Лекция 15

Функции обработки массивов. Указатель на функцию

Преподаватель Палехова Ольга Александровна,
кафедра И5 БГТУ «Военмех»

Передача массива в функцию и из функции

Особенности:

- массивы никогда **не копируются**, все функции работают с оригиналом;
- в функцию передается **адрес** первого элемента массива и его **размер**
- объявить формальный параметр-указатель можно двумя способами:

```
тип_элемента_массива * имя_параметра
тип_элемента_массива имя_параметра []
```

- если по одному алгоритму обрабатываются массивы, элементы которых имеют разные типы, нужно написать несколько подобных функций
- если функция не должна изменять значения элементов массива, формальный параметр-указатель лучше объявить с квалификатором ***const***

Передача массива в функцию и из функции

Задача: Найти сумму максимального элемента целочисленного массива A(25) и максимального элемента целочисленной матрицы B(4x6). Если хотя бы в одном из массивов все элементы совпадают, вывести сообщение об этом, сумму не вычислять.

Анализ задачи: Для обоих массивов нужно выполнить ввод данных и поиск максимального значения, и то, и другое требует полного перебора. Если память под матрицу выделять одним блоком, можно использовать для обработки обоих массивов одни и те же функции.

Проблема: Что возвращать из функции поиска максимума, если все элементы одинаковые?

Идея: возвращать не значение, а адрес максимума. При отсутствии максимума возвращать *NULL*.

Пример. Сумма максимумов

```
#include <stdio.h>
#define N 25
#define K 4
#define M 6

void input_array (int*, int);
int* search_max_array (const int*, int);

int main()
{
    int a[N], b[K][M], *max_a, *max_b;
    printf ("Заполните массив A(%d)\n", N);
    input_array (a, N);
    printf ("Заполните матрицу B(%dx%d)\n", K, M);
    input_array (b[0], K*M);
    max_a = search_max_array (a, N);
    max_b = search_max_array (b[0], K*M);
    if ( max_a == NULL || max_b == NULL )
        printf (" Сумму максимумов вычислить невозможно\n");
    else
        printf ("Сумма максимумов = %d\n", *max_a + *max_b);
    return 0;
}
```

прототипы функций

вызовы функций

Пример. Сумма максимумов

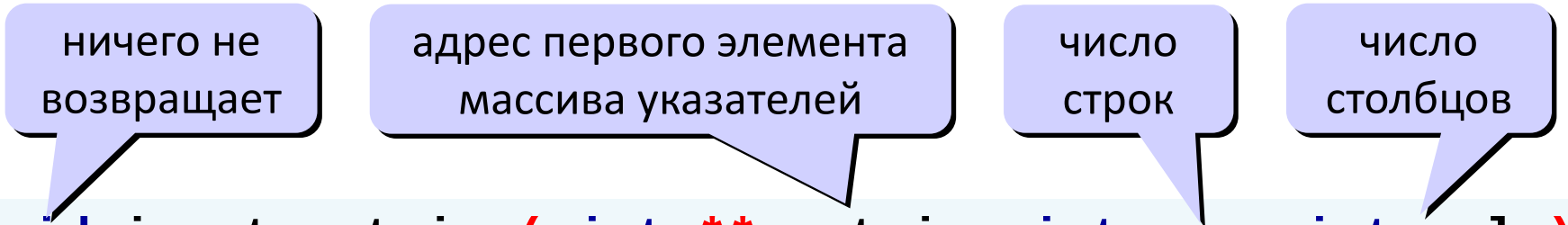
```
void input_array (int* a, int n)
{
    for (; n > 0 ; n-- )
        scanf ("%d", a++ );
}

int* search_max_array (const int* a, int n)
{
    int * pmax = a, flag = n;
    for ( a++ ; n > 1; n--, a++ )
        if ( *pmax < *a )
            pmax = a;
        else
            if ( *pmax == *a )
                flag--;
    if ( flag )
        return pmax;
    return NULL;
}
```

определения функций

Функция ввода матрицы

Задача. Написать функцию ввода элементов матрицы.




```
void input_matrix ( int ** matrix, int row, int col )
{
   int i, j;
   for ( i = 0 ; i < row ; i++ )
      for ( j = 0 ; j < col ; j++ )
         scanf ( "%d", matrix[i] + j );
}
```

Применение:

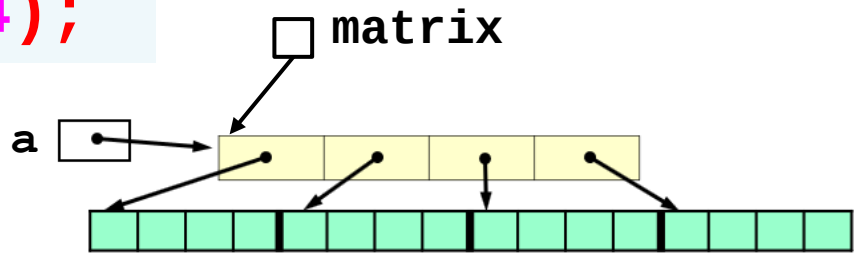
- ввод элементов динамической матрицы;
- ввод элементов динамического двумерного массива.

