



СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

ЛЕКЦИЯ № 3

ПРЕПОДАВАТЕЛЬ: ХУСТОЧКА А.В.



УКАЗАТЕЛЬ

Объявление указателя:
<тип>* <переменная>

Пример:
`int* pointer = NULL;`

Две основные операции для работы с указателем:

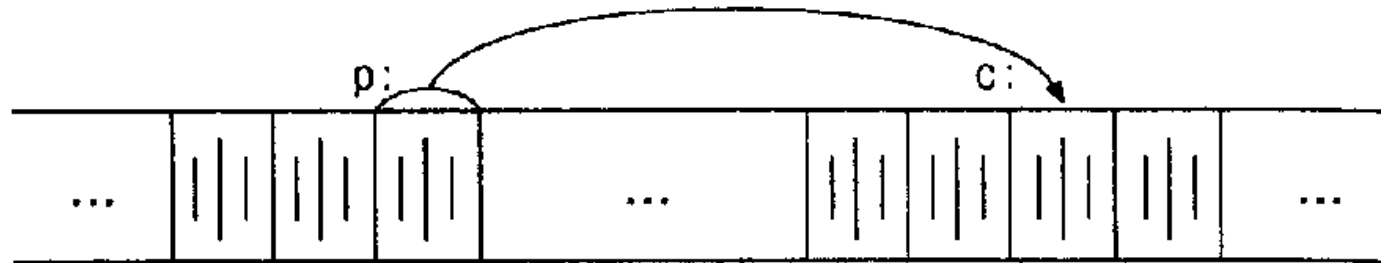
- & – оператор взятия адреса
- * – оператор разыменования

УНАРНЫЙ ОПЕРАТОР &

Оператор & – оператор взятия адреса

Пример:

```
int c = 5;  
int* p = &c;
```



Оператор & применяется только к объектам, расположенным в памяти: к переменным и элементам массивов. Его операндом не может быть ни выражение, ни константа, ни регистровая переменная.

УНАРНЫЙ ОПЕРАТОР *

Оператор * – оператор разыменования (косвенного доступа)

Пример:

```
int main()
{
    int x = 1, y = 2;
    int z[5] = { 1, 2, 3, 4, 5};
    int *p;      /* p - указатель на int */

    p = &x;      /* теперь p указывает на x */
    y = *p;      /* y теперь равен 1 */
    *p = 0;      /* x теперь равен 0 */
    p = &z[2];   /* p теперь указывает на z[2] (равен 3) */

    return 0;
}
```

УКАЗАТЕЛИ И АРГУМЕНТЫ ФУНКЦИИ

Определение функции:

```
void swap(int x, int y)
{
    int temp;
    temp = x;
    x = y;
    y = temp;
}
```

Вызов функции:

```
int a = 5;
int b = 6;
swap(a, b);
```

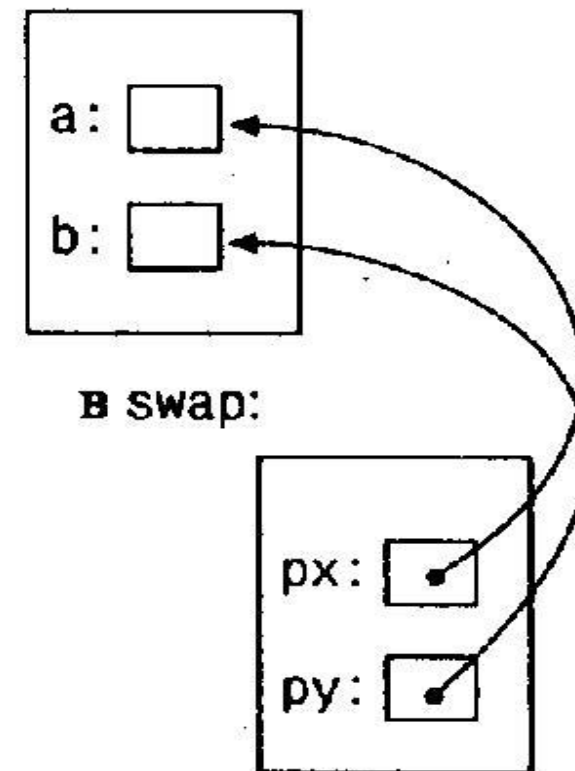
Определение функции:

```
void swap(int *x, int *y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
```

Вызов функции:

```
int a = 5;
int b = 6;
swap(&a, &b);
```

Графическое представление:



УКАЗАТЕЛИ И МАССИВЫ

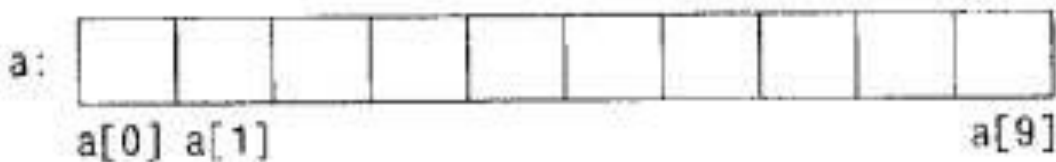
Объявление массива:

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

Объявление указателя на начало массива:

1. `int* p = &a[0];`
2. `int* p = a;`

Представление массива в памяти:



