

# Массивы и указатели

Семестр 1

Семинар 2

# Подсчет слов (повтор)

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

```
int main () {
```

```
    char c, nl = 0;
```

```
    while ( (c = getchar() ) != '\n' ) {
```

```
        if (c == ' ')
```

```
            nl ++;
```

```
    }
```

```
    printf ("%d \n", nl);
```

```
    return 0;
```

```
}
```

char хранит целое число и состоит из одного байта. По таблице ASCII его можно расшифровать как символ. Форматирующая последовательность %c

Функция считывания нажатого на клавиатуре символа

Между кавычками стоит пробел

# Задачи (повтор)

- Вывести на экран все множители числа  $n$ . Ввод  $n$  с клавиатуры.
- С экрана вводится последовательность нулей и единиц до введения любого другого символа. Подсчитать количество единиц.
- Вывести на экран ASCII-код цифры 0.
- Определить количество цифр во введенной последовательности символов.
- Вывести  $n$ -е простое число. Ввод  $n$  с клавиатуры.

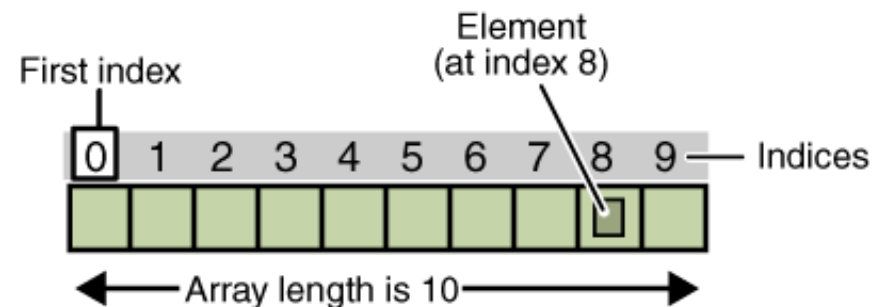
# Определение массива

- Массив – набор **однотипных** элементов **фиксированной** длины
  - нельзя объединять в массив элементы разного типа
  - длина массива – константа
  - сообщений о выходе за границы массива **нет**
  - память выделяется 1 куском

- Пример использования:

```
int a[10];
```

```
a[0] = -17 + a[3];
```



# Многомерные массивы

```
int a [3];  
for (i = 0; i < 3; i++)  
    a[i] = 0;
```

```
int m [3][4];  
for (i = 0; i < 3; i++)  
    for (j = 0; j < 4; j++)  
        m[i][j] = 0;
```

a[0][0]	a[0][1]	a[0][2]	a[0][3]	...	a[0][n]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	...	a[1][n]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	...	a[2][n]
...	...	...	...	...	...
a[m][0]	a[m][1]	a[m][2]	a[m][3]	...	a[m][n]

# Явная инициализация

- `int a [ ] = { 23, 7, 144 };`
- `int a [3] = { 23, 7, 144 };`  
`int a [4] = { 23, 7, 144 }; // можно`  
`int a [2] = { 23, 7, 144 }; // ошибка`
- `char str [ ] = {'H', 'e', 'l', 'l', 'o', '\0'};`  
`char str [ ] = "Hello";`  
`int x = strlen (str); // 5`
- `int a [5] = {0, 1, 2}; // a[2] = a[3] = a[4] = 2`  
`int a[10000] = {0}; //  $\forall i \in [0, 9999]: a[i] == 0$`
- `int a [3][4] = {`

<code>{13,</code>	<code>-8,</code>	<code>4, 12}</code>	<code>,</code>
<code>{-7,</code>	<code>-1,</code>	<code>14, 3}</code>	<code>,</code>
<code>{ 1,</code>	<code>-4,</code>	<code>2, 11}</code>	<code>};</code>

# Размер массива

```
int main ( ) {  
    int a [10], i;  
    for ( i = 0; i < 10; i++)  
        a [ i ] = 10 * i;  
    return 0;  
}
```

```
#define MAX_SIZE 10
```

```
int main ( ) {  
    int a [MAX_SIZE], i;  
    for ( i = 0; i < MAX_SIZE; i++)  
        a [ i ] = 10 * i;  
    return 0;  
}
```

# Переменный размер массива

```
int n;  
scanf("%d", &n);  
int arr[n]; // ошибка
```

Взять заведомо большую  
длину массива:

```
int arr[10000], n;  
scanf("%d", &n);
```

