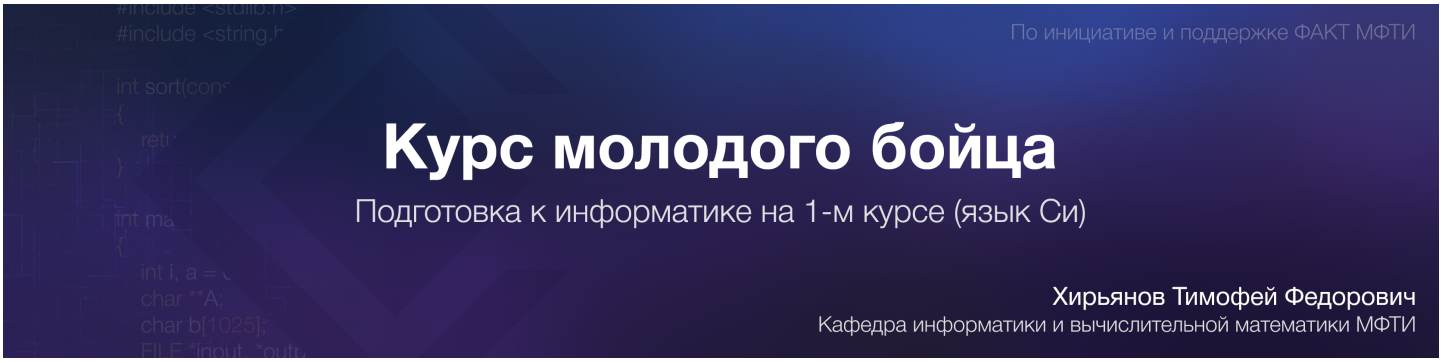
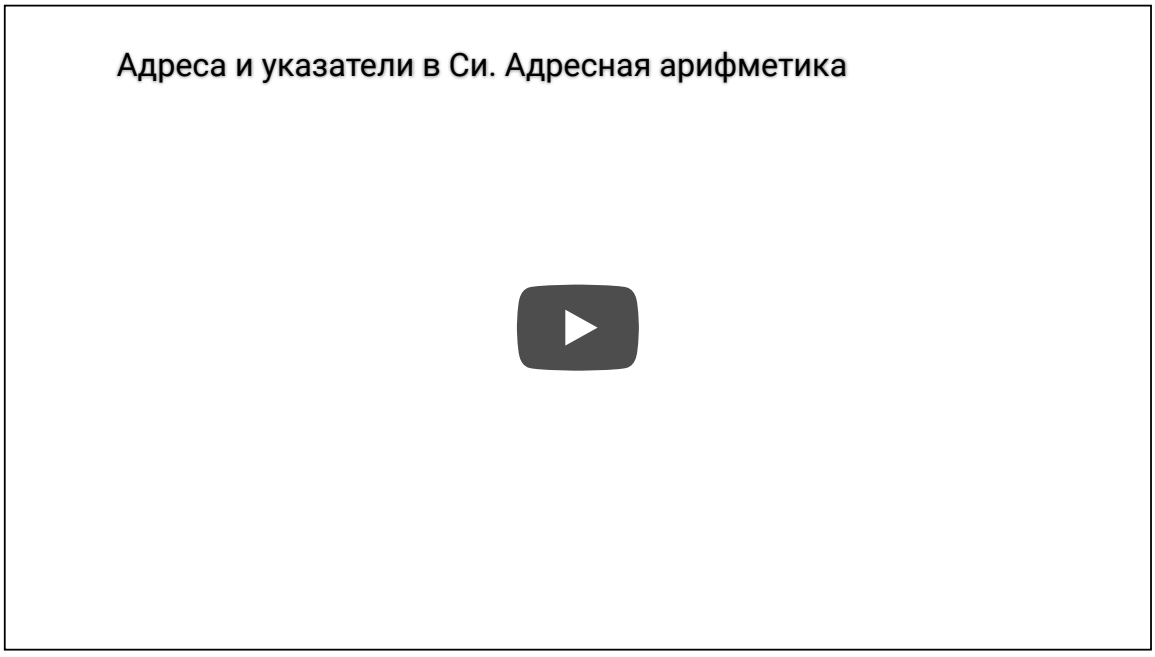


Главная

Адреса и указатели

Адреса и указатели в Си. Адресная арифметика



Объявление указателя. Разыменование * и взятие адреса &. Адресная арифметика. Имя массив как константный адрес.