Если мы объявим переменные:

```
int a1 = a[i];  
int a2 = *(a+i);
```

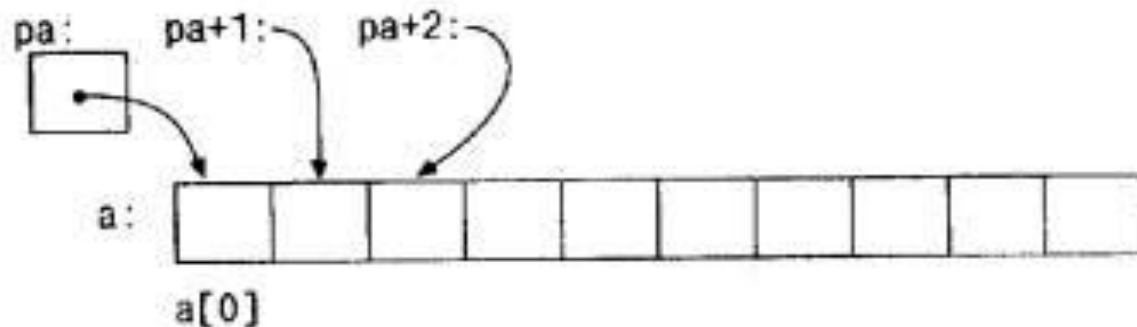
То будут ли значения `a1` и `a2` равны?

Объявление указателя на *i*-ый элемент:

```
int* p = &a[i];
```

Содержимое *i*-го элемента:

```
int ai = *p;
```



МАССИВ, КАК ПАРАМЕТР ФУНКЦИИ

Функция получения длины строки

```
int get_length(char *str)
{
    int i = 0;
    for (i = 0; *str != '\0'; str++) {
        i++;
    }
    return i;
}
```

Примеры вызова функции:

```
char* str_pointer = (char*)"kotik";
char str_array[] = "kotik";
int length = get_length((char*)"kotik");
int length_pointer = get_length(str_pointer);
int length_array = get_length(str_array);
int length_part_array = get_length(&str_array[2]);
```

Чему равна переменная?

```
char str_array[] = "kotik";
char symbol = str_array[-2];
```

СТЕК И КУЧА

	Стек	Куча
Определение	Область оперативной памяти	Область оперативной памяти
Принцип работы	Порядок LIFO – последний добавленный в стек кусок памяти будет первым в очереди на вывод из стека	Данные хранятся хаотично
Время жизни	Живёт в области видимости переменных. (Область видимость обозначается фигурными скобками {...})	Живёт до тех пор, пока её не освободит программист.
Размер	Фиксированная величина: размер задаётся при создании потока (В ОС Windows по умолчанию 1Мб, на некоторых Unix-системах – до 8Мб)	Выделяется операционной системой и ограничен только физически

Примечание: из-за своего принципа работы, стек работает быстрее, чем куча

ДИНАМИЧЕСКАЯ ПАМЯТЬ

Динамическая память выделяется в куче.

Функции, для работы с динамической памятью:

Функция	Описание
<code>malloc(size_t size)</code>	Выделяет заданный размер байт
<code>calloc(size_t count, size_t size)</code>	Выделяет заданный размер байт и инициализирует их нулём
<code>realloc(void* block, size_t size)</code>	Перевыделяет заданный размер байт, сохраняя значение выделенного ранее блока памяти. Если <code>block</code> равен <code>NULL</code> , то работает как <code>malloc()</code>
<code>free(void* block)</code>	Освобождает выделенную память

Пример работы malloc()

Память 1

Адрес: 0x000000F5A056D540

0x000000F5A056D540	cd cd cd cd	cd cd cd cd	cd cd cd cd	cd cd cd cd	fd fd fd fd
0x000000F5A056D55E	00 80 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0x000000F5A056D57C	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0x000000F5A056D59A	00 00 00 00	00 00 00 00	00 00 00 00	ba b1 51 88	00 08 00 80

ConsoleApplication2.cpp

ConsoleApplication2 (Глобальная область)

```
1 //define _CRT_SECURE_NO_WARNINGS
2 #include <stdio.h>
3 #include <malloc.h>
4
5 int main()
6 {
7     int* p1 = (int*)malloc(sizeof(int) * 5);
8     int* p2 = p1;
9     int* p3 = (int*)realloc(NULL, sizeof(int) * 5);
10    p3 = (int*)realloc(p3, sizeof(int) * 10);
11
12    free(p1);
13    free(p2);
14    free(p3);
15    return 0;
16 }
17
```

Debugger window showing variable p1: 0x000000f5a056d540 {0xcdcdcdcd}

Пример работы calloc()

Память 1

Адрес: 0x000000F5A056D450

0x000000F5A056D450	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	fd fd fd
0x000000F5A056D46E	00 80 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00
0x000000F5A056D48C	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00
0x000000F5A056D4AA	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	ab b1 60 88	00 05 00 80

ConsoleApplication2.cpp

ConsoleApplication2

(Глобальная область)

```
1 //define _CRT_SECURE_NO_WARNINGS
2 #include <stdio.h>
3 #include <malloc.h>
4
5 int main()
6 {
7     int* p1 = (int*)malloc(sizeof(int) * 5);
8     int* p2 = (int*)calloc(5, sizeof(int));
9     int* p3 = p2; // 0x000000f5a056d450 {0x00000000} ≤ 1 мс прошло
10    p3 = (int*)realloc(p3, sizeof(int) * 10);
11
12    free(p1);
13    free(p2);
14    free(p3);
15    return 0;
16 }
17
```

Пример работы realloc()

Память 1

Адрес: 0x000000F5A056DA40

0x000000F5A056DA40	cd cd cd cd cd cd cd cd cd cd cd cd cd cd cd cd cd cd fd f
0x000000F5A056DA5E	00 80 0
0x000000F5A056DA7C	00 0
0x000000F5A056DA9A	00 0