Далее используем только  
первые n элементов

Динамическая  
память



# Выход за границу массива

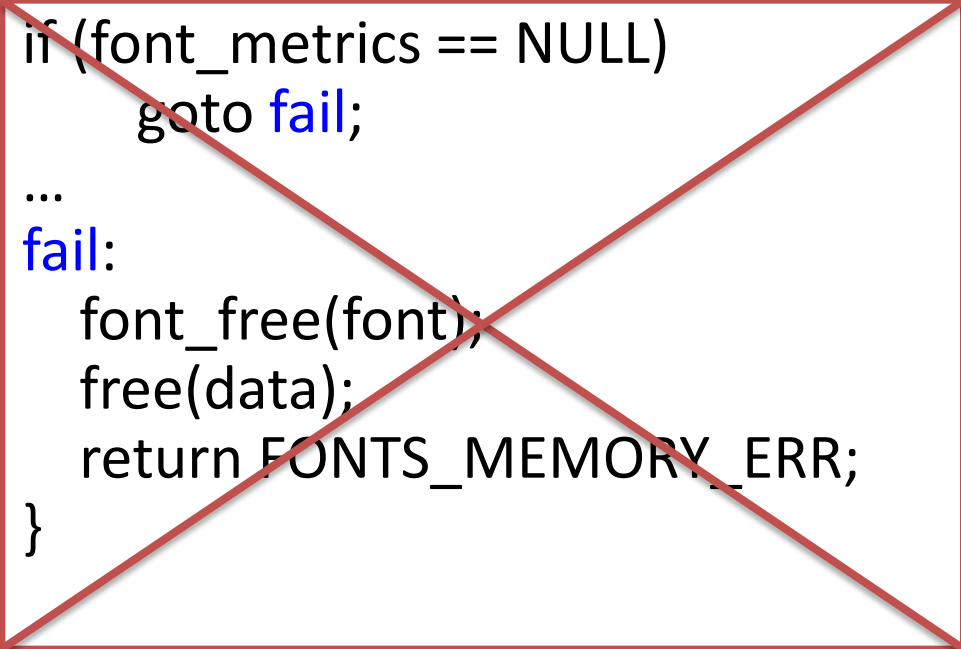
```
#include <stdio.h>

int main () {
    int a[3] = {0, 1, 2}, b = 5, c = 10;
    printf("a[3] = %d\n a[4] = %d", a[3], a[4]);
    a[3]++;
    a[4]*=2;
    printf("b = %d\n c = %d", b, c);
    return 0;
}
```

# break и continue

- **break** – выйти из ближайшего цикла (передать управление на первый оператор после цикла)
- **continue** – продолжить цикл (передать управление на проверку выражения цикла).
- **goto** *метка*;  
*метка* :

# Использование goto



```
if (font_metrics == NULL)
    goto fail;

...
fail:
    font_free(font);
    free(data);
    return FONTS_MEMORY_ERR;
}
```

goto не нужен в C

# Произвольные числа в С

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

```
#include <time.h>
```

```
int main()
```

```
{
```

```
    srand(time(NULL));
```

```
    printf("random 0 – 99: %d\n",rand()%100);
```

```
    printf("random 0 – 9: %d\n",rand()%10);
```

```
    printf("random 0 – 1: %d\n",rand()%2);
```

```
    return 0;
```

```
}
```

# Задачи

- Поиск максимума в массиве целых чисел.
- Ввести матрицу с клавиатуры и:
  - найти максимальное число в матрице
  - найти максимальное число по строкам
  - найти максимальное число по столбцам

Ожидаемая печать:

13	-8	4	12	-> 13
-7	-1	14	3	-> 14
1	-4	2	11	-> 11

-----

13	-1	14	12	14
----	----	----	----	----

- Напечатать все простые числа до N включительно.  
Использовать алгоритм «решето Эратосфена»

# RAM - это массив байт

- Память - это массив байт.
- Каждый байт имеет свой индекс в массиве памяти. Назовем индекс **адресом**.
- **RAM** - random-access memory, запоминающее устройство с произвольным доступом. В нашем случае – оперативная память.



# Адреса и указатели

```
int x = 1, y = 2;
```

```
int * ip;    // ip имеет тип int *, ip - указатель на int
```

```
ip = & x;    // & - операция взятия адреса, теперь в  
             // ip хранится адрес x; иначе говоря,  
             // ip указывает на x
```

```
y = * ip;    // * - операция разыменования  
             // указателя, теперь y == 1.
```

```
* ip = 0;    // x == 0;
```

- **NULL** – нулевой указатель, константа. `NULL = 0`.