addresses_and_pointers.c

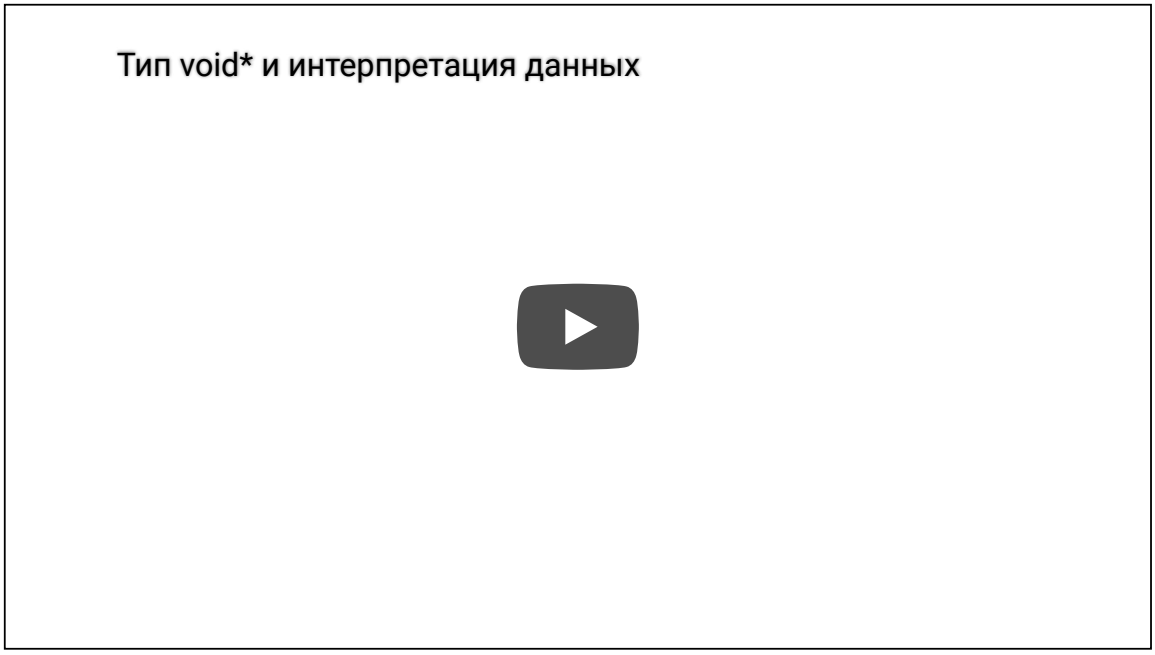
```c
#include <stdio.h>

int main(int argc, char* argv[])
{
    int i = 10;
    int *pi = &i;
    int **ppi = &pi;
    int ***pppi = &ppi;

    printf("%d\n", i);
    *pi = 20;
    printf("%d\n", i);
    **ppi = 30;
    printf("%d\n", i);
    ***pppi = 40;
    printf("%d\n", i);
```

```
    return 0;
}
```

address_arithmetics.c

```c
#include <stdio.h>

int main(int argc, char* argv[])
{
    int A[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    printf("%d\n", *A);

    int *p = A + 5;
    printf("%d\n", p[-1]);

    int *q = A + 7;
    if (p > q)
        printf("p > q\n");
    else
        printf("p <= q\n");

    printf("p - q = %d", p-q);

    return 0;
}
```

