



Библиотеки

Библиотеки

Статические

Подключаются к программе во время **компоновки**.

Динамические

Подключаются к программе во время **выполнения программы**



Статические библиотеки

Библиотеки

Статические

```
gcc -c main.c -o main1.o  
gcc -c lib.c -o lib.o  
ar cr libmain.a lib.o  
gcc main1.o libmain.a -o
```

Динамические

```
gcc -c main.c -o main2.o  
gcc -c lib.c -o lib.o  
gcc -shared -o libmain1.so lib.o  
gcc main2.o libmain1.so -Wl,-rpath,. -o main2
```



Статические библиотеки

Библиотеки

Статические

```
gcc -c main.c -o main1.o
gcc -c lib.c -o lib.o
ar cr libmain.a lib.o
gcc main1.o libmain.a -o
```

Выполните команды

```
ldd main1
```

```
ldd main2
```

Объясните полученный
результат

Динамические

```
gcc -c main.c -o main2.o
gcc -c lib.c -o lib.o
gcc -shared -o libmain1.so lib.o
gcc main2.o libmain1.so -Wl,-rpath,. -o main2
```



Статические библиотеки

Выполните команды
objdump main1 -d
objdump main2 -d
Найдите отличия в файлах

Библиотеки

Выполните команды
ldd main1
ldd main2
Объясните полученный
результат

Статические

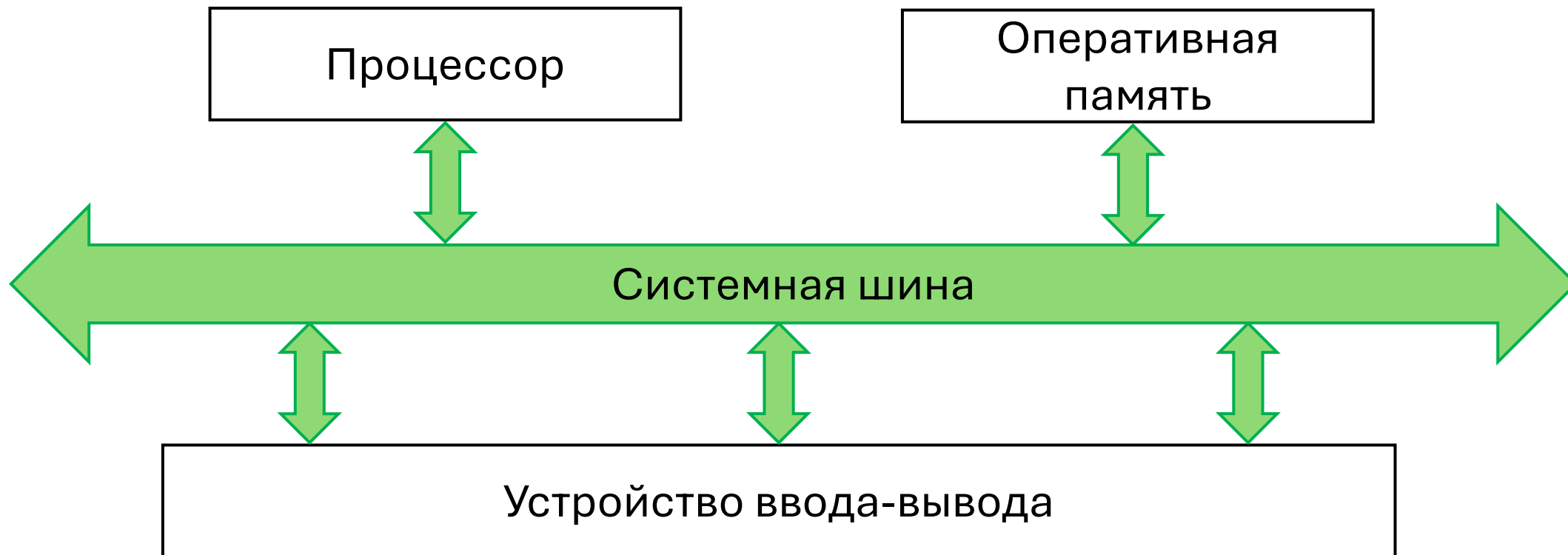
Динамические

```
gcc -c main.c -o main1.o
gcc -c lib.c -o lib.o
ar cr libmain.a lib.o
gcc main1.o libmain.a -o
```

```
gcc -c main.c -o main2.o
gcc -c lib.c -o lib.o
gcc -shared -o libmain1.so lib.o
gcc main2.o libmain1.so -Wl,-rpath,. -o main2
```