ConsoleApplication2.cpp

ConsoleApplication2 (Глобальная область)

```
1 //define _CRT_SECURE_NO_WARNINGS
2 #include <stdio.h>
3 #include <malloc.h>
4
5 int main()
6 {
7     int* p1 = (int*)malloc(sizeof(int) * 5);
8     int* p2 = (int*)calloc(5, sizeof(int));
9     int* p3 = (int*)realloc(NULL, sizeof(int) * 5);
10    p3 = (i p3 0x000000f5a056da40 {0xcdcdcdcd} -> ошло
11
12    free(p1);
13    free(p2);
14    free(p3);
15    return 0;
16 }
17
```

Пример работы realloc()

Память 1

Адрес: 0x000000F5A05635F0

0x000000F5A05635F0	cd	cd	cd	cd	cd	cd	cd	cd	cd	cd	cd	cd	cd	cd	cd	cd	cd	cd	cd	cd	cd	cd	cd	cd	cd	cd	cd	cd		
0x000000F5A056360E	cd	cd	cd	cd	cd	cd	cd	cd	cd	fd	fd	fd	fd	00	00	00	00	00	00	00	00	00	00	00	00	d0	21	9d	78	
0x000000F5A056362C	4f	02	00	0a	43	00	3a	00	5c	00	57	00	69	00	6e	00	64	00	6f	00	77	00	73	00	5c	00	73	00	79	00
0x000000F5A056364A	73	00	74	00	65	00	6d	00	33	00	32	00	5c	00	4b	00	45	00	52	00	4e	00	45	00	4c	00	42	00	41	00

ConsoleApplication2.cpp

ConsoleApplication2 (Глобальная область) main()

```
1  //define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3  #include <malloc.h>
4
5  int main()
6  {
7      int* p1 = (int*)malloc(sizeof(int) * 5);
8      int* p2 = (int*)calloc(5, sizeof(int));
9      int* p3 = (int*)realloc(NULL, sizeof(int) * 5);
10     p3 = (int*)realloc(p3, sizeof(int) * 10);
11
12     free(p1);
13     free(p2);
14     free(p3);
15     return 0;
16 }
```

Debugger window: p3 0x000000f5a05635f0 {0xcdcdcdcd}

Пример работы realloc()

Адрес: 0x000000E1F351D5E0

0x000000E1F351D5E0	01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 fd fd fd fd da 7e 46 a9 00 09
0x000000E1F351D5FE	00 80 00
0x000000E1F351D61C	00 00
0x000000E1F351D63A	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 e1 7e 41 a9 00 0a 00 80 00 00 00 00 00

ConsoleApplication2.cpp

ConsoleApplication2 (Глобальная область) main()

```
1 // #define _CRT_SECURE_NO_WARNINGS
2 #include <stdio.h>
3 #include <string.h>
4 #include <malloc.h>
5
6 int main()
7 {
8     int* p1 = (int*)malloc(sizeof(int) * 5);
9     int* p2 = (int*)calloc(5, sizeof(int));
10    int* p3 = (int*)realloc(NULL, sizeof(int) * 5);
11    memset(p3, 1, sizeof(int) * 5);
12    p3 = (int*)realloc(p3, sizeof(int) * 10);
13
14    free(p1);
15    free(p2);
16    free(p3);
17    return 0;
18 }
19
```

Debugger window: p3 0x000000e1f351d5e0 {0x01010101} ...

Пример работы realloc()

Адрес: 0x000000E1F35135F0

0x000000E1F35135F0	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	cd	cd	cd	cd	cd	cd	cd	cd	cd		
0x000000E1F351360E	cd	cd	cd	cd	cd	cd	cd	cd	cd	cd	fd	fd	fd	fd	00	00	00	00	00	00	00	00	00	00	00	66	3d	5c	06	
0x000000E1F351362C	1a	fd	00	0a	43	00	3a	00	5c	00	57	00	69	00	6e	00	64	00	6f	00	77	00	73	00	5c	00	73	00	79	00
0x000000E1F351364A	73	00	74	00	65	00	6d	00	33	00	32	00	5c	00	4b	00	45	00	52	00	4e	00	45	00	4c	00	42	00	41	00

ConsoleApplication2.cpp

ConsoleApplication2

(Глобальная область)

main()

```
1  //define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3  #include <string.h>
4  #include <malloc.h>
5
6  int main()
7  {
8      int* p1 = (int*)malloc(sizeof(int) * 5);
9      int* p2 = (int*)calloc(5, sizeof(int));
10     int* p3 = (int*)realloc(NULL, sizeof(int) * 5);
11     memset(p3, 1, sizeof(int) * 5);
12     p3 = (int*)realloc(p3, sizeof(int) * 10);
13     p3 0x000000e1f35135f0 {0x01010101}
14     free(p1);
15     free(p2);
16     free(p3);
17     return 0;
18 }
```

≤ 1 мс прошло

АДРЕСНАЯ АРИФМЕТИКА

Арифметика указателей нужна:

- Для поддержки адекватной работы операции разыменования
- Для поддержки арифметических операций

Указатели поддерживают следующие операции:

- Сложение
- Вычитание
- Сравнение

АДРЕСНАЯ АРИФМЕТИКА

Почему так делать нельзя?

```
double* p1;  
float* p2 = (float*)p1;
```

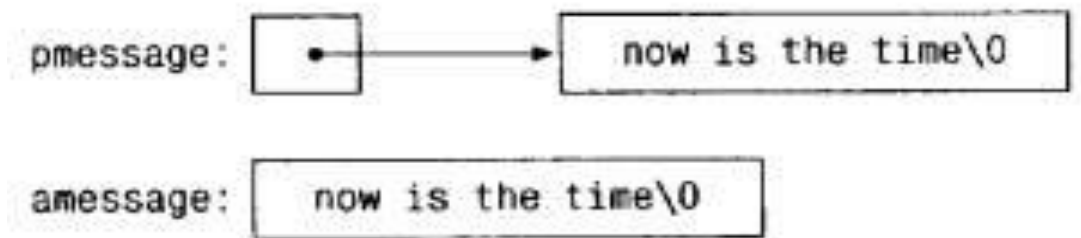
Почему так делать можно?

```
typedef struct {  
    int age;  
    char* name;  
} PONY;  
  
void print_pony_information(void* info)  
{  
    PONY* pony = (PONY*)info;  
    printf("Pony name: %s\nPony age: %d\n",  
        pony->name, pony->age);  
}
```

```
int main()  
{  
    const char* name = "Ignat";  
    int size_name = strlen(name) + 1;  
    PONY pony;  
    pony.age = 5;  
    pony.name = (char*)name;  
    print_pony_information(&pony);  
  
    PONY* pony_dynamic = (PONY*)calloc(1, sizeof(PONY));  
    pony_dynamic->age = 5;  
    pony_dynamic->name = (char*)calloc(size_name, sizeof(char));  
    strcpy_s(pony_dynamic->name, size_name, name);  
    print_pony_information(&pony);  
  
    free(pony_dynamic->name);  
    free(pony_dynamic);  
    return 0;  
}
```

СИМВОЛЬНЫЕ УКАЗАТЕЛИ

```
char amessage[] = "now is the time";  
char *pmessage = "now is the time";
```



Что делает функция?

```
void function(char *a, char *b)  
{  
    while (*a++ = *b++);  
}  
  
int main()  
{  
    char* a = (char*)"meow";  
    char b[10] = { 0 };  
  
    function(b, a);  
    function(a, b);  
  
    return 0;  
}
```

МНОГОМЕРНЫЕ МАССИВЫ

Рассмотрим задачу перевода даты «день/месяц» в «день года».

Например, 1 марта – это 60 день не високосного года или 61 – високосного.

```
#define COUNT_OF_MOUNTH 12  
#define COUNT_OF_YEAR 2
```

```
typedef enum { JANUARY, FEBRUARY, MARCH, APRIL, MAY, JUNE,  
              JULY, AUGUST, SEPTEMBER, OCTOBER, NOVEMBER, DECEMBER };
```

```
static char days[COUNT_OF_YEAR][COUNT_OF_MOUNTH] = {  
    { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},  
    { 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}  
};
```

```
/* day_of_year: определяет день года по месяцу и дню */
```

```
int day_of_year(int year, int month, int day)
{
    int is_leap_year = 0;
    is_leap_year = ((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0);
    for (int i = 0; i < month; i++) {
        day += days[is_leap_year][i];
    }
    return day;
}
```

```
/* month_day: определяет месяц и день по дню года */
```

```
void month_day(int year, int yearday, int *month, int *day)
{
    int i = 0;
    int is_leap_year = 0;
    is_leap_year = ((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0);
    for (i = 0; yearday > days[is_leap_year][i]; i++) {
        yearday -= days[is_leap_year][i];
    }
    *month = i;
    *day = yearday;
}
```

```
int main()
{
    int day = 0;
    int mounth = 0;
    int x = 0;
    int y = 0;

    x = day_of_year(2022, MARCH, 1);
    y = day_of_year(2024, MARCH, 1);
    printf("1st March of 2022 = %d\n", x);
    printf("1st March of 2024 = %d\n", y);

    month_day(2022, x, &mounth, &day);
    printf("%d day of 2022 = %d/%d/2022\n", x, day, mounth + 1);
    printf("%d day of 2024 = %d/%d/2024\n", y, day, mounth + 1);
    return 0;
}
```

Результат работы программы:

```
1st March of 2022 = 60
1st March of 2024 = 61
60 day of 2022 = 1/3/2022
61 day of 2024 = 1/3/2024
```

МАССИВ УКАЗАТЕЛЕЙ

Добавим к предыдущей задачи функцию, выводящую по номеру месяца его название:

```
const char *get_month_name(int n)
{
    static const char *name[] = {
        "JANUARY", "FEBRUARY", "MARCH", "APRIL", "MAY", "JUNE",
        "JULY", "AUGUST", "SEPTEMBER", "OCTOBER", "NOVEMBER", "DECEMBER", "INVALID VALUE"
    };
    return (n < 0 || n > 11) ? name[COUNT_OF_MOUNTH] : name[n];
}
```

И изменим вывод в основной функции программы:

```
printf("%d day of 2022 = %d/%s/2022\n",
      x, day, get_month_name(mounth));
printf("%d day of 2024 = %d/%s/2024\n",
      y, day, get_month_name(mounth));
```

Результат работы программы:

```
60 day of 2022 = 1/MARCH/2022
61 day of 2024 = 1/MARCH/2024
```

УКАЗАТЕЛИ И МНОГОМЕРНЫЕ МАССИВЫ

Объявление двумерного массива:

```
int a[10][20];
```

Объявление указателя на массив:

```
int *b[10];
```

Записи `a[3][4]` и `b[3][4]` – синтаксически верны, однако в памяти они выглядят по-разному.

Рассмотрим пример:

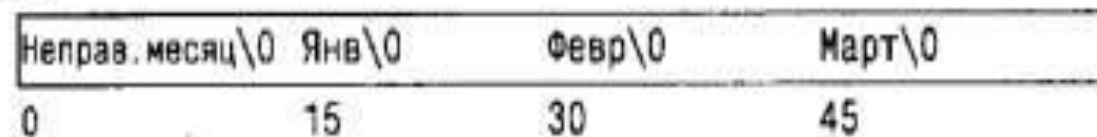
```
const char *name[] = { "Неправильный месяц", "Янв", "Февр", "Март" };
```

```
char aname[][15] = { "Неправ. месяц", "Янв", "Февр", "Март" };
```

name:



aname:



ДВОЙНЫЕ УКАЗАТЕЛИ

Объявление указателя на массив :

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

Объявление двойного указателя на массив:

```
int **b;
```

Записи `a[3][4]` и `b[3][4]` – синтаксически верны, однако в памяти они выглядят по-разному.