Вызов функции ввода матрицы

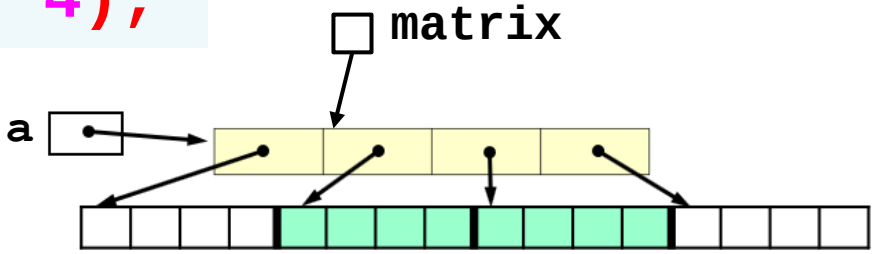
- для заполнения динамического двумерного массива или его части

```
int **a, i;
a = calloc (4, sizeof(int*));
a[0] = calloc (16, sizeof(int));
for ( i = 1; i < 4; i++ )
    a[i] = a[i-1] + 4;
```

```
input_matrix (a, 4, 4);
```



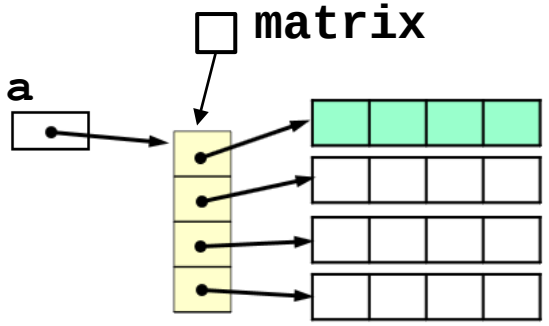

```
input_matrix (a+1, 2, 4);
```



Вызов функции ввода массива

- для заполнения динамической матрицы

```
int **a, i;
a = calloc (4, sizeof(int*));
for ( i = 0; i < 4; i++ )
    a[i] = calloc (4, sizeof(int));
```

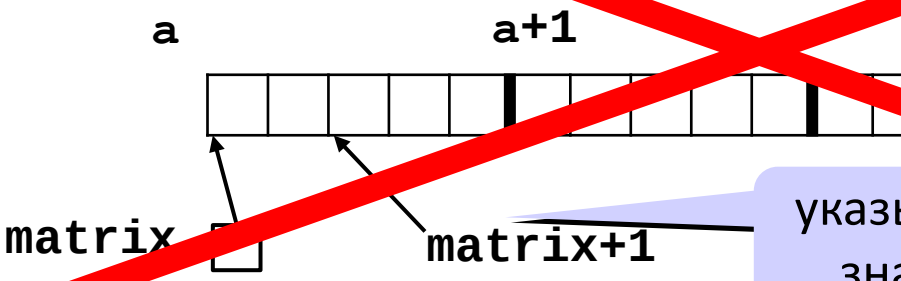



```
input_matrix (a, 4, 4);
```

? Можно заполнить статический двумерный массив?

```
int a[4][5];
input_matrix (a, 4, 5);
```

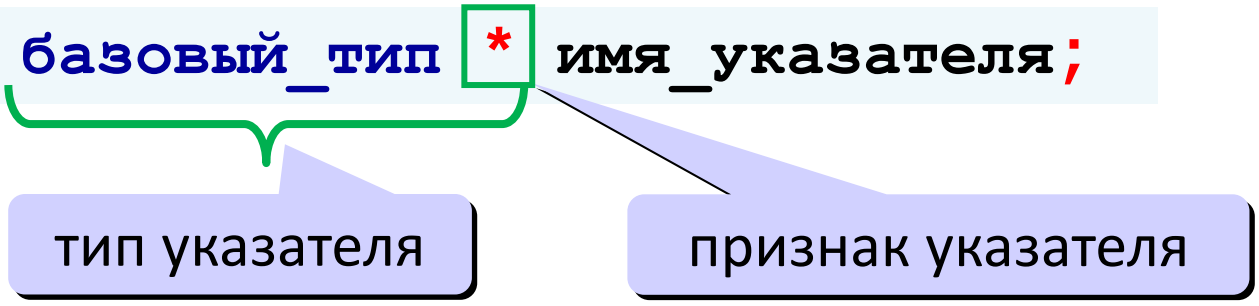
тип указателя int (*)[5]



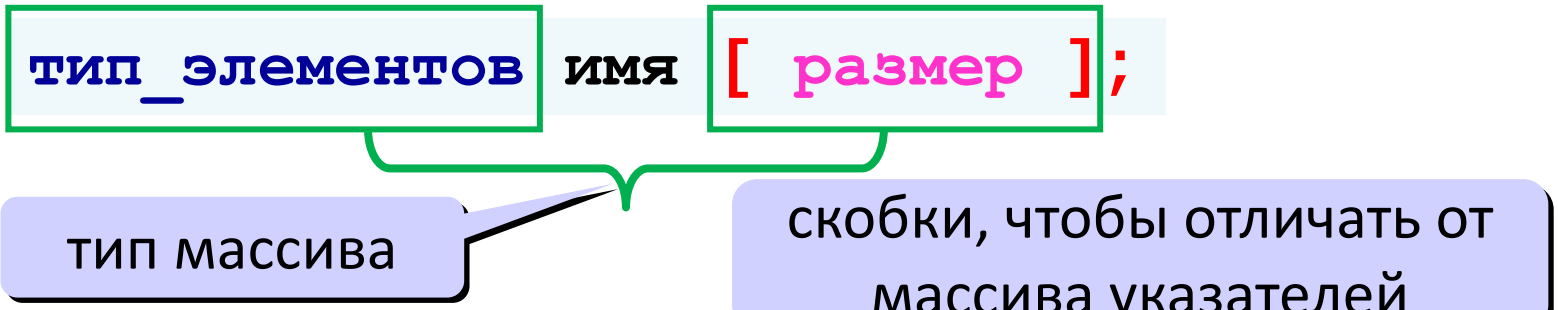
указывает не туда, воспринимает значение элемента как адрес