Тип void* и интерпретация данных



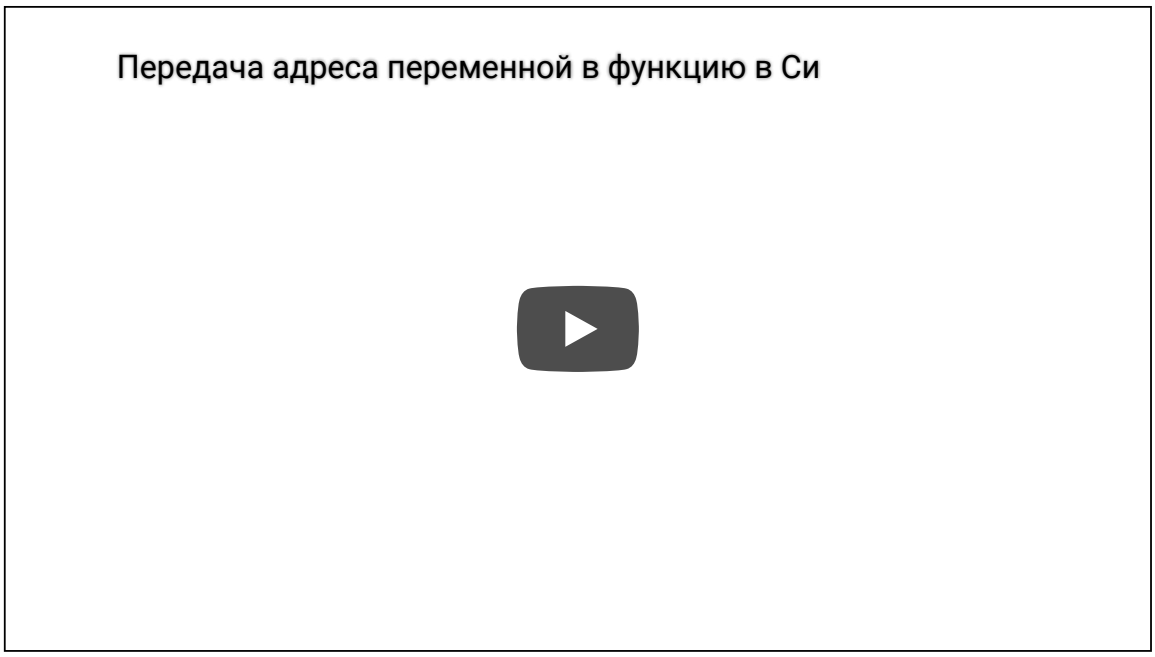
Размер ячейки для хранения адреса. Тип void*. Невозможность разыменования. Преобразование типа указателя. Пример реинтерпретации double как unsigned char. Интерпретация void* внутри функции по маркеру типа.

void_reinterpretation.c

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

void print_abstract(void *p, int type_marker);

int main(int argc, char* argv[])
{
    char c = 'W';
    int i = 450;
    double d = -1;

    void *p;
```

```
    p = &c;
    print_abstract(p, 1);
    p = &i;
    print_abstract(p, 2);
    p = &d;
    print_abstract(p, 3);

    return 0;
}

void print_abstract(void *p, int type_marker)
{
    if (type_marker == 1)
        printf("%c\n", *(char *)p);
    else if (type_marker == 2)
        printf("%d\n", *(int *)p);
    else if (type_marker == 3)
        printf("%lf\n", *(double *)p);
    else
    {
        printf("Unknown type marker. Exitting.");
        exit(1);
    }

}
```

Передача адреса переменной в функцию



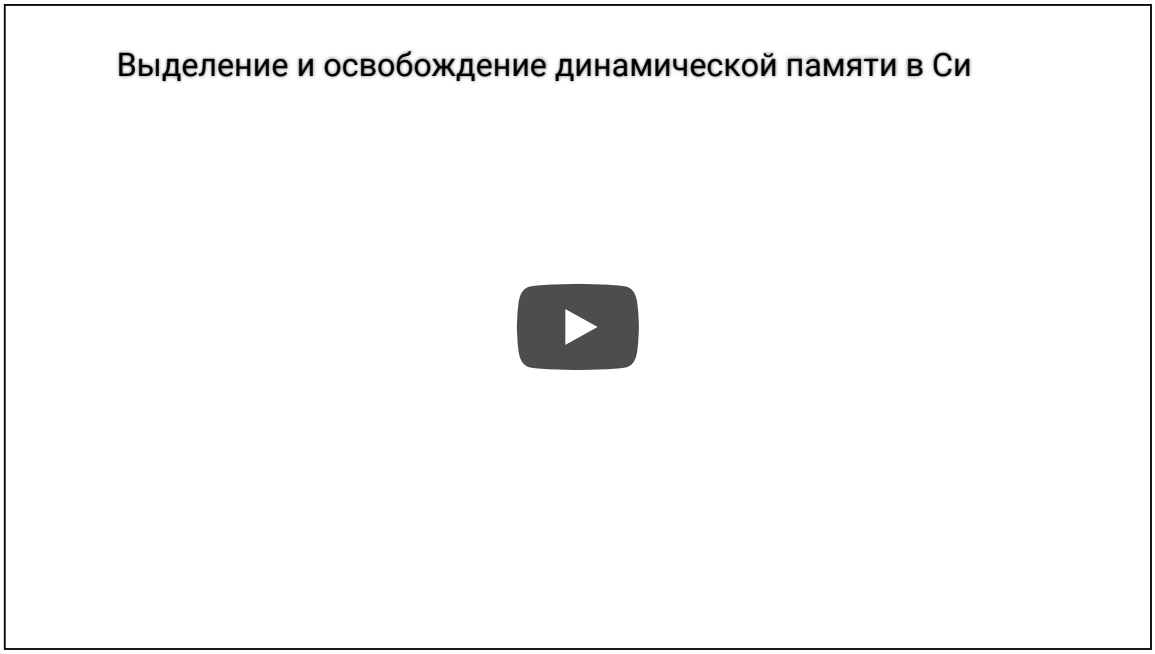
По умолчанию параметры передаются по значению, то есть копируются. Передаём функции параметр адресного типа. Изменение переменной-параметра из функции. Нельзя передавать адрес локальной переменной вне функции.

function_address_parameter.c

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

void foo(int* p)
{
    printf("Got: *p = %d\n", *p);
    *p += 10;
    printf("Did: *p = %d\n", *p);
}

int* bar()
{
    int y = 888;
```

```
    printf("y = %d\n", y);
    return &y;
}

int main(int argc, char* argv[])
{
    int x = 7;
    printf("x = %d\n", x);
    foo(&x);
    printf("x = %d\n", x);

    int *py = bar();
    printf("*py = %d\n", *py);

    return 0;
}
```