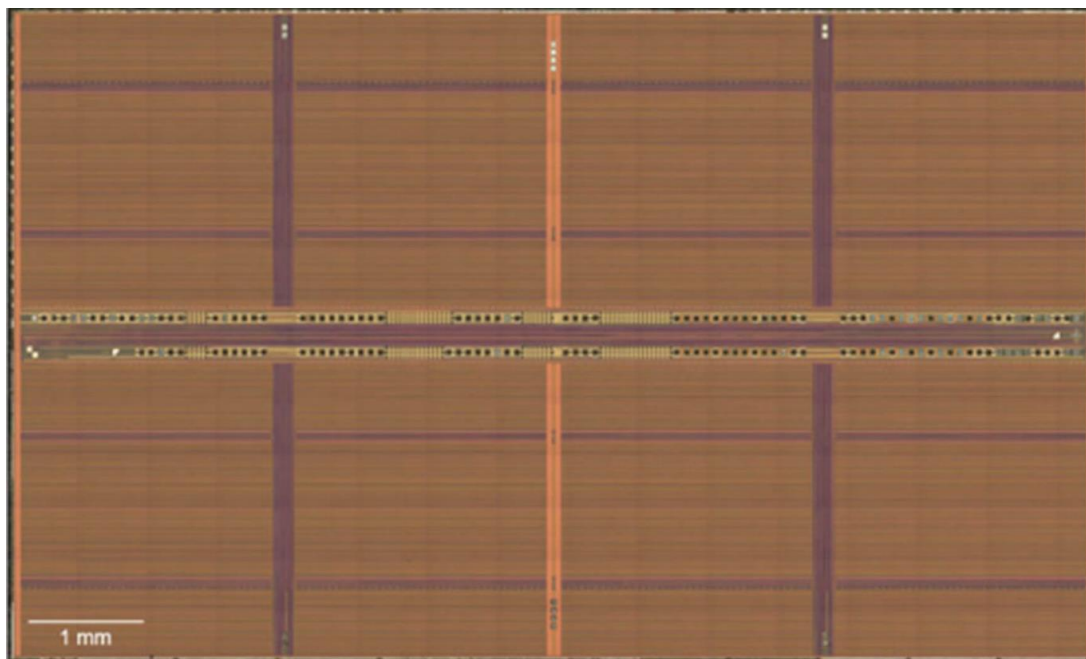


Схема организации ЭВМ

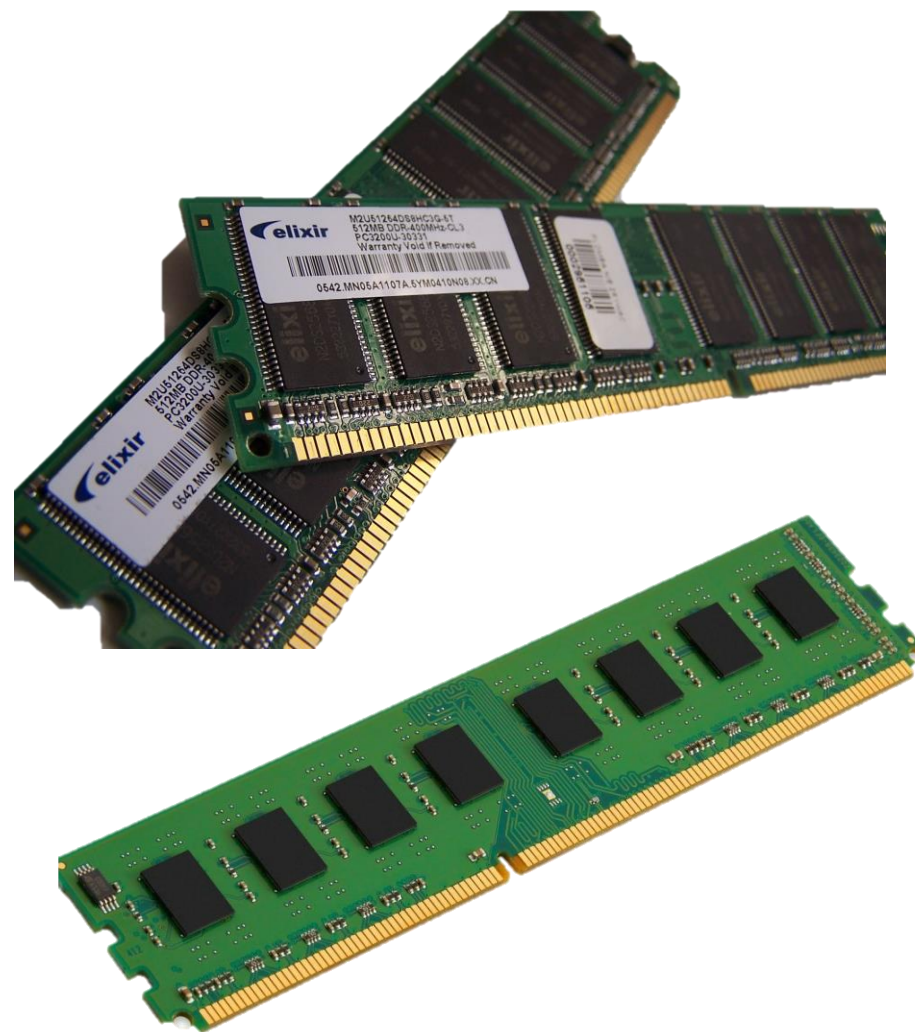




Оперативная память

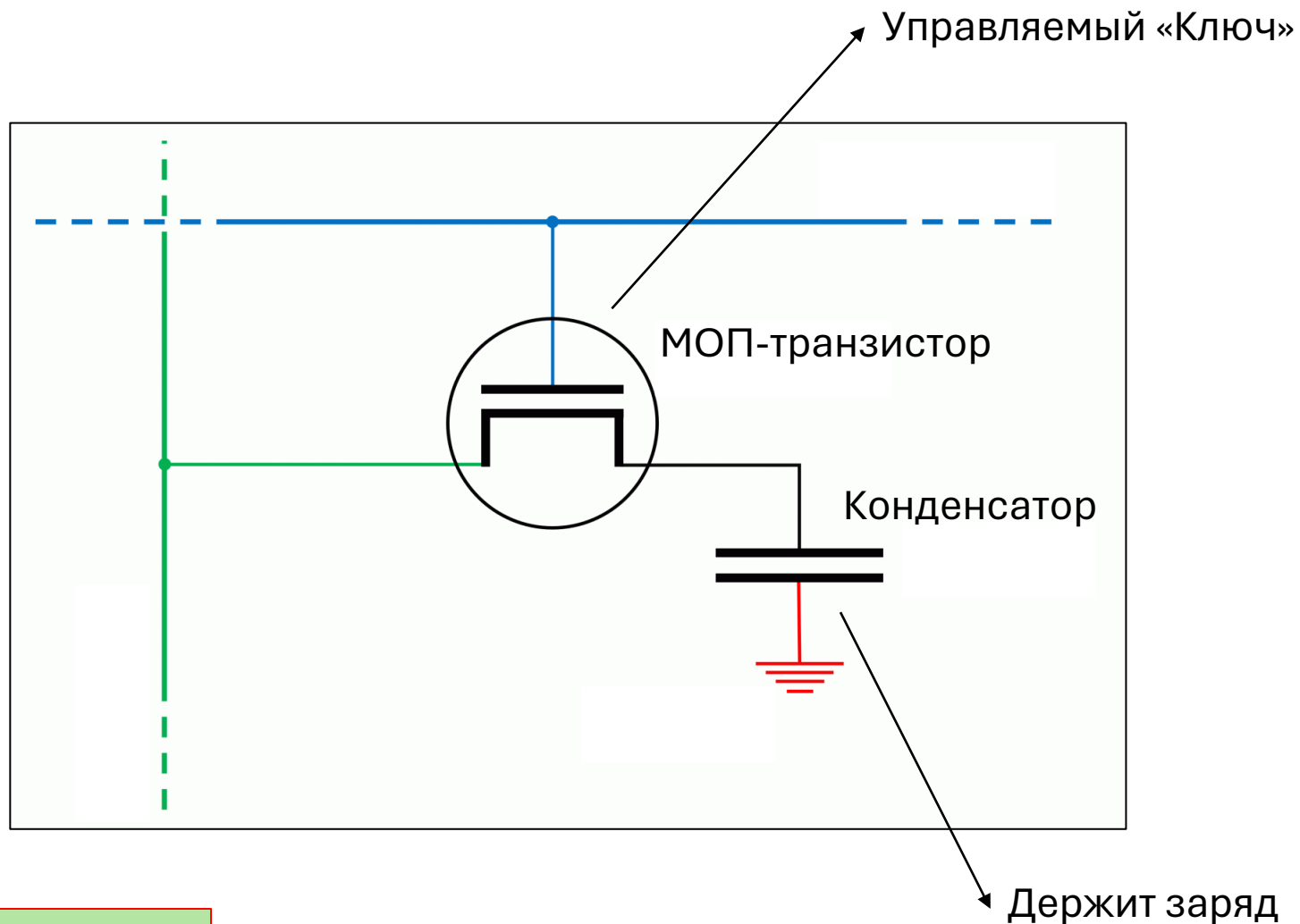
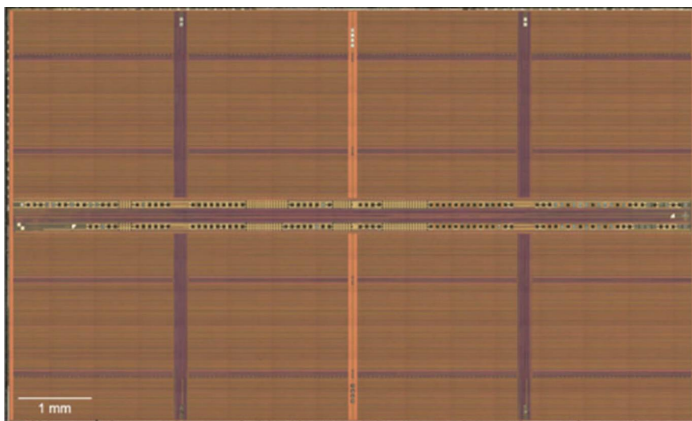