Еще раз об указателях

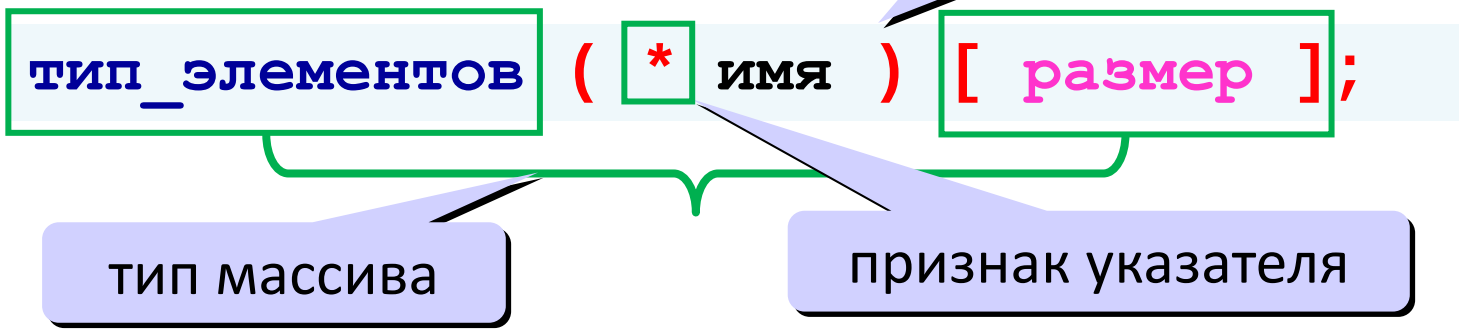
Общий вид объявления указателя :



Общий вид объявления массива :



Общий вид объявления указателя на массив :



Указатель на функцию

Общий вид объявления функции :

```
тип_результата имя_функции (спецификация_параметров);
```

тип функции

Общий вид объявления указателя на функцию:

```
тип_результата ( * имя ) (спецификация_параметров);
```

тип функции

скобки, чтобы отличать от функции, возвращающей адрес

признак указателя

Применение:

- передача функции в функцию через **параметры**;
- организация **массивов** указателей для обращения к списку однотипных функций в цикле.

Указатель на функцию

Что есть что?

```
double f1 (double);
double ( *f2 ) (double);
double * f3 (double);
double * ( *f4 ) (double);
int f5 (double);
int ( *f6 ) (double);
int f7 (int*);
int ( *f8 ) (int*);
double * f9 (int*);
double * ( *f10 ) (int*);
```

прототип функции

указатель на функцию

прототип функции

указатель на функцию

прототип функции

указатель на функцию

прототип функции

указатель на функцию

прототип функции

указатель на функцию

! **Имя функции** по сути является **указателем-константой** на ее начало

Указатель на функцию

Инициализация указателей на функции

```
double f1 (double);
double ( *f2 ) (double) = &f1;
double ( *f3 ) (double) = f1;
double ( *f4 ) (double) = sin;
double ( *f5[] ) (double) = {sin, cos, tan};
```

массив указателей на функции

Вызов функции через указатель на функцию

```
(*f2)(x)   ≡   f2(x)   ≡   f1(x)
(*f4)(x)   ≡   f4(x)   ≡   sin(x)
```

```
for ( i = 0 ; i < 3 ; i++ )
   f5[i](x)
```

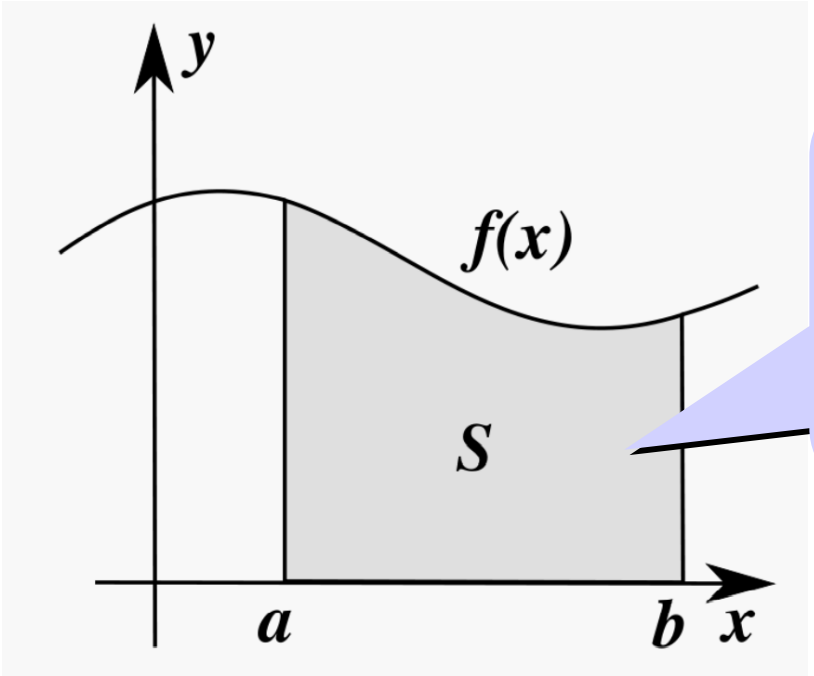
≡

```
sin(x)
cos(x)
tan(x)
```

обращение к элементу массива указателей на функции

Указатель на функцию в параметрах функций

Задача. Написать функцию для вычисления определенного интеграла вида $\int_a^b f(x)dx$