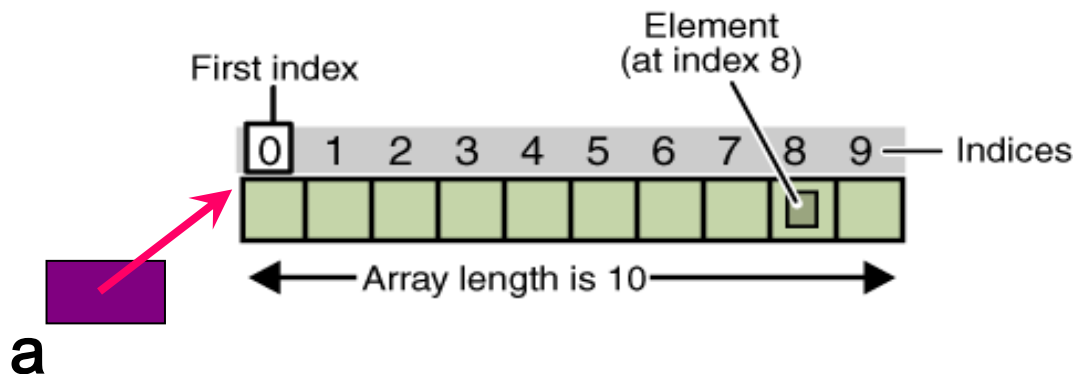
# Указатели и массивы

массив хранится одним куском памяти

- ```
int a[10];  
int * p;
```

```
p = &(a[0]);  
p = a;  
// эти две строчки делают одно  
// и то же: переменная p имеет  
// тип int* и является указателем  
// на нулевой элемент массива
```

```
a = p; //ошибка
```





# Адресная арифметика

```
int a[10], x;
```

```
int * p;
```

```
p = a;
```

```
x = *(p + 3);
```

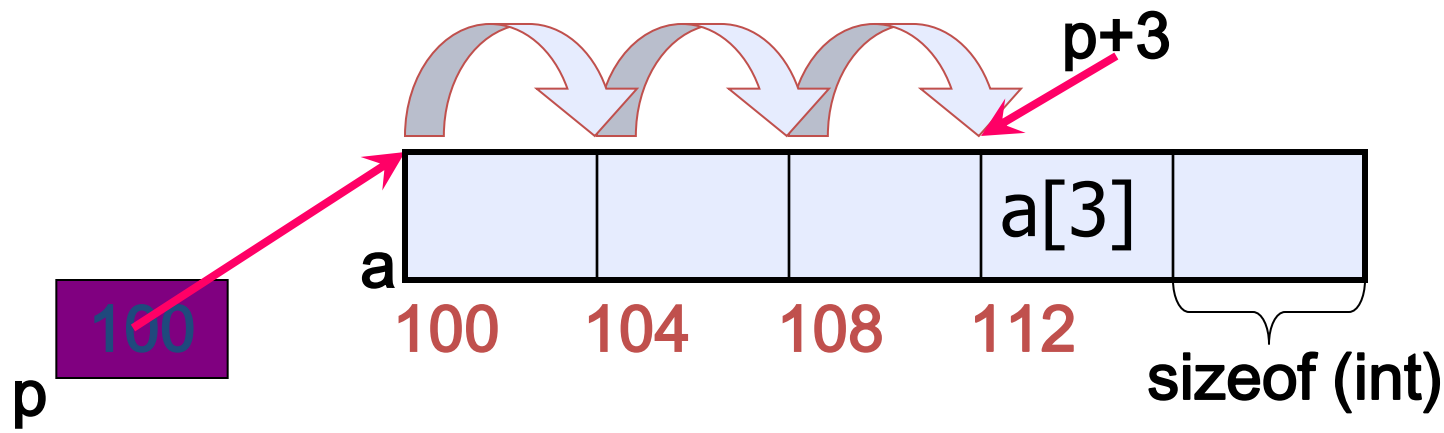
// хотим получить a[3]

// теперь x == a[3]: сдвиг на 3 ячейки,

// а не байта

// поэтому массивы нумеруются с нуля

// a[0] == \*a == \*(a+0)



# Операции с адресами

- адрес + целое = адрес
- адрес - адрес = целое
- адрес + адрес // ошибка
- адрес *сравнить* адрес
  - адрес == адрес
  - адрес != адрес
  - адрес < адрес // машинно-зависимое
  - адрес > адрес // машинно-зависимое

# Переменный размер массива

```
unsigned int n;  
scanf ("%u", &n);
```

```
int * p = malloc (n * sizeof (int) );
```

```
// выделили память размера n * sizeof(int) байт
```

```
// то же самое: p = calloc (n, sizeof(int) );
```

```
// p - указатель на начало выделенной памяти
```

После этого обращаемся к элементам как в обычном массиве:  
p [3] = 15;

```
free (p);    // освобождаем память
```

```
// когда станет не нужной
```

```
// в конце работы программы
```

# Задачи

- Выяснить, является ли строка палиндромом (А роза упала на лапу Азора).
  - чистый палиндром («qwertyytrewq»)
  - игнорируя пробелы («q wer ty ytrew q»)
- Переворот строки, введенной с клавиатуры.
- Сортировка одномерного массива целых чисел.
  - методом простого выбора
  - пузырьком.
- Перевернуть каждое второе слово в строке.