Чип DRAM под микроскопом





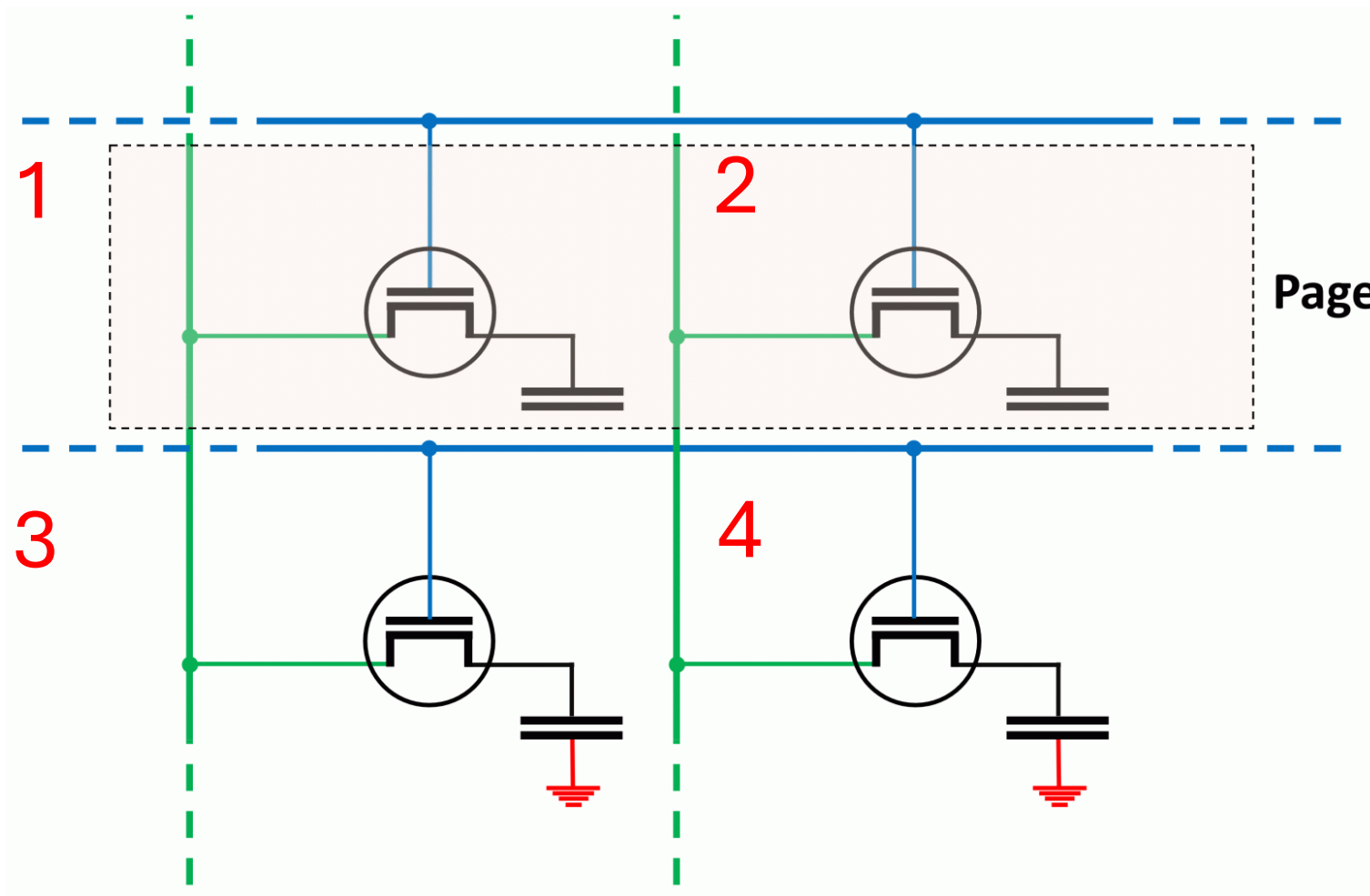
Оперативная память



Конденсаторы не способны хранить заряд вечно и каждую ячейку памяти нужно обновлять по 15-30 раз в секунду.