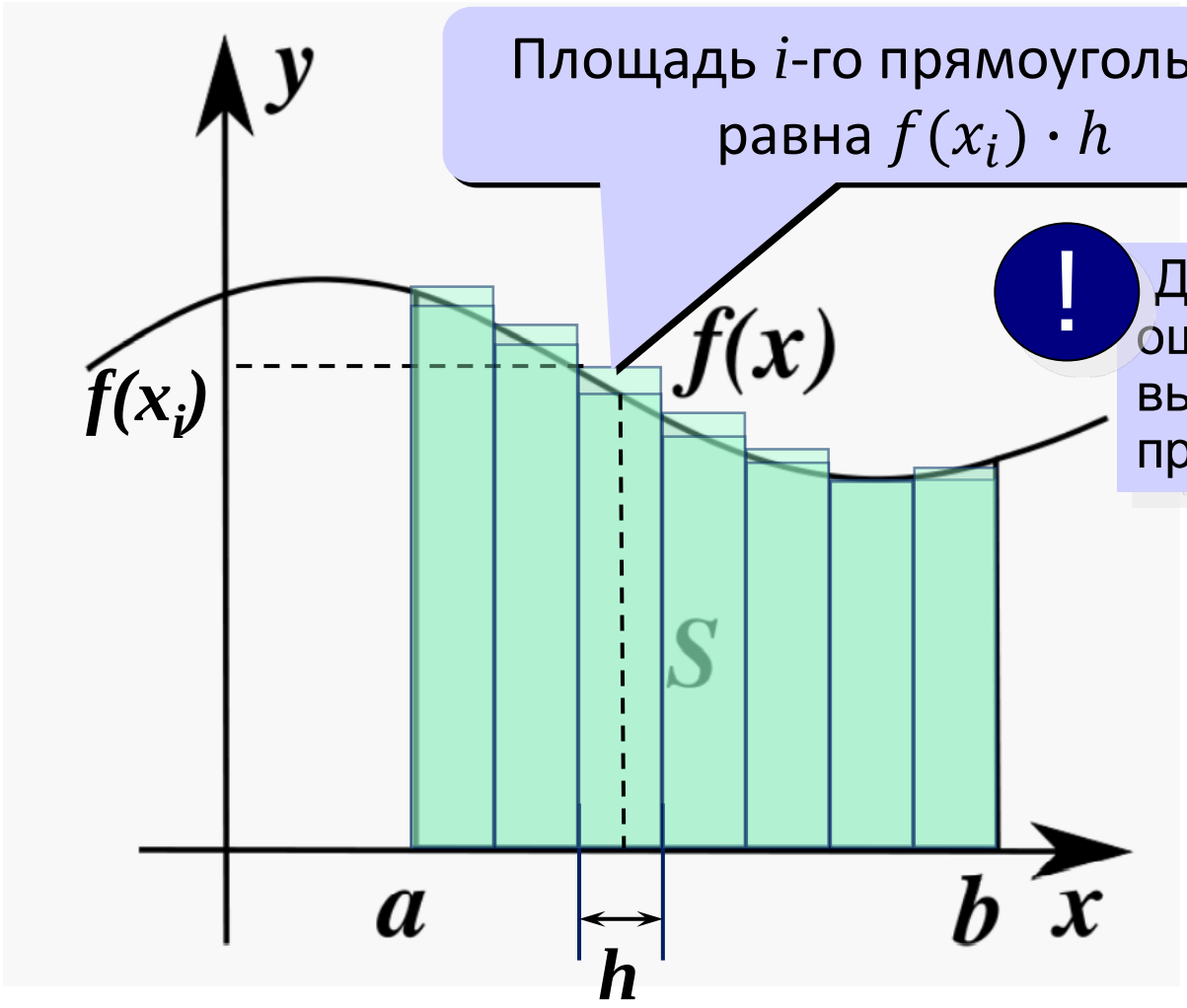


геометрический смысл определенного интеграла - значение интеграла равно площади под кривой, образованной графиком функции

Задача вычисления интеграла сводится к задаче вычисления площади фигуры.