Рассмотрим пример: объявим двойной массив `a` на стеке, указатель на массив `b` и двойной указатель `c`

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

```
int *b[3] = { a[0], a[1], a[2] };
int **c = b;
```

```
void print_array(int** arr)
{
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d\t", arr[i][j]);
        }
        printf("\n");
    }
}
```



```

int a[3][3];
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        a[i][j] = i + j;
    }
}

```

```

printf("Array a:\n");
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        printf("%d\t", a[i][j]);
    }
    printf("\n");
}

```

```

int *b[3] = { a[0], a[1], a[2] };
printf("Array b:\n");
print_array(b);

```

```

int **c = b;
printf("Array c:\n");
print_array(c);

```

Память 1

Адрес: 0x00000023AC3BF8B8

0x00000023AC3BF8B8	00 00 00 00	01 00 00 00	02 00 00 00
0x00000023AC3BF8C4	01 00 00 00	02 00 00 00	03 00 00 00
0x00000023AC3BF8D0	02 00 00 00	03 00 00 00	04 00 00 00
0x00000023AC3BF8DC	cc cc cc cc	cc cc cc cc	cc cc cc cc
0x00000023AC3BF8E8	cc cc cc cc	cc cc cc cc	cc cc cc cc
0x00000023AC3BF8F4	cc cc cc cc	cc cc cc cc	cc cc cc cc

a	0x00000023ac3bf8b8 {0x00000023ac3bf8b8 {0x00000000, 0x00000001, 0x00000002}, 0x00000000, 0x00000001, 0x00000002}
▶ [0x00000000]	0x00000023ac3bf8b8 {0x00000000, 0x00000001, 0x00000002}
▶ [0x00000001]	0x00000023ac3bf8c4 {0x00000001, 0x00000002, 0x00000003}
▶ [0x00000002]	0x00000023ac3bf8d0 {0x00000002, 0x00000003, 0x00000004}

b	0x00000023ac3bf978 {0x00000023ac3bf8b8 {0x00000000}, 0x00000023ac3bf8c4 {0x00000001}, 0x00000023ac3bf8d0 {0x00000002}}
▶ [0x00000000]	0x00000023ac3bf8b8 {0x00000000}
▶ [0x00000001]	0x00000023ac3bf8c4 {0x00000001}
▶ [0x00000002]	0x00000023ac3bf8d0 {0x00000002}

c	0x00000023ac3bf978 {0x00000023ac3bf8b8 {0x00000000}}
▶ [0x00000000]	0x00000023ac3bf8b8 {0x00000000}

```

Array a:
0      1      2
1      2      3
2      3      4

Array b:
0      1      2
1      2      3
2      3      4

Array c:
0      1      2
1      2      3
2      3      4

```

```

int **c;
c = (int**)calloc(3, sizeof(int));
for (int i = 0; i < 3; i++) {
    c[i] = (int*)calloc(3, sizeof(int));
    for (int j = 0; j < 3; j++) {
        c[i][j] = a[i][j];
    }
}

```

Память 1

Адрес: 0x000000340465D540

0x000000340465D540	00 00 00 00 00 00 00 00	00 00 00 00 fd fd fd fd	00 00 00 00 00 00 00 00	0b cb 03 4a
0x000000340465D55C	00 07 00 80 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00	...Ъ.....
0x000000340465D578	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00
0x000000340465D594	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	04 cb 08 4a 00 08 00 80

ConsoleApplication2.cpp X

ConsoleApplication2 (Глобальная область) main()

78 int a[3][3];

79 for (int i = 0; i < 3; i++) {

80 for (int j = 0; j < 3; j++) {

81 a[i][j] = i + j;

82 }

83 }

84

85 printf("Array a:\n");

86 for (int i = 0; i < 3; i++) {

87 for (int j = 0; j < 3; j++) {

88 printf("%d\t", a[i][j]);

89 }

90 printf("\n");

91 }

92

93 int **c;

94 c = (int**)calloc(3, sizeof(int));

95 for (int i = 0; i < 3; i++) {

96 c[i] = (int*)calloc(3, sizeof(int));

97 for (int j = 0; j < 3; j++) {

98 c[i][j] = a[i][j];

Память 1

Адрес: 0x000000340465D540

0x000000340465D540	80 d6 65 04 34 00 00 00 00 00 00 00 00 fd fd fd fd 00 00 00 00 00 00 0b cb 03 4a
0x000000340465D55C	00 07 00 80 00
0x000000340465D578	00 00
0x000000340465D594	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 04 cb 08 4a 00 08 00 80

ConsoleApplication2.cpp

ConsoleApplication2 (Глобальная область) main()

```
92
93 int **c;
94 c = (int**)calloc(3, sizeof(int));
95 for (int i = 0; i < 3; i++) {
96     c[i] = (int*)calloc(3, sizeof(int));
97     for (int j = 0; j < 3; j++) { ≤ 2 мс прошло
98         c[i][j] = a[i][j];
99     }
100 }
101 printf("Array c:\n");
102 print_array(c);
103
104 return 0;
105 }
106
```

c 0x000000340465d540 (0x000000340465d680 {0x00000000})

Память 1

Адрес: 0x000000340465D680

0x000000340465D680	00 00 00 00 00 00 00 00 00 00 00 00 fd fd fd fd 00 00 00 00 00 00 00 00 37 cb 1f 4a
0x000000340465D69C	00 0b 00 80 00
0x000000340465D6B8	00 00
0x000000340465D6D4	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 30 cb 64 4a 00 0c 00 8b