Оперативная память





Память





Хранение данных

Тип данных	Назначение
int	Целое число
char	Символ
float	Вещественное число одинарной точности
double	Вещественное число двойной точности

Размер?

MS-DOS
(DOSBox)

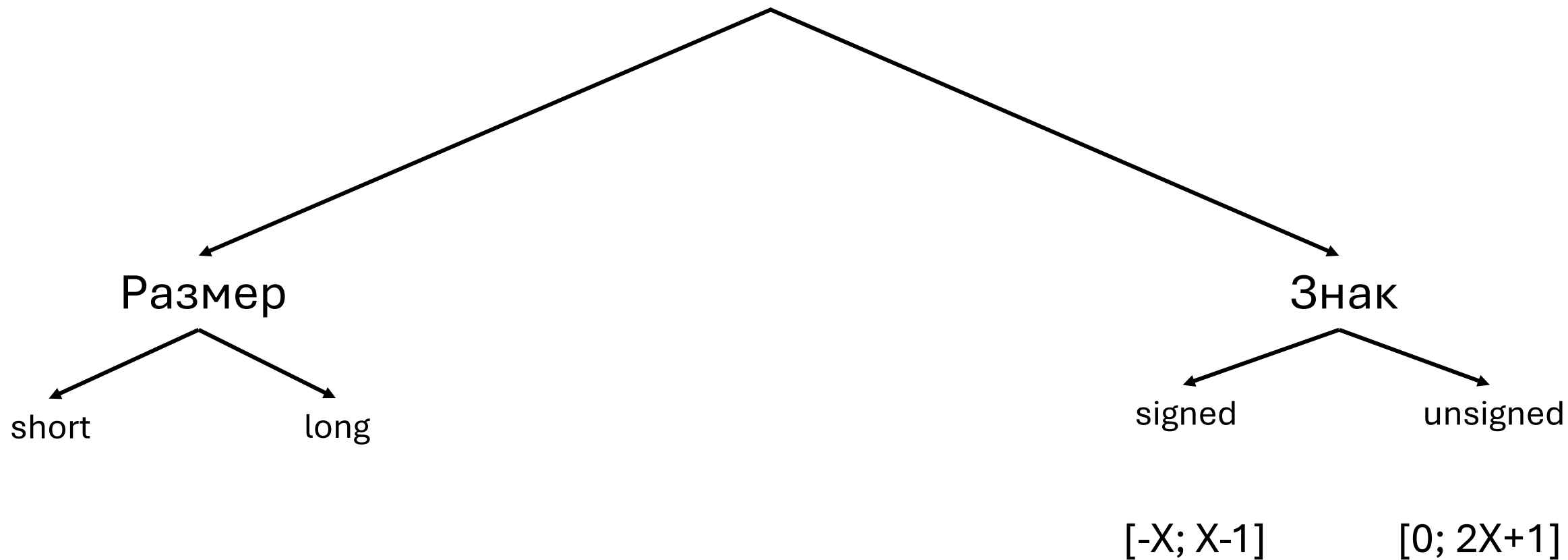
```
C:\PROGRAM>c:\Borlandc\bin\bc.exe  
Sizeof int = 2 bytes  
Sizeof char = 1 bytes  
Sizeof float = 4 bytes  
Sizeof double = 8 bytes
```

Win10

```
C:\Users\79629\Programs>varsize.exe  
Sizeof int = 4 bytes  
Sizeof char = 1 bytes  
Sizeof float = 4 bytes  
Sizeof double = 8 bytes
```



Хранение данных. Модификаторы.





Хранение данных. Модификаторы.

MS-DOS

(DOSBox)

```
C:\PROGRAM>c:\Borlandc\bin\bc.exe  
Sizeof int = 2 bytes  
Sizeof char = 1 bytes  
Sizeof float = 4 bytes  
Sizeof double = 8 bytes
```

With modifiers

```
Sizeof short int = 2 bytes  
Sizeof long int = 4 bytes  
Sizeof long long int = 4 bytes
```

Win10 (x64)

```
C:\Users\79629\Programs>varsize.exe  
Sizeof int = 4 bytes  
Sizeof char = 1 bytes  
Sizeof float = 4 bytes  
Sizeof double = 8 bytes
```

With modifiers

```
Sizeof short int = 2 bytes  
Sizeof long int = 4 bytes  
Sizeof long long int = 8 bytes
```

Linux (x64)

```
sab@LAPTOP-B03PIUAN:~/ubuntu_files$ ./test  
Sizeof int = 4 bytes  
Sizeof char = 1 bytes  
Sizeof float = 4 bytes  
Sizeof double = 8 bytes
```

With modifiers

```
Sizeof short int = 2 bytes  
Sizeof long int = 8 bytes  
Sizeof long long int = 8 bytes
```



Типы данных

stdint.h

Тип	Описание
int8_t	8-битовый со знаком
int16_t	16-битовый со знаком
int32_t	32-битовый со знаком
int64_t	64-битовый со знаком
uint8_t	8-битовый без знака
uint16_t	16-битовый без знака
uint32_t	32-битовый без знака
uint64_t	64-битовый без знака

char <= short <= int <= long <= long
long
float <= double <= long double

Стандарт языка Си никак не оговаривают конкретные размеры типов данных.
Для создания переменных заранее известного размера возможно использовать типы из стандартной библиотеки