Вычисление определенного интеграла

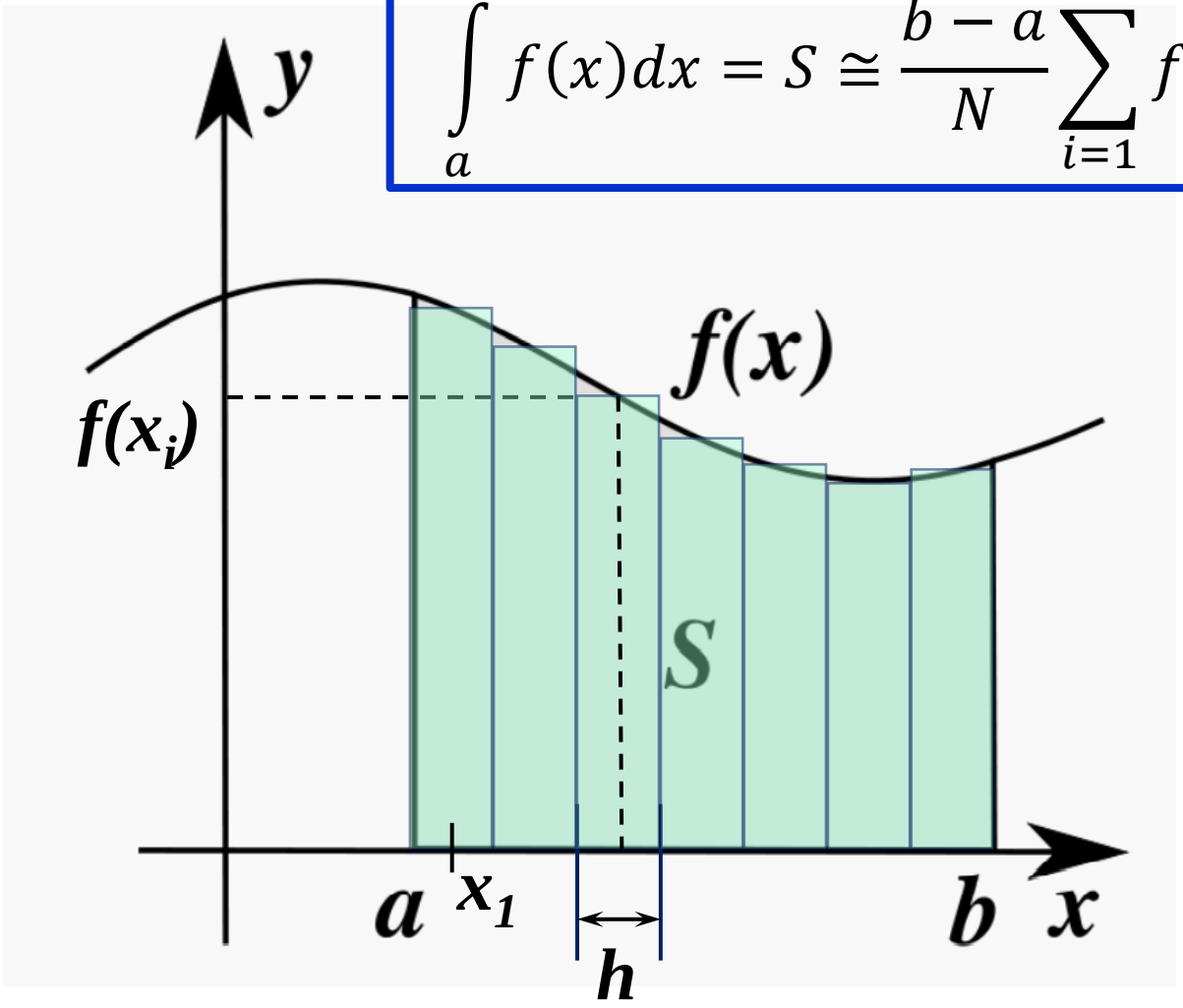
Идея: поделить искомую площадь на простые фрагменты, например, прямоугольники



Вычисление определенного интеграла

Делим интервал $[a, b]$ на N равных частей.

$$\int_a^b f(x)dx = S \cong \frac{b-a}{N}\sum_{i=1}^{N} f(x_i)$$



$$h = \frac{b-a}{N}$$

$$x_1 = a + \frac{h}{2}$$

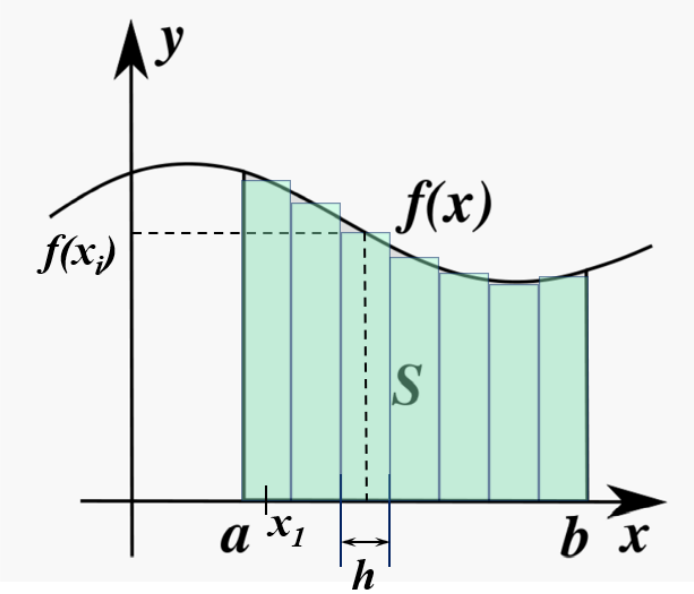
$$x_i = x_{i-1} + h$$

$$S_i = f(x_i) \cdot h$$

Вычисление определенного интеграла

Значение определенного интеграла можно приближенно вычислить по формуле

$$\int_a^b f(x)dx = S \cong \frac{b-a}{N}\sum_{i=1}^{N} f(x_i)$$



Особенности:

- алгоритм применим к любой функции, непрерывной на интервале [a, b];
- чем больше N, тем точнее результат

Указатель на функцию в параметрах функции

Задача. Написать функцию для вычисления определенного интеграла вида $\int_a^b f(x)dx$

```
double integral (double a, double b,
                 double (*f)(double))
{
    double h = ( b - a ) / 1000, s = 0;
    for ( a += h / 2 ; a < b ; a += h )
        s += f(a);
    return s * h;
}
```