Выделение и освобождение динамической памяти



Распределение ресурсов операционной системой. Выделение динамической памяти: malloc(). Функция sizeof(тип), вычисляемая при компиляции. Необходимость освобождения памяти: free(). Независимость выделяемых отрезков памяти. Чем отличается функция calloc().

dynamic_memory.c

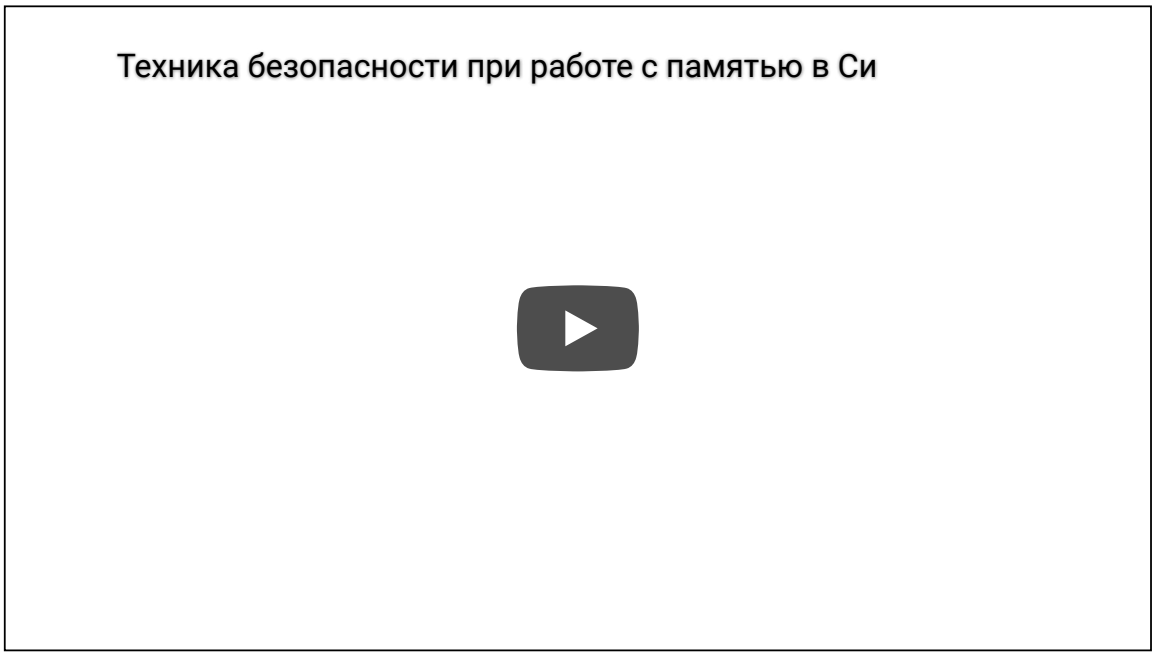
```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
    int N;
    printf("Enter size of array to create:");
    scanf("%d", &N);

    char *A = (char *)malloc(N);
    if (NULL == A)
    {
        printf("OS didn't gave memory. Exit...\n");
        exit(1);
    }
    for (int i = 0; i < N; ++i)
        A[i] = i;
    printf("Array A successfully created!\n");
    system("pause");
```

```
    return 0;
}
```