ConsoleApplication2.cpp

ConsoleApplication2 (Глобальная область) main()

```
92
93 int **c;
94 c = (int**)calloc(3, sizeof(int));
95 for (int i = 0; i < 3; i++) {
96     c[i] = (int*)calloc(3, sizeof(int));
97     for (int j = 0; j < 3; j++) { ≤ 2 мс прошло
98         c[i][j] = a[i][j];
99     }
100 }
101 printf("Array c:\n");
102 print_array(c);
103
104 return 0;
105 }
106
```





c 0x000000340465d540 (0x000000340465d680 {0x00000000})

Память 1

Адрес: 0x000000340465D540

0x000000340465D540	80 d6 65 04 34 00 00 00	10 d8 65 04 34 00 00 00	90 da 65 04 34 00 00 00	0b
0x000000340465D567	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00
0x000000340465D58E	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00

Контрольные значения 1





Имя	Значение
▸  c	0x000000340465d540 {0x000000340465d680 {0x00000000}}
▸  c[0]	0x000000340465d680 {0x00000000}
▸  c[1]	0x000000340465d810 {0x00000001}
▸  c[2]	0x000000340465da90 {0x00000002}

Память 1

Адрес: 0x000000340465D680

0x000000340465D680	00	00	00	00	01	00	00	00	02	00	00	00	fd	fd	fd	fd	00	00
0x000000340465D6A7	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x000000340465D6CE	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x000000340465D6F5	00	00	00	f0	da	65	04	34	00	00	00	50	4f	0e	4b	f8	7f	00
0x000000340465D71C	fd	fd	fd	fd	5f	4e	4f	5f	44	45	42	55	47	5f	48	45	41	50
0x000000340465D743	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x000000340465D76A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x000000340465D791	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x000000340465D7B8	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x000000340465D7DF	90	50	d6	65	04	34	00	00	00	60	da	65	04	34	00	00	00	00
0x000000340465D806	00	00	51	00	00	00	fd	fd	fd	fd	01	00	00	00	02	00	00	00
0x000000340465D82D	10	00	80	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x000000340465D854	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x000000340465D87B	4a	00	11	00	80	00	00	00	00	00	00	00	00	00	00	00	00	00
0x000000340465D8A2	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x000000340465D8C9	cb	7a	4a	00	12	00	80	00	00	00	00	00	00	00	00	00	00	00
0x000000340465D8F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x000000340465D917	00	cf	cb	47	4a	00	13	00	80	00	00	00	00	00	00	00	00	00
0x000000340465D93E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x000000340465D965	00	00	00	c8	cb	4c	4a	00	14	00	80	00	00	00	00	00	00	00
0x000000340465D98C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x000000340465D9B3	00	00	00	00	00	c5	cb	49	4a	00	15	00	80	00	00	00	00	00
0x000000340465D9DA	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x000000340465DA01	00	00	00	00	00	00	00	fe	cb	56	4a	00	16	00	80	00	00	00
0x000000340465DA28	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x000000340465DA4F	00	00	00	00	00	00	00	00	fb	cb</								

Контрольные значения 1

Имя	Значение
 c	0x000000340465d540 {0x000000340465d680 {0x00000000}}
 c[0]	0x000000340465d680 {0x00000000}
 c[1]	0x000000340465d810 {0x00000001}
 c[2]	0x000000340465da90 {0x00000002}

УКАЗАТЕЛИ И СТРУКТУРЫ

Объявим структуру для описания пони:

```
typedef struct {  
    int age;  
    char* name;  
} PONY;
```

Рассмотрим задачу формирования динамического массива структур для описания «группы пони».

Необходимо обратить внимание на:

- Операторы доступа к полям структуры (-> и .)
- Передачу структуры в функцию через указатели
- Смысл двойных указателей в примере
- Приоритет операций внутри функции для работы с полями структуры

```

void add_pony(PONY** pony)
{
    const int name_size = 10;
    *pony = (PONY*)calloc(1, sizeof(PONY));
    (*pony)->age = rand() % 100;
    (*pony)->name = (char*)calloc(name_size, sizeof(char));
    for (int i = 0; i < name_size; i++) {
        (*pony)->name[i] = '0' + rand() % 72;
    }
    (*pony)->name[name_size - 1] = 0;
}

void release_pony_info(PONY* pony)
{
    free(pony->name);
    free(pony);
}

void print_pony_info(PONY* pony)
{
    printf("This pony's name is '%s'\tShe is %d years old.\n",
    pony->name, pony->age);
}

```

```

#include <time.h>
#include <stdlib.h>

int main()
{
    const int count = 3;
    PONY** herd = NULL;

    srand(time(NULL));
    herd = (PONY**)calloc(count, sizeof(PONY*));
    for (int i = 0; i < count; i++) {
        add_pony(&herd[i]);
    }

    for (int i = 0; i < count; i++) {
        print_pony_info(herd[i]);
    }

    for (int i = 0; i < count; i++) {
        release_pony_info(herd[i]);
    }
    free(herd);
}

```

```

This pony's name is 'DuPUW^E4[' She is 38 years old.
This pony's name is 'Z2rBoFcmJ' She is 25 years old.
This pony's name is '=MdIk8]^V' She is 96 years old.

```

СПИСКИ

Список – это структура, в которой есть ссылка на саму себя.

