

## Лекция 6

# Указатели и динамические массивы

# Статические и динамические массивы.

Все характеристики **статического массива** (имя массива, тип его элементов, размерность) полностью определялись при его объявлении и не могли меняться в течение выполнения программы. Память под статический массив выделялась при компиляции программы.

Однако, при решении многих задач необходимо, чтобы память для массива выделялась в процессе выполнения программы, т.е. потребности в памяти заранее не известны и не могут быть определены при объявлении массива. В этом случае используются **динамические массивы**. При работе с динамическими массивами обязательно выделение и освобождение выделенной динамической памяти.

Динамическое выделение памяти также позволяет эффективно использовать память компьютера.

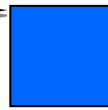
# Моделирование одномерных массивов

с использованием динамической памяти

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

a

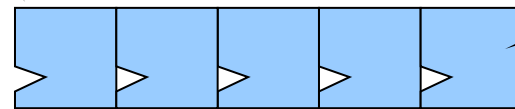
По умолчанию



int \*

$*(a+0) == a[0]$

a



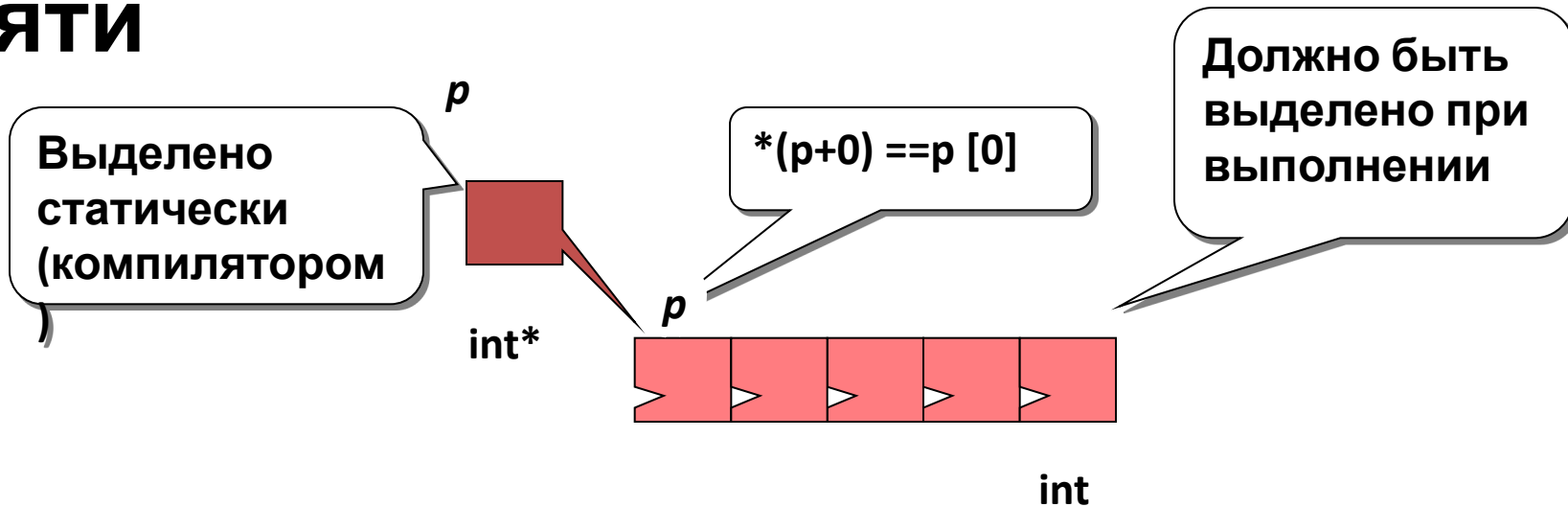
int

Выделено статически (компилятором)