dynamic_int.c

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

int main(int argc, char* argv[])
{
    int N = 50000000;

    for (int k = 0; k < 1000; ++k)
    {
        int *A = (int *)malloc(N*sizeof(int));
        if (NULL == A) {
            printf("OS didn't gave memory. Exit...\n");
            exit(1);
        }
        printf("Allocate array - OK. iteration %d.\n", k);
        for (int i = 0; i < N; ++i)
            A[i] = i;
        //free(A); //TODO: uncomment this line
    }
    printf("Program is on finish!\n");
    system("pause");

    return 0;
}
```

Техника безопасности при работе с памятью



Ошибки работы с памятью в Си: Segmentation fault, Memory leak. Инициализация указателей: NULL. Проверка корректности адреса. Ответственность за освобождение памяти.

segmentation_fault.c

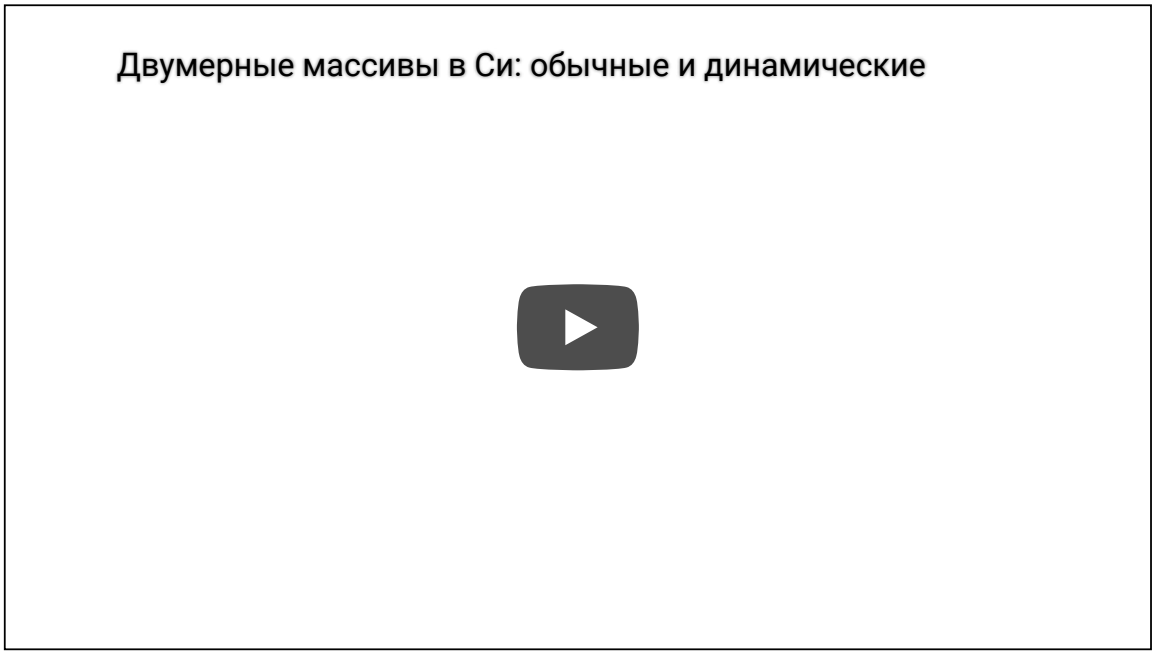
```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

void foo(int *pointer)
{
    assert(pointer);
    *pointer = 0; //potential Segmentation fault
```

```
}

int main()
{
    int *p = NULL;  // Uninitialized pointer!
    //*p = 10; // Using it => Segmentation fault!