указатель на функцию

вызов подынтегральной функции через указатель

Указатель на функцию в параметрах функции

Задача. Вычислить $\int_0^{\frac{\pi}{2}} \cos x \, dx$ и $\int_{-1}^{2} (2x^2 - 1) dx$

Прототип функции вычисления интеграла:

```
double integral (double, double,
                 double (*)(double));
```

Определение второй подынтегральной функции:

```
double parabola (double x)
{
   return 2 * x * x - 1;
}
```

формальный параметр – указатель на функцию

фактический параметр – имя функции

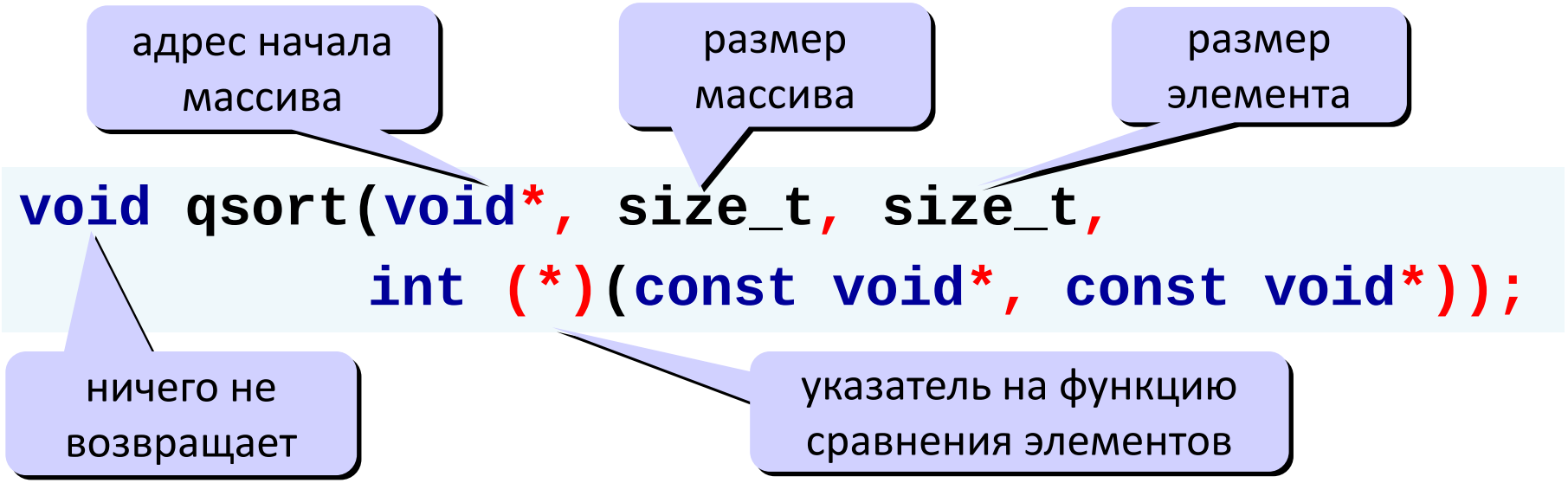
Вызов функции вычисления интеграла:

```
integral_1 = integral (0, M_PI_2, cos);
integral_2 = integral (-1, 2, parabola);
```

Использование функции *qsort()*

Задача: Отсортировать первую треть массива A(33) по возрастанию, оставшуюся часть массива и элементы строк матрицы B(6x8) - по убыванию, затем переставить строки матрицы по возрастанию элементов в последнем столбце.

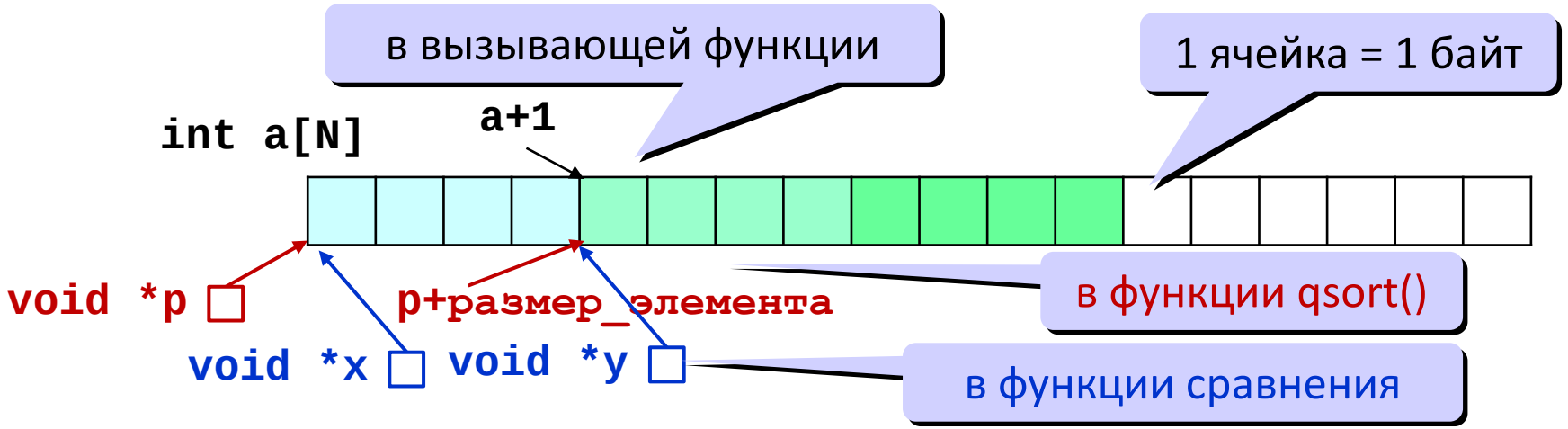
Прототип функции **qsort ()** (заголовочный файл *stdlib.h*):



Требования к функции сравнения элементов:

- **тип** функции должен **совпадать** с базовым типом указателя на функцию;
- результат должен быть >0, если элементы надо менять местами.

