[5];
```

Их размер соответственно:

```
size_t a_size = sizeof(a);  
size_t s_size = sizeof(s);  
printf("%lu", a_size);  
printf("%lu", s_size);
```

Тип `size_t` синоним типа `unsigned long int`.

Индексация массива начинается с нуля.

Причина: удобная адресная арифметика (будет подробно при динамических массивах).

Пример. Необходимо считать 5 целых чисел и распечатать их в обратном порядке.

```
1   int i, a[5];  
2   for (i = 0; i < 5; i++) {  
3       scanf("%d", &a[i]);  
4   }  
5   for (i = 4; i >= 0; i--) {  
6       printf("%d□", a[i]);  
7   }
```


Некорректные индексы.

Обращение к некорректным индексам может привести к неопределенному поведению программы:

UndefinedBehavior

Некорректные индексы.

Первый вариант.

```
1 #include <stdio.h>
2 int main() {
3     int a[16];
4     a[2050] = 10;
5     printf("a[2050] = %d", a[2050]);
6     return 0;
7 }
```

Программа падает во время исполнения с ошибкой сегментации (segmentation fault).

```
$gcc prog.c -o prog
$./prog
segmentation fault (core dumped)
```

В терминале появится «ошибка сегментирования» (стек

Некорректные индексы.

Второй вариант. Портятся значения других переменных.

```
1 #include <stdio.h>
2 int main() {
3     int a[16];
4     int b[16];
5     a[16] = 10;
6     print("b[0] = %d\n", b[0]);
7     return 0;
8 }
```

Вывод:

```
$gcc prog.c -o prog
$./prog
b[0] = 10
```

Некорректные индексы.

Третий вариант (самый ужасный). Всё иногда работает!

```
1 #include <stdio.h>
2 int main() {
3     int a[16];
4     a[2050] = 10;
5     print("a[2050] = %d\n", a[2050]);
6     return 0;
7 }
```

Вывод:

```
$gcc prog.c -o prog
$./prog
a[2050] = 10
$./prog
segmentation fault (core dumped)
```

Санитайзер — волшебное средство.

```
1 $ gcc prog.c -o prog -fsanitize=undefined
2 $ ./prog
3 prog.c:4:10: runtime error:
4 index 16 out of bounds for type 'int_[16]'
```

Сумма чисел.

Задача. Дано целое n от 1 до 100, далее n целых чисел.
Необходимо найти сумму чисел.

Ввод

```
5
5 4 3 2 1
```

Вывод

```
15
```

Сумма чисел.

```
1  #include <stdio.h>
2  int main() {
3      int n, i, a[100];
4      scanf("%d", &n);
5      for (i = 0; i < n; i++) {
6          scanf("%d", &a[i]);
7      }
8      int sum = 0;
9      for (i = 0; i < n; i++) {
10         sum += a[i];
11     }
12     printf("%d\n", sum);
13     return 0;
14 }
```

Дано целое число n от 1 до 100, далее n вещественных чисел. Необходимо вывести те числа, которые больше среднего арифметического.

Ввод

```
5
5 4 3 2 1
```

Вывод

```
5 4
```


Фильтр.

```
1      int n, i;
2      double a[100], mean = 0;
3      scanf("%d", &n);
4      for (i = 0; i < n; i++) {
5          scanf("%f", &a[i]);
6          mean += a[i];
7      }
8      mean /= n;
9      for (i = 0; i < n; i++) {
10         if (a[i] > mean) {
11             printf("%d□", a[i]);
12         }
13     }
```

Фибоначчи.

Задача. Дано целое n от 1 до 90. Необходимо вывести первые N чисел Фибоначчи, которые определяются рекуррентно:

$$\begin{cases} f_0 = 0, \\ f_1 = 1, \\ f_i = f_{i-1} + f_{i-2}, i \geq 2. \end{cases}$$

Ввод

7

Вывод

0 1 1 2 3 5 8

Фибоначчи.

```
1  #include <stdio.h>
2  int main() {
3      int n, i, f[90] = {0, 1};
4      scanf("%d", &n);
5      for (i = 2; i < n; i++) {
6          f[i] = f[i - 1] + f[i - 2];
7      }
8      for (i = 0; i < n; i++) {
9          printf("%d□", f[i]);
10     }
11     putchar('\n');
12     return 0;
13 }
```

Позиции всех максимумов.

Задача. Дано целое n от 1 до 1000. Далее n вещественных чисел от -10^9 до 10^9 . Найти позиции всех максимумов.

Ввод

```
5
5.5 3.1 5.5 5.5 1.2
```

Вывод

```
1 3 4
```

Решается в 2 прохода:

- 1) ищем максимум,
- 2) выводим все позиции, в которых элементы равны максимуму (нумерация на выводе будет на 1 больше, чем нумерация в массиве).

Позиции всех максимумов.

```
1  double a[1000];
2  int i, n;
3  scanf("%d", &n);
4  for (i = 0; i < n; i++)
5      scanf("%f", &a[i]);
6
7  double max = a[0];
8  for (i = 1; i < n; i++)
9      if (max < a[i])
10         max = a[i];
11
12  for (i = 0; i < n; i++)
13      if (a[i] == max)
14         printf("%d ", i + 1);
```

Подсчет.

Задача. Дано целое n от 1 до 100. Далее n целых чисел от 0 до 999. Напечатать только те числа, которые есть в этом массиве, по возрастанию.

Ввод

```
6
5 1 5 2 5 1
```

Вывод

```
1 2 5
```

Подсчет.

Метод «подсчета»: в дополнительном массиве `count` подсчитывать количество соответствующих элементов в исходном массиве. То есть для чисел 5, 1, 5, 2, 5, 1 массив для подсчета будет следующим:

x	0	1	2	3	4	5	6	...
count[x]	0	2	1	0	0	3	0	...

Как только мы встречаем очередной элемент, который равен к примеру `x`, то увеличиваем `count[x]` на один:

```
count[x] = count[x] + 1;
```

Подсчет.

```
1      int n, i, x, count[1000] = {0};
2      scanf("%d", &n);
3      for (i = 0; i < n; i++) {
4          scanf("%d", &x);
5          count[x] = count[x] + 1;
6      }
7
8      for (x = 0; x <= 1000; x++) {
9          if (count[x] > 0) {
10             printf("%d□", x);
11         }
12     }
13     putchar('\n');
```


Матрицы и многомерные массивы.

Матрицы индексируются аналогично с нуля. Например, так объявляется матрица размера 2×3 (2 строки и 3 столбца) и заполняются два её угловых элемента:

```
1  int a[2][3];  
2  a[0][0] = 1;  
3  a[1][2] = 6;
```

1	???	???
???	???	6

Матрицы и многомерные массивы.

Сгенерируем матрицу $a_{ij} = i \cdot j$ размера 4×4 .

```
1  int a[4][4];  
2  for (i = 0; i < n; i++) {  
3      for (j = 0; j < n; j++) {  
4          a[i][j] = (i + 1) * (j + 1);  
5      }  
6  }
```

1	2	3	4
2	4	6	8
3	6	9	12
4	8	12	16

Матрицы и многомерные массивы.

Инициализировать матрицу можно при объявлении:

1

```
int a[2][3] = {{1, 2, 3}, {4, 5, 6}};
```

1	2	3
4	5	6