Память





Память



Char



Int

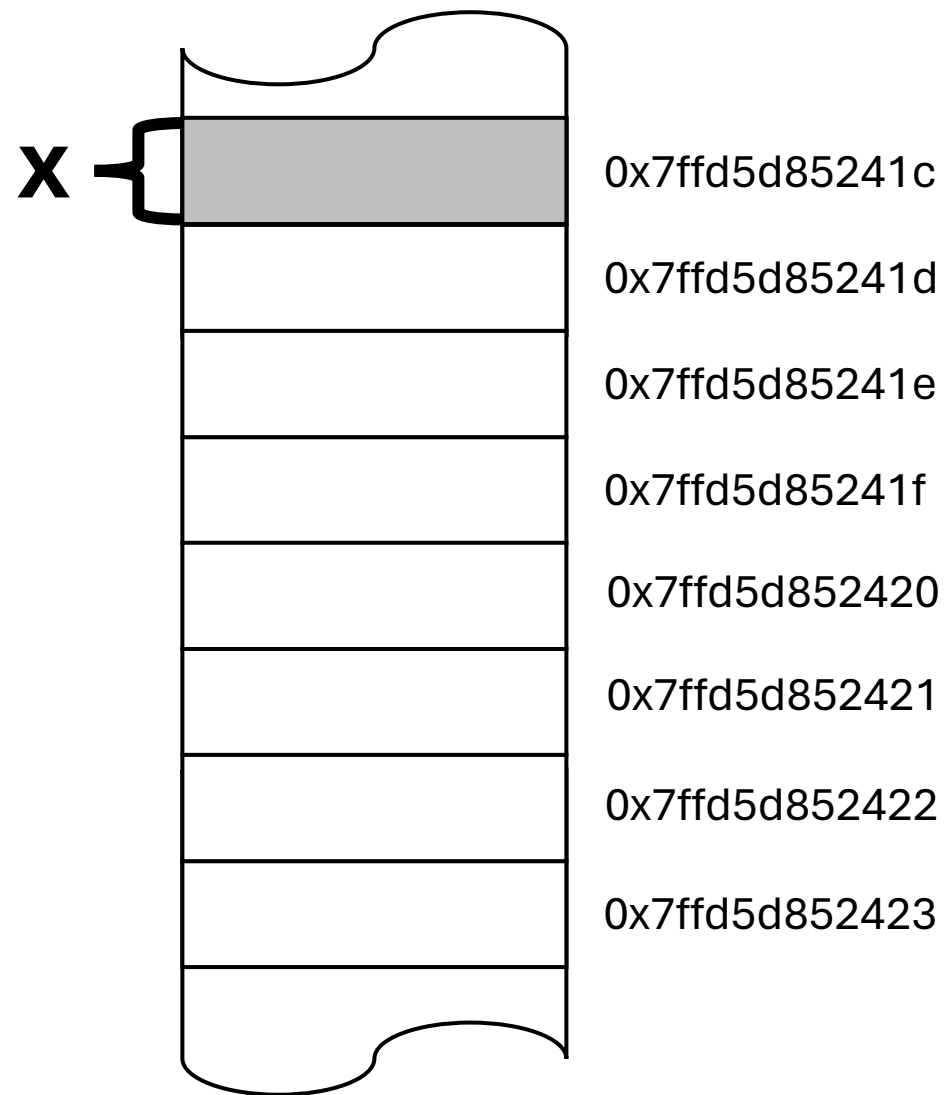


Double



Адреса

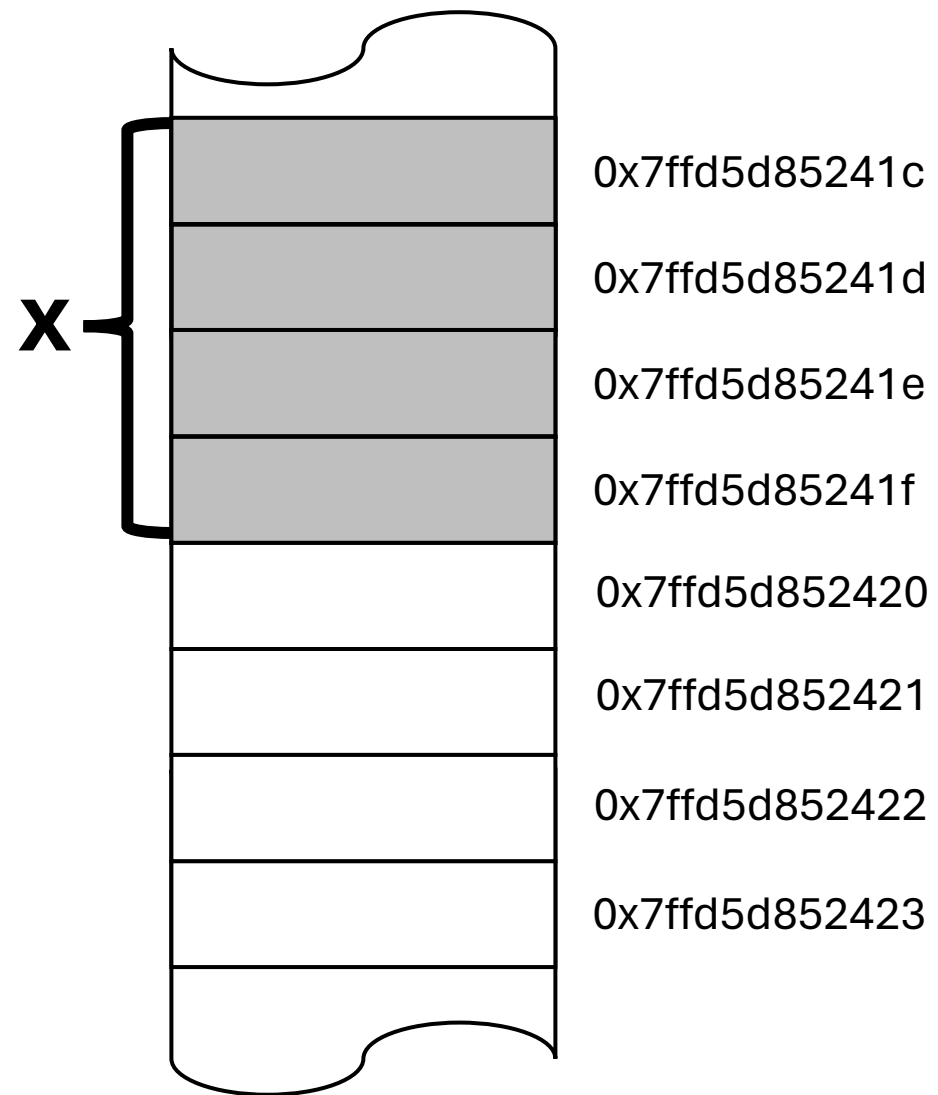
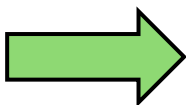
char **x**