Использование функции *qsort()*



Функция сравнения целых чисел: первое значение больше второго:

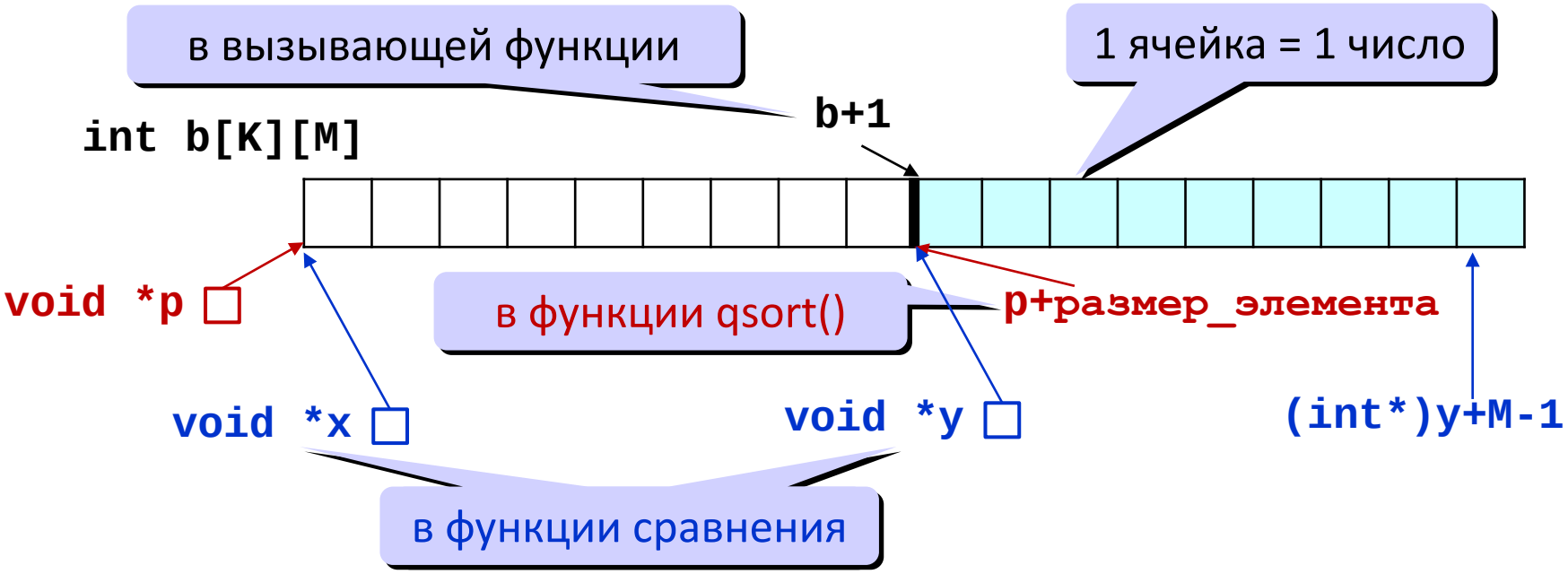
```
int comp_int_more (const void* x, const void* y)
{
    return  *(int*)x - *(int*)y ;
}
```

Функция сравнения целых чисел: первое значение меньше второго:

```
int comp_int_less (const void* x, const void* y)
{
    return *(int*)y - *(int*)x;
}
```

Использование функции *qsort()*

Функция сравнения строк: последний элемент в первой строке больше последнего элемента во второй строке:




```
int comp_row (const void* x, const void* y)
{
    return *((int*)x+M-1) - *((int*)y+M-1);
}
```

адрес последнего элемента в строке x

адрес последнего элемента в строке y

Пример: Использование функции *qsort()*

Задача: Отсортировать первую треть массива A(33) по возрастанию, оставшуюся часть массива и элементы строк матрицы B(6x8) - по убыванию, затем переставить строки матрицы по возрастанию элементов в последнем столбце.

Объявления:

```
#include <stdio.h>
#include <stdlib.h>
#define N 33
#define K 6
#define M 8

void input_array (int*, int);
void output_array (const int*, int, int);
int comp_int_more (const void*, const void*);
int comp_int_less (const void*, const void*);
int comp_row (const void*, const void*);
```

прототипы функций

Пример (продолжение)

Функция ***main()***:

```
int main()
{
    int a[N], b[K][M], i;
    printf ("Заполните массив A(%d)\n", N);
    input_array (a, N);
    printf ("Заполните матрицу B(%d,%d)\n", K, M);
    input_array (b[0], K*M);
    qsort (a, N/3, sizeof(int), comp_int_more);
    qsort (a + N/3, N-N/3, sizeof(int), comp_int_less);
    for ( i = 0 ; i < K ; i++ )
        qsort (b[i], M, sizeof(int), comp_int_less);
    qsort (b, K, sizeof(int)*M, comp_row);
    printf ("Результирующий массив A:\n");
    output_array (a, 1, N);
    printf ("Результирующая матрица B:\n");
    output_array (b[0], K, M);
    return 0;
}
```