$*(a+i) == a[i]$

# Моделирование одномерных массивов

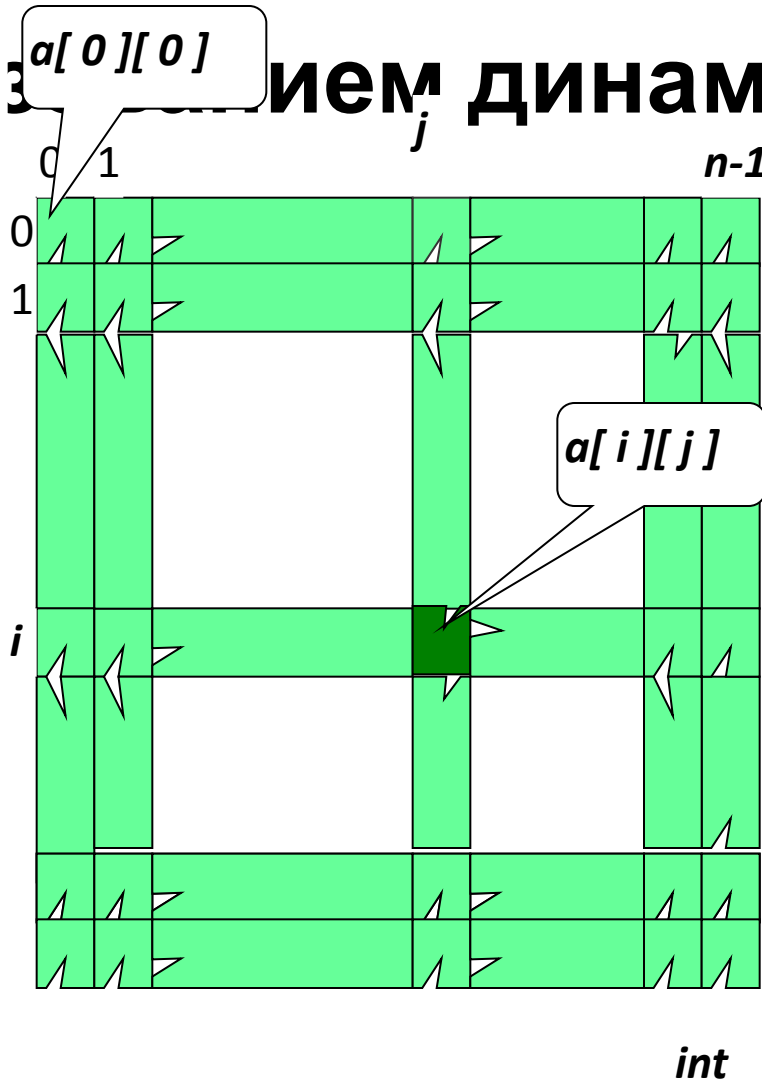
## с использованием динамической памяти



$$*(p+i) == p[i]$$

# Моделирование многомерных массивов

с использованием динамической памяти

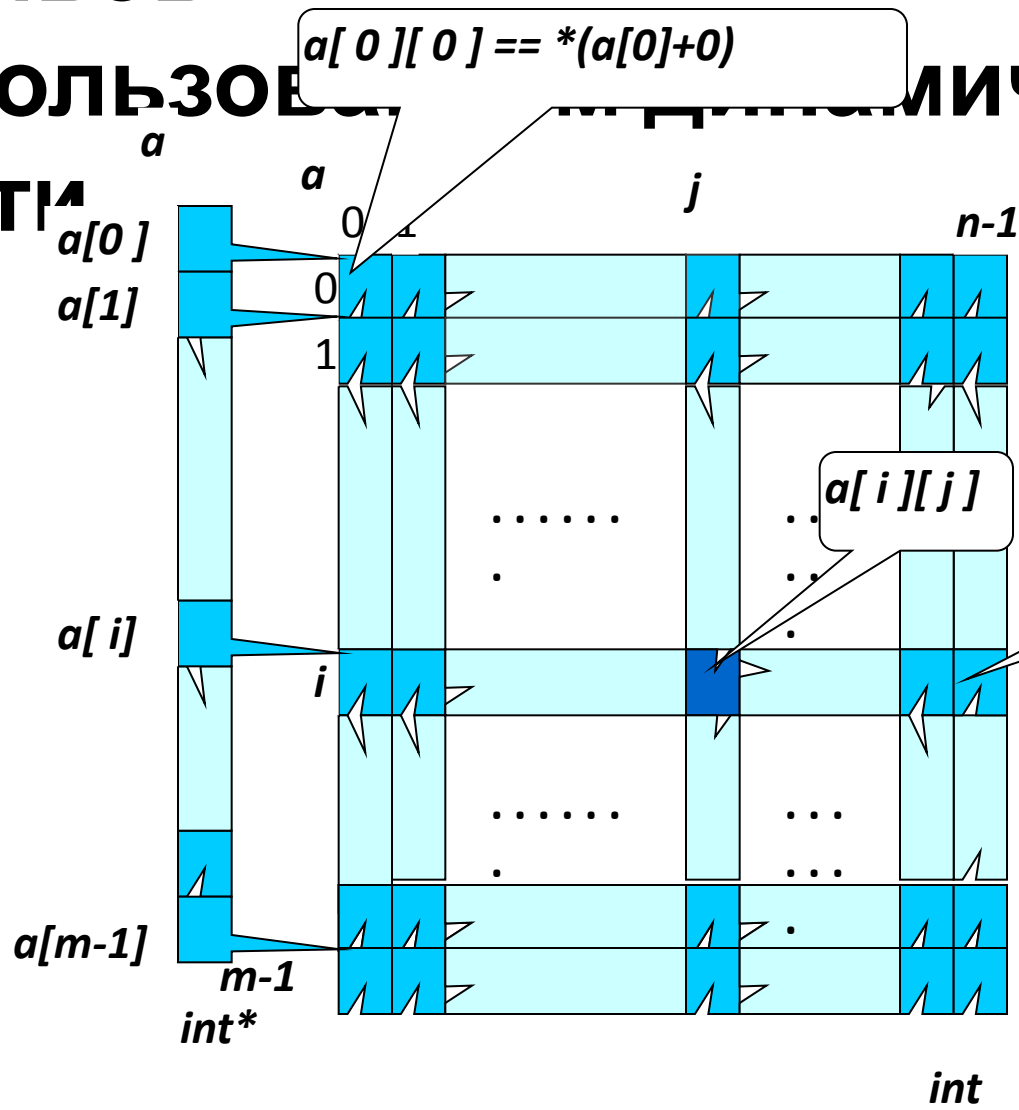


матрицей динамической

Это матрица в обычном смысле; ее элементы:  $a[0][0]$ ,  $a[1][1]$  и т.д.

# Моделирование многомерных массивов

## с использованием динамической памяти



Матрица  
одномерный массив  
из одномерных  
массивов (строк);  
 $a[i]$  – строка с  
номером  $i$ .

Следовательно,  $a[i]$  -  
указатель на  
первый байт  $i$ -й  
строки.

Т.е.  $a$  – одномерный  
массив указателей.  
Запишем  
выражения для  
элементов с учетом  
этого факта.