Адреса

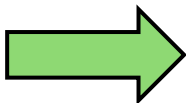
int **x**



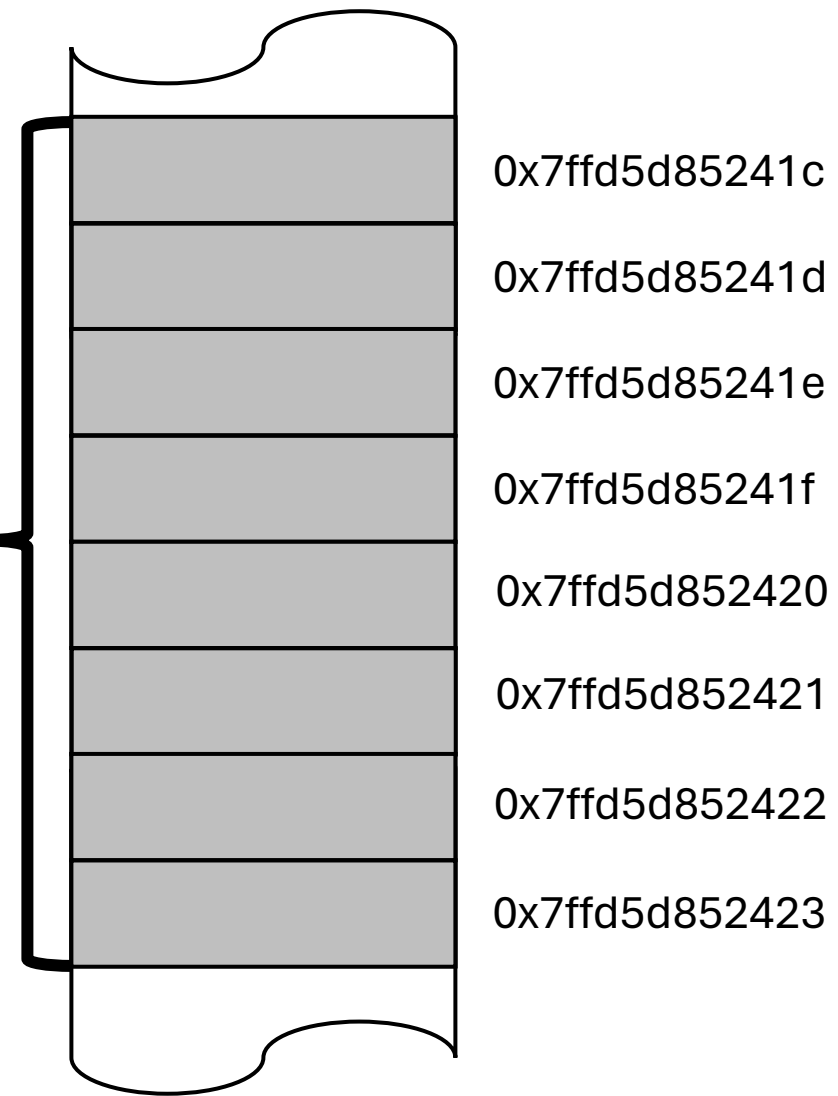


Адреса

double **x**



x





Области видимости переменных (scope)

File scope	Переменные видны в файле
Block scope	Переменные видны в блоке, ограниченном фигурными скобками
Function scope	Переменная видна в рамках области действия конкретной функции (<i>Label</i>)
Function-prototype scope	Переменные видны внутри описания прототипа функции

`int x;`
`int x;` → Так нельзя

`int x;`
{
 `int x;`
}

→ Так можно

Объявление – указание компилятору, что существует переменная определенного типа. Возможно, под неё выделили память, но в другом участке трансляции.

Определение – выделение участка памяти под переменную в **текущем** участке трансляции.



Ещё пару слов о разбиении на файлы

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

int main(void)
{
    int x = summ(5, a);
    printf("%d", x);
    return 0;
}
```

main.c

```
int a = 10;

int summ(int a, int b)
{
    return a + b;
}
```

lib.c

```
#pragma once
int summ(int a, int b);
```

lib.h

Как в main.c увидеть a из lib.c ?



Ещё пару слов о разбиении на файлы

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

int main(void)
{
    int x = summ(5, a);
    printf("%d", x);
    return 0;
}
```

main.c

```
int a = 10;

int summ(int a, int b)
{
    return a + b;
}
```

lib.c

```
#pragma once
int summ(int a, int b);
extern int a;
```

lib.h

Ключевое слово **extern** позволяет **объявить** переменную, но не **определить**



Ещё пару слов о разбиении на файлы

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

int a = 20;
int main(void)
{
    int x = summ(5, a);
    printf("%d", x);
    return 0;
}
```

main.c

```
int a = 10;

int summ(int a, int b)
{
    return a + b;
}
```

lib.c

```
#pragma once
int summ(int a, int b);
```

lib.h

Как обойти двойное определение?



Статические глобальные переменные

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

int a = 20;
int main(void)
{
    int x = summ(5, a);
    printf("%d", x);
    return 0;
}
```

main.c

```
static int a = 10;

int summ(int a, int b)
{
    return a + b;
}
```

lib.c

```
#pragma once
int summ(int a, int b);
```

lib.h

Ключевое слово **static** позволяет сузить область видимости переменной до файла



Статические локальные переменные

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

```
int summ(int a)
{
    int b = 5;
    b++;
    return a + b;
}
```

```
int main(void)
{
    int x = summ(5);
    printf("%d\n", x);
    x = summ(5);
    printf("%d\n", x);
    return 0;
}
```

```
X1 = 11
X2 = 11
```