адрес начала

количество элементов

размер элемента

функция сравнения

адрес начала второй трети

количество элементов

длина строки матрицы

Пример (продолжение)

Определения функций сравнения:

```
int comp_int_more (const void* x, const void* y)
{
    return *(int*)x - *(int*)y;
}

int comp_int_less (const void* x, const void* y)
{
    return *(int*)y - *(int*)x;
}

int comp_row (const void* x, const void* y)
{
    return *( (int*)x + M - 1 ) - *( (int*)y + M - 1 );
}
```

Пример (продолжение)

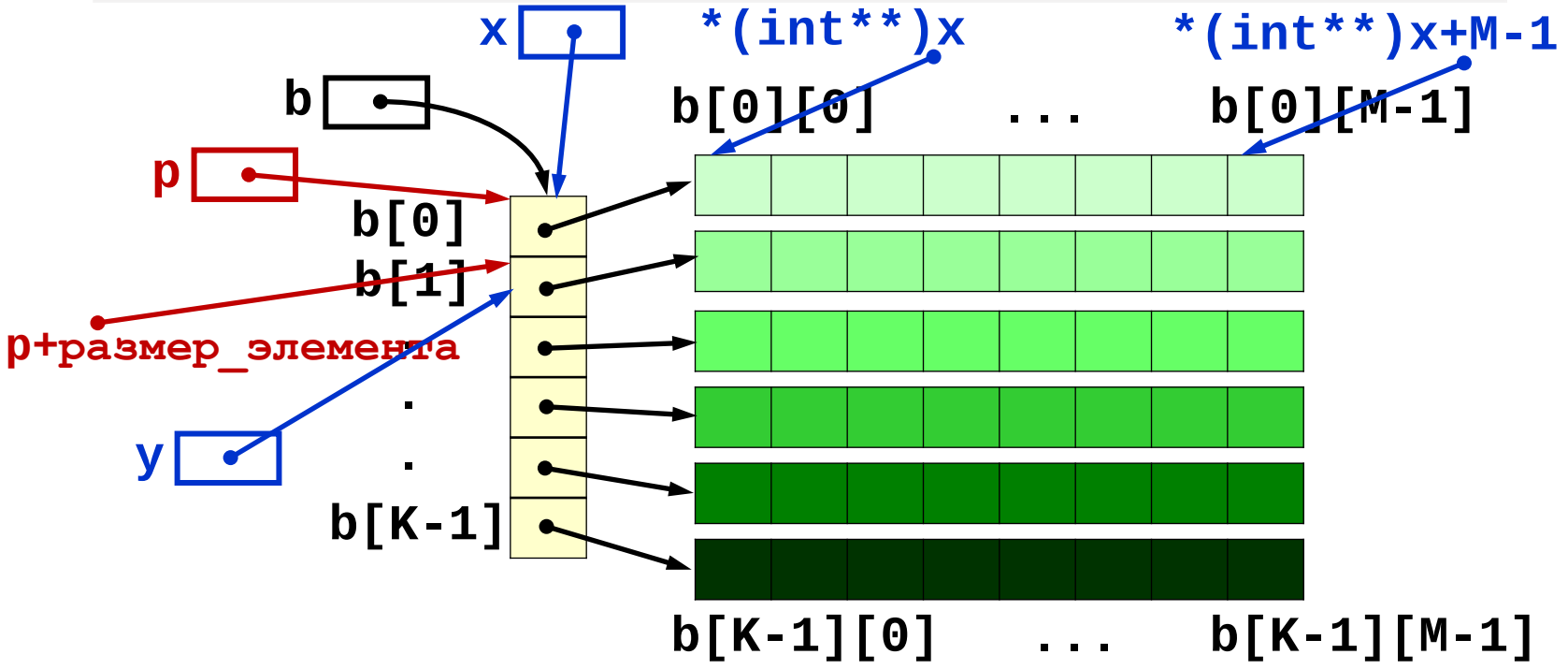
Определения функций ввода и вывода:

```
void input_array (int* a, int n)
{
    for (; n > 0 ; n-- )
        scanf ("%d", a++ );
}

void output_array (const int* a, int row, int col)
{
    int i;
    for (; row > 0 ; row-- )
    {
        for ( i = 0; i < col ; i++ )
            printf ("%8d", *a++);
        printf ("\n");
    }
}
```

Пример (продолжение)

? Как изменится программа, если матрицу B(KxM) программировать динамической матрицей?




```
int comp_row_p (const void* x, const void* y)
{
    return *(*(int**)x+M-1) - *(*(int**)y+M-1);
}
```

Пример (продолжение)

Функция ***main()***:

```
int main()
{
    int a[N], **b , i;
    printf ("Заполните массив A(%d)\n", N);
    input_array (a, N);
    printf ("Заполните матрицу B(%dx%d)\n", K, M);
 for ( i = 0 ; i < K ; i++ ) input_array (b[i], M);
    qsort (a, N/3, sizeof(int), comp_int_more);
    qsort (a + N/3, N-N/3, sizeof(int), comp_int_less);
    for ( i = 0 ; i < K ; i++ )
        qsort (b[i], M, sizeof(int), comp_int_less);
   qsort (b, K, sizeof(int*), comp_row_p);
    printf ("Результирующий массив A:\n");
    output_array (a, 1, N);
    printf ("Результирующая матрица B:\n");
 for ( i = 0 ; i < K ; i++ ) output_array (b[i], 1, M);
    return 0;
}
```

добавить выделение памяти

добавить освобождение памяти