    //foo(p);  // Another use of uninitialized pointer => Segmentation fault!

    int x = 100;
    scanf("%d", x); // Very popular Segmentation fault.

    return 0;
}
```

memory_leak.c

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

// Danger function: it's not responsible for
// the memory it allocates for the duplicate!
int* duplicate_array(int *A, size_t N)
{
    int * B = (int *) malloc(sizeof(int)*N);
    for(size_t i = 0; i < N; i++)
        B[i] = A[i];
    printf(" duplicate_array() allocated memory for the duplicate.\n");
    return B;
}

int main()
{
    printf("Calling irresponsible function duplicate_array():\n");
    int A[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int *B = duplicate_array(A, 10);
    for (int i = 0; i < 10; ++i)
        printf("%d\t", B[i]);

    printf("Since caller function is not taking responsibility by itself,\n");
    printf(" memory for the array above will never be released...\n\n");

    printf("The same situation for the standard function strdup():\n");
    char *hello = "Hello, World!";
    char *message = strdup(hello);
    printf("Strdup allocated memory for this message: \"%s\"\n", message);
    printf("It'll never be released...\n\n");

    int *p;
    for (int i = 0; i < 10; i++)
    {
        p = (int *)malloc(sizeof(int));
        printf("Allocating memory many times in cycle.\n");
        *p = i;
    }
    free(p);
    printf("But releasing it just once...\n");

    return 0;
}
```

Двумерные массивы: обычные и динамические



Обычные двумерные массивы в C. Передача двумерного массива в функцию. Динамические двумерные массивы в C. Выделение и освобождение памяти для динамического двумерного массива. Передача динамического двумерного массива в функцию и возврат из функции.

static_2d_array.c

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

#define MATRIX_HEIGHT 4
#define MATRIX_WIDTH 5

void static_array_print(int A[][MATRIX_WIDTH], size_t N)
{
    for(int i = 0; i < N; i++) {
        for(int j = 0; j < MATRIX_WIDTH; j++) {
            printf("%*d", 5, A[i][j]);
        }
        printf("\n");
    }
}

void static_array_test(size_t N)
{
    int A[N][MATRIX_WIDTH];
    int x = 1;
    for(int i = 0; i < N; i++) {
        for(int j = 0; j < MATRIX_WIDTH; j++) {
            A[i][j] = x;
            x += 1;
        }
    }
    static_array_print(A, N);

    /*memory investigation*/
    printf("\n Direct memory access:\n");
    for(int *p = (int *)A; p < (int *)A + 20; p++)
        printf("%3d", *p);
    printf("\n\n");
}

int main()
{
    static_array_test(MATRIX_HEIGHT);
    return 0;
}
```

dynamic_2d_array.c

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

void dynamic_array_print(int **A, size_t N, size_t M)
{
    for(int i = 0; i < N; i++) {
        for(int j = 0; j < M; j++) {
            printf("%*d", 5, A[i][j]);
        }
        printf("\n");
    }
}

/*
    return pointer on 2d dynamic array
    !allocates memory -> to be freed later
*/
int ** dynamic_array_alloc(size_t N, size_t M)
{
    int **A = (int **)malloc(N*sizeof(int *));
    for(int i = 0; i < N; i++) {
        A[i] = (int *)malloc(M*sizeof(int));
    }
    return A;
}

void dynamic_array_free(int **A, size_t N)
{
    for(int i = 0; i < N; i++) {
        free(A[i]);
    }
    free(A);
}

void dynamic_array_test(size_t N, size_t M)
{
    int **A = dynamic_array_alloc(N, M);
    int x = 1;
    for(int i = 0; i < N; i++) {
        for(int j = 0; j < M; j++) {
            A[i][j] = x;
            x += 1;
        }
    }
    dynamic_array_print(A, N, M);
    /*memory investigation*/
    printf("\n Pointers to lines: ");
    for(int **p = A; p < A + 3; p++)
        printf("%10d", (long int)*p);
    printf("\n Direct memory access:\n");
    for(int *p = (int*)*A; p < (int*)*A + 25; p++)
        printf("%d\t", *p);
    dynamic_array_free(A, N);
}

int main()
{
    int matrix_height = 4;
    int matrix_width = 5;

    dynamic_array_test(matrix_height, matrix_width);
    return 0;
}
```

Самостоятельная работа

Уважаемые студенты! К 5-му уроку контеста не предусмотрено.

Вместо этого я приглашаю вас к самостоятельному изучению материалов на сайте http://acm.mipt.ru

А также к участию в учебных соревнованиях на сайте http://codeforces.com/