# Моделирование многомерных массивов

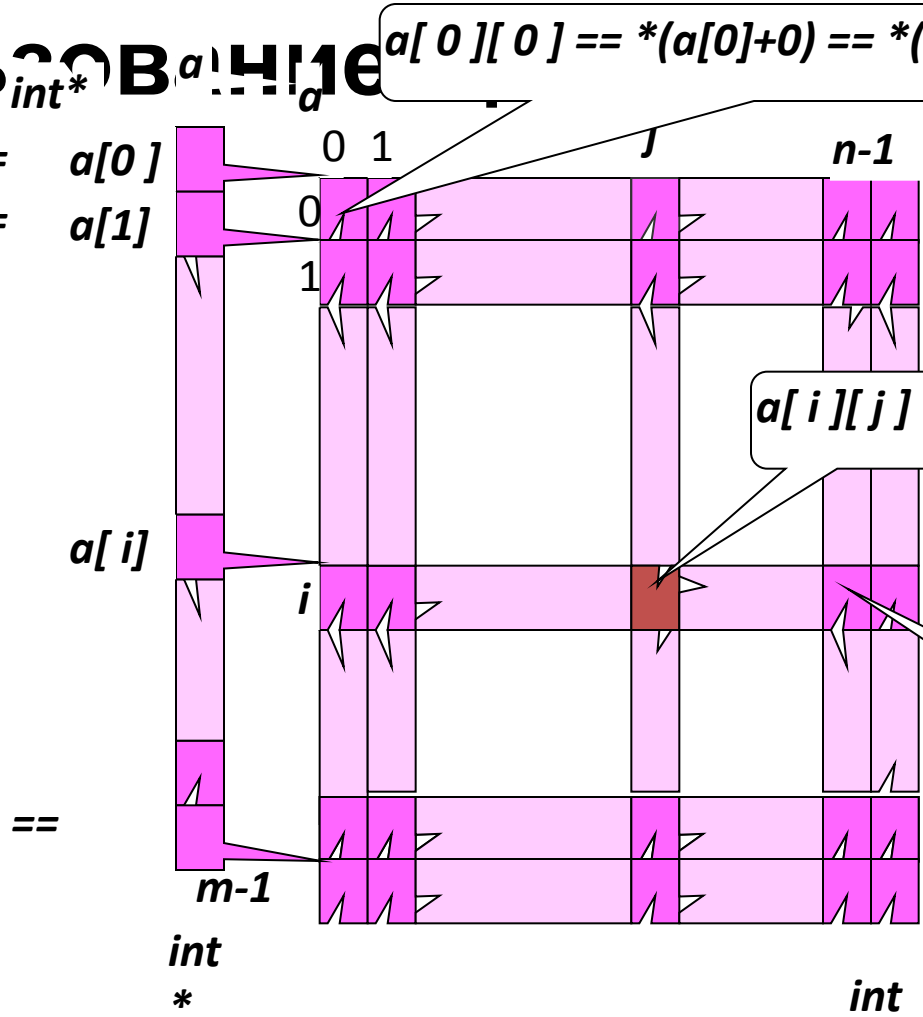
с использованием **интерпретации**

память

$*(a+0) ==$   
 $*(a+1) ==$

$*(a+i) ==$

$*(a+(m-1)) ==$   
 $a[m-1]$



$a[0][0] == *(a[0]+0) == (*(a+0)+0)$

$a[i][j]$

$a$  –  
одномерный  
массив  
указателей,  
следовательно  
 $a$  – указатель  
на начало  
этого массива.

$a[i]$

Запишем  
выражения  
для элементов  
с учетом этого  
факта.

# Динамическое распределение памяти.

Под динамической памятью понимают память, которая выделяется программе во время её работы. Динамическое распределение памяти включает выделение программе памяти по её запросу и последующее освобождение программой этой памяти. Для выделения памяти в языке C используются следующие функции:

```
void* malloc( size_t size );  
void* calloc( size_t nitems, size_t size);  
void* realloc( void* ptr, size_t size);
```

Прототипы этих функций находятся в заголовочном файле `stdlib.h`.

Параметры этих функций содержат тип `size_t`, который является переопределением типа `unsigned int`. Тип `void*` указателем на «пустую» память. Впоследствии этот указатель приводится к указателю на нужный тип.

## Важно!

- Динамическую память нужно освобождать после её использования. Иначе остаются неиспользованные блоки памяти, которые называются «мусором». Распространенная ошибка программирования “**утечка памяти**” (memory leak) заключается в том, что память выделяется динамически в цикле, но не освобождается. Постепенно свободная память заканчивается, хотя есть много неиспользуемых участков памяти, и программа заканчивается АВАРИЙНО.
- Динамическое выделение памяти используется только в том случае, если программе заранее неизвестно, какой размер памяти нужно отвести под данные



# Функция malloc

Функция `malloc` резервирует непрерывный блок ячеек памяти для хранения указанного объекта и возвращает указатель на первую ячейку этого блока. Обращение к функции имеет вид:

```
void *malloc(size);
```

Здесь `size` — целое беззнаковое значение, определяющее размер выделяемого участка памяти в байтах.

Если резервирование памяти прошло успешно, то функция возвращает указатель на выделенный участок памяти (переменную типа `void *`, которую можно привести к любому необходимому типу указателя), иначе - `NULL` при невозможности выделить память.

Пример:

```
double *p;  
if (( p=(double*) malloc (sizeof(double))) == NULL)  
{  
    printf ("Нехватка памяти \n");  
    return;  
}  
  
int i;  
scanf(" %d " , &i )  
*p = (double) i;
```

Здесь мы выделяем память под вещественное число типа `double` (`double*`) – приведение типа к требуемому типу объекта, для которого выделяем память `sizeof(double)` - функция, вычисляющая размер объекта данного типа в байтах

# Функция `calloc`

Функция - `calloc` также предназначена для выделения памяти. Следующая запись означает,

что будет выделено

`num` элементов по `size` байт :

```
void *calloc (num, size);
```

Эта функция возвращает указатель на выделенный участок или **NULL** при невозможности выделить память.

Особенностью функции `calloc` является обнуление всех выделенных элементов.

# Функция `realloc`

Функция `realloc` Функция `realloc` перераспределяет память, на которую указывает `p`, до размера в `size` байт. То есть программе распределяется новый блок памяти в `size` байт. При этом `size` байт информации из старого блока копируются в новый блок. Блок памяти может уменьшаться или увеличиваться в размере. После этого старый блок памяти освобождается.