Статические локальные переменные

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

```
int summ(int a)
{
    static int b = 5;
    b++;
    return a + b;
}
```

```
int main(void)
{
    int x = summ(5);
    printf("%d\n", x);
    x = summ(5);
    printf("%d\n", x);
    return 0;
}
```

X1	=	11
X2	=	12



Ключевые слова: const

const – ключевое слово, указывающее компилятору, что данные неизменяемы.

```
const int x = 10;    //константная переменная
x = 15;
```

```
int * ptr;           //указатель
*ptr = 0;            // +
ptr = NULL;          // +
```

```
int const * ptrToConst; //указатель на константу
*ptrToConst = 0;        // -
ptrToConst = NULL;      // +
```

```
int * const constPtr;  //константный указатель
*constPtr = 0;          // +
constPtr = NULL;        // -
```

```
int const * const constPtrToConst; //константный указатель на константу
*constPtrToConst = 0;               // -
constPtrToConst = NULL;              // -
```



Ключевые слова: const

```
int x = 10;  
const int * ptrx = &x;  
*ptrx = 11;      //менять не можем  
x++;             //спокойно меняем, ptrx указывает на новое значение
```



Передача в функцию значений

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

```
int f(const int *x){  
    *x = 15;    //-  
    x = NULL;   //+  
}
```

```
int main()  
{  
    int x = 10;  
    f(&x);  
}
```



volatile

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

void f(){
    uint32_t * pReg = (uint32_t *) 0x1234;
    while (*pReg == 0) {
        printf("%s\n", "hello!");
        // тут происходит что-то полезное
    }
}
```

```
.LC0:
    .string "hello!"

f:
    movzx    eax, BYTE PTR ds:4660
    test     al, al
    jne      .L6
    sub      rsp, 8
.L3:
    mov      edi, OFFSET FLAT:.LC0
    call     puts
    movzx    eax, BYTE PTR ds:4660
    test     al, al
    je       .L3
    add      rsp, 8
    ret
.L6:
    ret
```

Без оптимизации все хорошо.

Volatile — ключевое слово языков C/C++, которое информирует компилятор о том, что значение переменной может меняться из вне и что компилятор не будет оптимизировать эту переменную.



volatile

-01

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

void f(){
    uint32_t * pReg = (uint32_t *) 0x1234;
    while (*pReg == 0) {
        printf("%s\n", "hello!");
        // тут происходит что-то полезное
    }
}
```

```
.LC0:
    .string "hello!"

f:
    cmp     BYTE PTR ds:4660, 0
    jne     .L6
    sub     rsp, 8
.L3:
    mov     edi, OFFSET FLAT:.LC0
    call    puts
    cmp     BYTE PTR ds:4660, 0
    je      .L3
    add     rsp, 8
    ret
.L6:
    ret
```

Все не очень хорошо! Значение pReg скопировалось в цикл и он превратился в бесконечный



volatile

-01

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

void f(){
    uint32_t volatile * pReg = (uint32_t volatile *) 0x1234;
    while (*pReg == 0) {
        printf("%s\n", "hello!");
        // тут происходит что-то полезное
    }
}
```

```
.LC0:
    .string "hello!"

f:
    movzx    eax, BYTE PTR ds:4660
    test     al, al
    jne      .L6
    sub      rsp, 8





.L3:
    mov      edi, OFFSET FLAT:.LC0
    call     puts
    movzx    eax, BYTE PTR ds:4660
    test     al, al
    je       .L3
    add      rsp, 8
    ret