Описание односвязного списка:

```
struct <имя> {  
    <тип_1> <переменная_1>,  
    ...  
    <тип_N> <переменная_N>,  
    <имя>* next,  
}
```

```
struct NODE {  
    int element;  
    NODE* next;  
};
```

Описание двусвязного списка:

```
struct <имя> {  
    <тип_1> <переменная_1>,  
    ...  
    <тип_N> <переменная_N>,  
    <имя>* left,  
    <имя>* right,  
}
```

```
struct NODE {  
    int element;  
    NODE* left;  
    NODE* right;  
};
```



```
struct NODE {
    int element;
    NODE* next;
};

void init_node(NODE* node)
{
    (*node).element = rand() % 100;
    (*node).next = NULL;
}

void add_node(NODE* src, NODE* dst)
{
    (*src).next = dst;
}
```

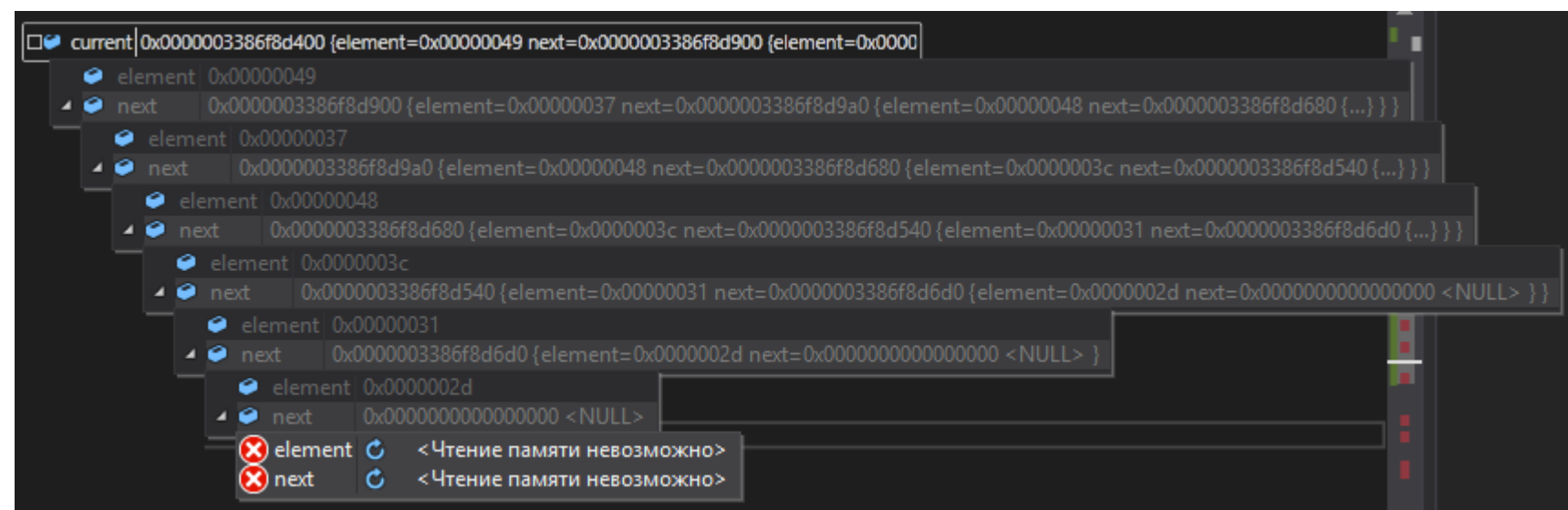
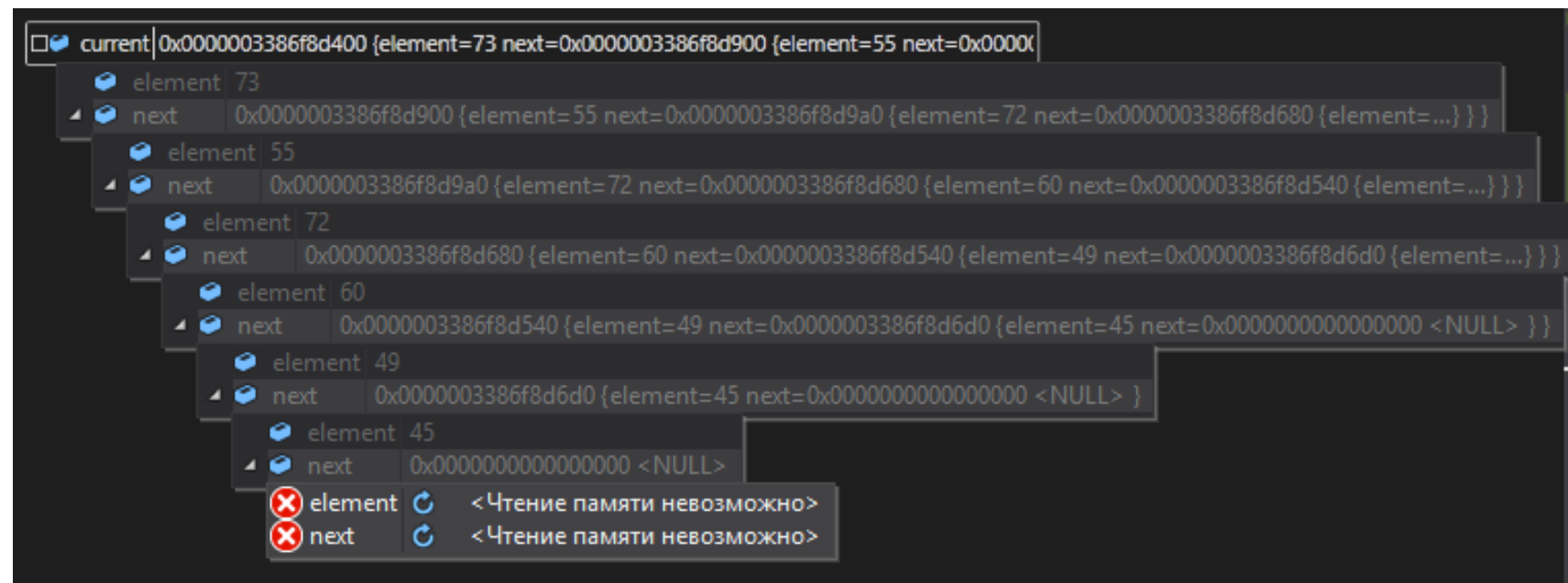
```
srand(time(NULL));
const int count = 5;
NODE* root = NULL;
NODE* current = NULL;

root = (NODE*)calloc(1, sizeof(NODE));
init_node(root);
current = root;
for (int i = 0; i < count; i++) {
    NODE* next = (NODE*)calloc(1, sizeof(NODE));
    init_node(next);
    add_node(current, next);
    current = next;
}

current = root;
while (current) {
    printf("Node element = %d\n", current->element);
    current = current->next;
}

current = root;
while (current) {
    NODE* next = current->next;
    free(current);
    current = next;
}
```

```
Node element = 73
Node element = 55
Node element = 72
Node element = 60
Node element = 49
Node element = 45
```



```
struct NODE {
    int element;
    NODE* left;
    NODE* right;
};

void init_node(NODE* node)
{
    (*node).element = rand() % 100;
    (*node).left = NULL;
    (*node).right = NULL;
}

void add_node(NODE* current, NODE* next)
{
    (*current).left = next;
    (*next).right = current;
}
```

```
srand(time(NULL));
const int count = 5;
NODE* root = NULL;
NODE* current = NULL;

root = (NODE*)calloc(1, sizeof(NODE));
init_node(root);
current = root;
for (int i = 0; i < count; i++) {
    NODE* next = (NODE*)calloc(1, sizeof(NODE));
    init_node(next);
    add_node(current, next);
    current = next;
}
current = root;
while (current) {
    printf("Node element = %d\n", current->element);
    printf("Node element left = %d\n",
           current->left ? current->left->element : -1);
    printf("Node element right = %d\n\n",
           current->right ? current->right->element : -1);
    current = current->left;
}
current = root;
while (current) {
    NODE* next = current->left;
    free(current);
    current = next;
}
```

```
Node element = 17
Node element left = 88
Node element right = -1
```