Обращаются к ней так:

```
char *realloc (void *p, size);
```

Здесь `p` — указатель на ранее выделенную область памяти, размер которой нужно изменить на `size` байт.

Если в результате работы функции меняется адрес области памяти, то новый адрес вернется в качестве результата.

Если фактическое значение первого параметра `NULL`, то функция `realloc` работает также, как и функция `malloc`, то есть выделяет участок памяти размером `size` байт.

Если `size` равен 0, то ранее выделенная память будет освобождена, как если бы была вызвана функция `free`, и возвращается нулевой указатель.

Функция `realloc` в случае успешного завершения своей работы возвращает указатель на выделенный участок памяти или `NULL` в случае неудачи.

# Функция realloc

Таким образом, используя функцию `realloc()`, можно изменить размер ранее зарезервированного блока памяти.

```
void *ptr;      /* Указатель на предварительно */
                /* зарезервированный блок памяти */
int  size;     /* Новый размер блока в байтах */
void *realloc(ptr, size)
```

## ПРИМЕР :

```
int *a;
a=(int *)malloc(10*sizeof(int));
...
(int *)realloc(a,20*sizeof(int));
...
(int *)realloc(a,8*sizeof(int));
```

# функция `free`

Для освобождения выделенной памяти используется функция `free`. Обращаются к ней так:

```
void free (void *p size);
```

Здесь `p` — указатель на участок памяти, ранее выделенный функциями `malloc`, `calloc` или `realloc`.

# Операторы `new` и `delete`

Операторы `new` и `delete` в языке C++

аналогичны функциям `malloc` и `free` в языке C.

**New** выделяет память, а его единственный аргумент — это выражение, определяющее количество байт, которые будут зарезервированы. Возвращает оператор указатель на начало выделенного блока памяти.

```
// пример использования операции new :
```

```
int *ptrvalue = new int;
```

```
//где ptrvalue – указатель на выделенный участок памяти типа  
int
```

```
//new – операция выделения свободной памяти под создаваемый  
объект.
```

Оператор **delete** освобождает память, его аргумент — адрес первой ячейки блока, который необходимо освободить.

```
// пример использования операции delete:
```

```
delete ptrvalue;
```

```
// где ptrvalue – указатель на выделенный участок памяти типа  
int
```

```
// delete – операция высвобождения памяти
```

# Динамические массивы.

*Динамический массив* — массив переменной длины, память под который выделяется в процессе выполнения программы.

Для того чтобы динамически создать одномерный массив нужно:

- объявить в программе указатель на тип, соответствующий типу элементов этого массива
- выделить память под массив, используя одну из функций `malloc` или `calloc`.

# Динамические массивы.

При этом считается **грубой ошибкой** то, что описания

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

```
int *pa;
```

полностью равносильны одно другому.

Для того, чтобы указатель стал полностью эквивалентен массиву, необходимо заставить его ссылаться на область памяти соответствующей длины. Это можно сделать при помощи стандартных функций `malloc()`, захватывающих требуемое количество байт памяти и возвращающих адрес первого из них.

```
pa = (int*)malloc(10*sizeof(int));
```



# Пример создания одномерного

## целочисленного массива

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int *a, n;          // указатель на массив, размерность массива
    printf("Input a size of an array: ");
    scanf("%d", &n);
    a = (int*)malloc(n * sizeof(int));    // выделяем память под массив
    if (!a)
    {
        // если ошибка, то выходим из программы
        printf("Error: there is no memory.\n");
        return 0;
    }
    printf("Input elements of the array: ");    // вводим элементы массива
    for (int i = 0; i < n; ++i)
        scanf("%d", &a[i]);
    printf("Input elements of the array: ");    // можно ввести элементы массива и так
    for (i = 0; i < n; ++i)
        scanf("%d", a + i);
    printf("You input the array: ");    // что-то делаем с массивом
    for (i = 0; i < n; ++i)
        printf("%d ", a[i]);
    printf("\n");
    free(a);    // освобождаем выделенную память
    return 1;
}
```

# Многомерные динамические

## массивы

Память под многомерные динамические массивы выделяется по строкам, начиная с первого индекса. Это делается для того, чтобы обеспечить применение операции индексирования [ ] столько раз, какова размерность многомерного массива. В этом случае тип динамического массива объявляется как указатель, который содержит оператор обращения по адресу '\*' столько раз, какова размерность массива. Например, указатели на двумерный и трехмерный целочисленные массивы могут быть объявлены следующим образом:

```
int    **a;    // указатель на двумерный массив
int    ***b;   // указатель на трехмерный
массив
```

# Пример создания двумерного целочисленного массива

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

int main()
{
    int **a, n, m; // указатель на массив, размерности
                  массива
    int i, j; // индексы элементов матрицы
    printf ("Input two sizes of an array: ");
    scanf ("%d%d", &n, &m);

    // выделяем память для указателей на строки
    a = (int**)malloc(n * sizeof(int*));
    if (!a) // если ошибка, то выходим из программы
    {
        printf("Error: there is no memory.");
        return 0;
    }

    for (i = 0; i < n; ++i) // выделяем память для строк
    {
        a[i] = (int*)malloc(m * sizeof(int));
        if (!a[i]) // если ошибка, то освобождаем
        память
            //и выходим из программы
        {
            for (j = 0; j < i; ++j)
                free(a[j]);
            free(a);
            printf("Error: there is no memory.\n");
            return 0;
        }
    }
}
```

```
// вводим элементы массива
printf("Input elements of the matrix:\n");
for (i = 0; i < n; ++i)
    for (j = 0; j < m; ++j)
        scanf("%d", &a[i][j]);
// что-то делаем с матрицей
printf("You input the matrix:\n");
for (i = 0; i < n; ++i)
{
    for (j = 0; j < m; ++j)
        printf("%d ", a[i][j]);
    printf("\n");
}

// освобождаем захваченную
память
for (i = 0; i < n; ++i)
    free(a[i]);
free(a);
return 1;
}
```

# Пример. Скалярное произведение двух векторов. Функция malloc.

```
# include <stdio.h>
# include <malloc.h>
# include <conio.h>
// Функция ввода элементов
динамического
одномерного
массива
void input_array( float *x, char m_name,int n)
{
printf ("\n Введите элементы массива %c
[%i]\n ",m_name,n);
for (int i=0;i<n;i++)
{
printf ("%c [%i] = ",m_name,i);
scanf ("%f", (x+i));
}
}
// Функция вывода элементов
динамического
одномерного
массива
void output_array( float *x, char m_name,int n)
{
printf ("\n Массив %c [%i] \n",m_name,n);
for (int i=0;i<n;i++)
printf ("\t%4.2f", *(x+i));
puts (" ");
```

```
// ----- Основная программа -----
void main (void)
{
int i,m;
float *a, *b, *c;
clrscr();
printf ("\n Введите размер массивов для вычисления
их
скалярного произведения
");
scanf ("%i",&m);
a=(float *)malloc(m*sizeof(float));
b=(float *)malloc(m*sizeof(float));
c=(float *)malloc(m*sizeof(float));
input_array(a,'a',m);
input_array(b,'b',m);
for (i=0;i<m;i++)
*(c+i)=*(a+i) * *(b+i);
printf ("\n Исходные векторы :");
output_array(a,'a',m);
output_array(b,'b',m);
printf ("\n Результирующий вектор скалярного
произведения ");
output_array(c,'c',m);
free(a);
free(b);
free(c);
}
```

# Пример. Функция realloc.

Пример с изменением размера одномерного динамического массива в ходе исполнения программы.

Задача: вводится последовательность оценок (1-5) и высчитывается среднее арифметическое. Количество оценок заранее не известно, конец последовательности - 0.

```
# include <stdio.h>
# include <malloc.h>
# include <conio.h>
void main (void)
{
    int *a;
    clrscr();
    a=(int *)malloc(sizeof(int));
    int i=0;
        Do
        {
            (int *)realloc(a,(i+1)*sizeof(int));// Последовательное
            //увеличение размера динамического
            массива
            // на одну ячейку целого типа
            printf ("%i-я оценка :",i+1);
            scanf("%i", (a+i));
            i++;
        }
        while (*(a+i-1)!=0);
    puts (" ");
    int s=0;
        for (int j=0;j<i-1;j++)
        {
            printf ("%3i",*(a+j));
            s+=*(a+j);
        }
        float avg=s/i;
    printf ("\n Среднее арифметическое : %2.3f ",avg);
    free(a);
    getch();
}
```