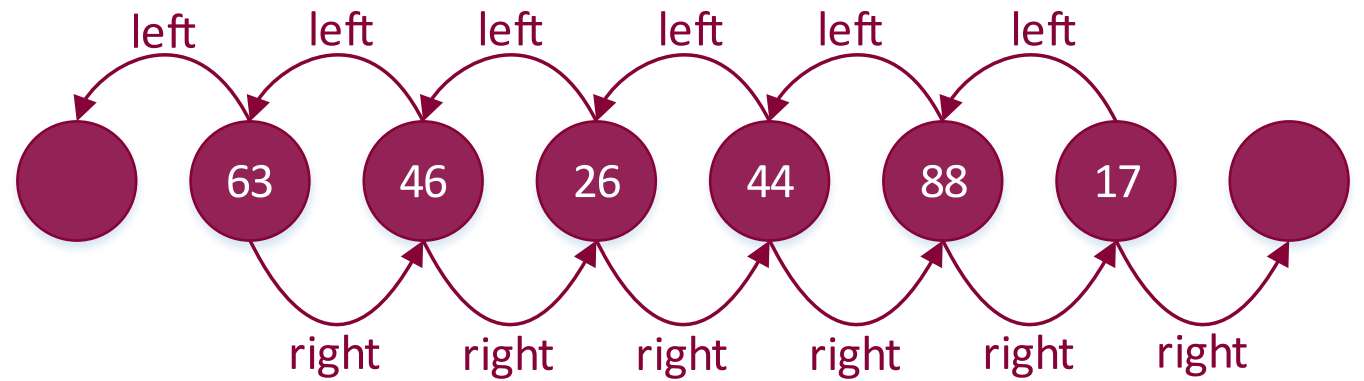
```
Node element = 88
Node element left = 44
Node element right = 17
```

```
Node element = 44
Node element left = 26
Node element right = 88
```

```
Node element = 26
Node element left = 46
Node element right = 44
```

```
Node element = 46
Node element left = 63
Node element right = 26
```

```
Node element = 63
Node element left = -1
Node element right = 46
```



Упражнение 1. Используя указатели, напишите функцию *strcat* (функция *strcat(s,t)* копирует строку *t* в конец строки *s*).

Упражнение 2. Напишите функцию *strend(s,t)*, которая выдает 1, если строка *t* расположена в конце строки *s*, и нуль в противном случае.

Упражнение 3. Напишите варианты библиотечных функций *strncpy*, *strncat* и *strncpy*, которые оперируют с первыми символами своих аргументов, число которых не превышает *n*.

Например, *strncpy(t,s,n)* копирует не более *n* символов *t* в *s*.

Упражнение 4. В функциях *day_of_year* и *month_day* нет никаких проверок правильности вводимых дат. Устраните этот недостаток.

Упражнение 5. Развернуть однонаправленный список.

Упражнение 6. Развернуть двунаправленный список.

Упражнение 7. Написать макроопределение, принимающий два параметра. Макрос должен заменить все вхождения второго параметра на *@*

Упражнение 8. Написать функцию, принимающую в качестве примера строку и символ и возвращающую число символов между первым и вторым вхождениями символа-параметра в строку-параметре.

Упражнение 9. Написать функцию, принимающую строку и символ. Из строки удалить все вхождения символа.

Упражнение 10. Сколько раз программа выведет "kotiche" ?

Упражнение 10. Код:

```
#define if(expr) if(0 && (expr))

int main()
{
    for (int i = 0; i < 10; i++) {
        if (i > 5) {
            printf("kotiche\n");
        }
    }
}
```