.L6:
    ret
```

Теперь снова все хорошо



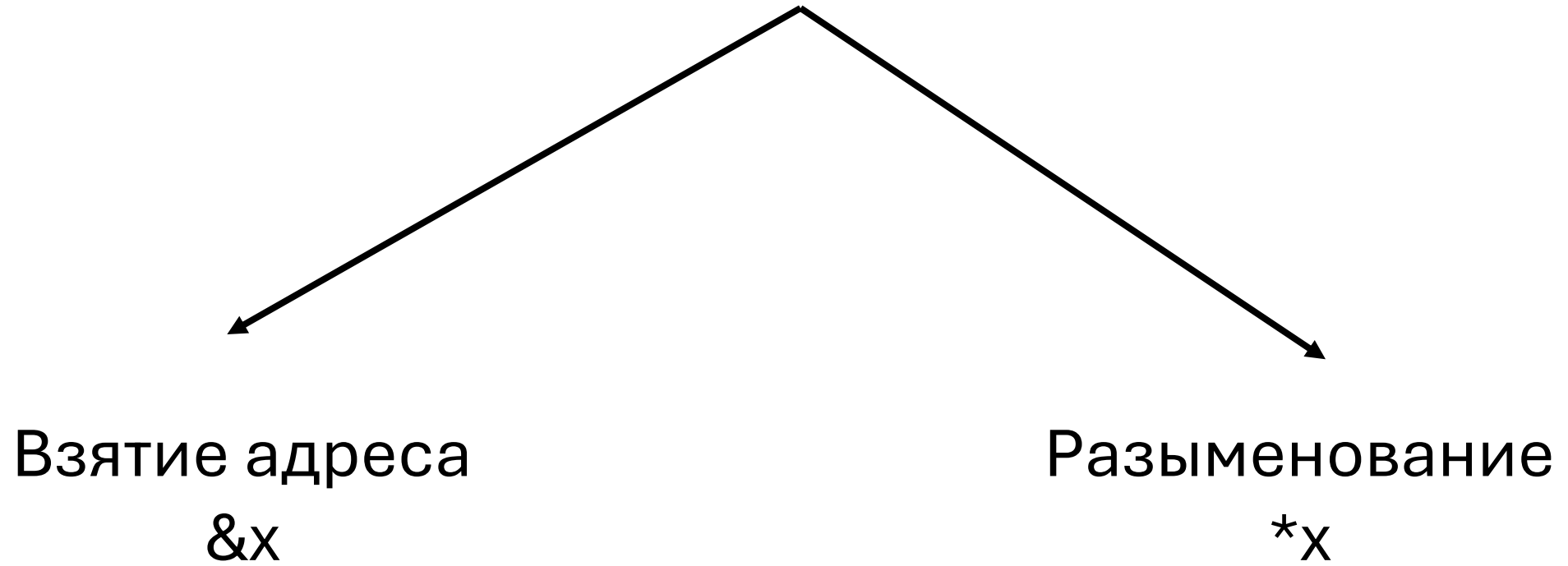
Указатели

char		храним 'char'
int		храним 'int'
double		храним 'double'
int*		храним адрес 'int'

Указатель - это адрес переменной в памяти.



Операции с указателями





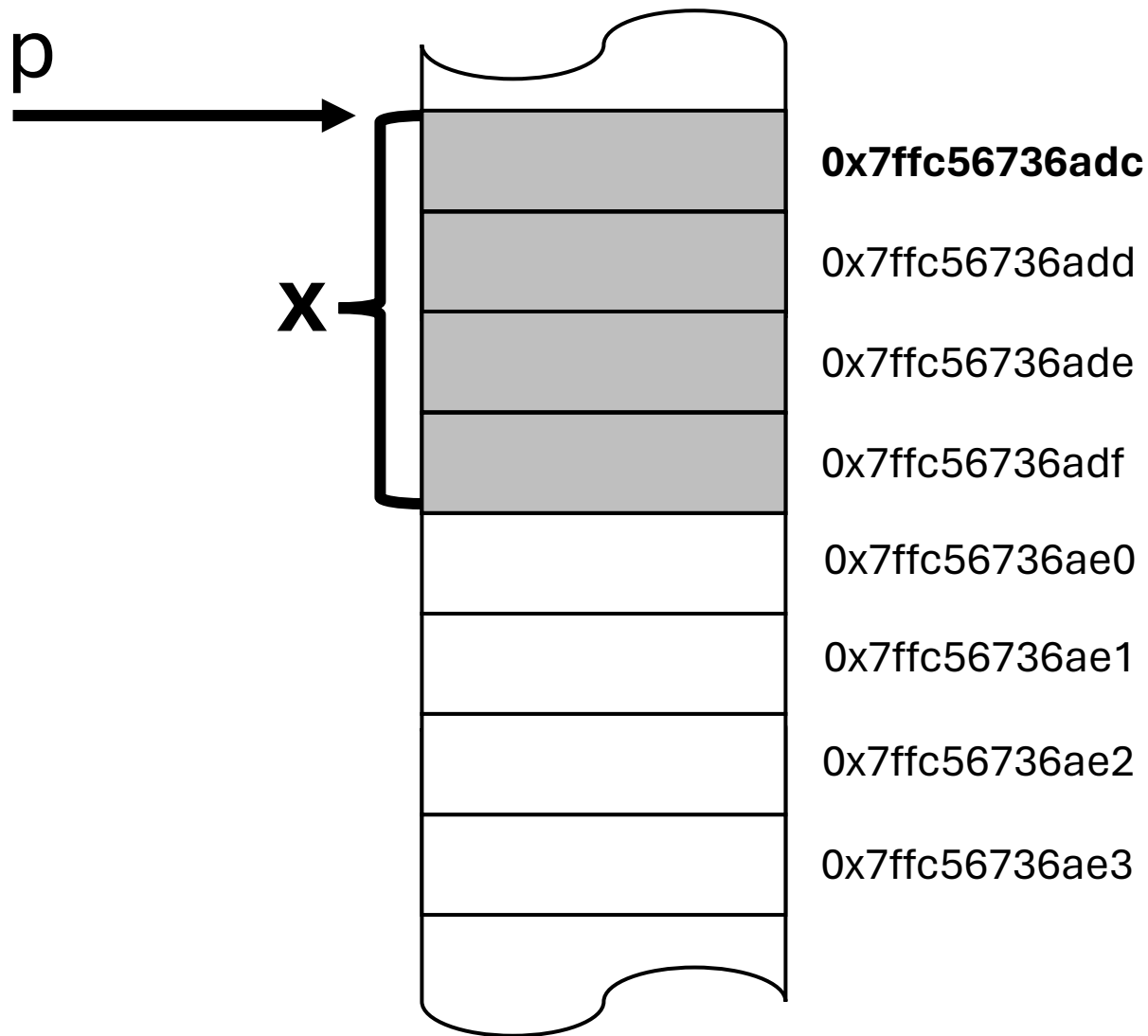
Указатели

```
int x = 10;  
int* p;  
p = &x;  
printf("%p\n", p);  
printf("%d\n", *p);  
printf("%p\n", &p);
```

p=0x7ffc56736adc

*p=10

&p=0x7ffc56736ad0

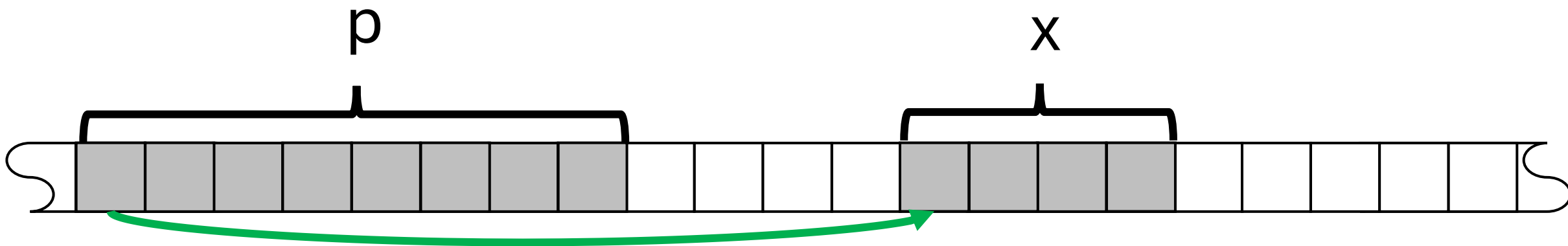


Чему равен размер указателя?



Указатели

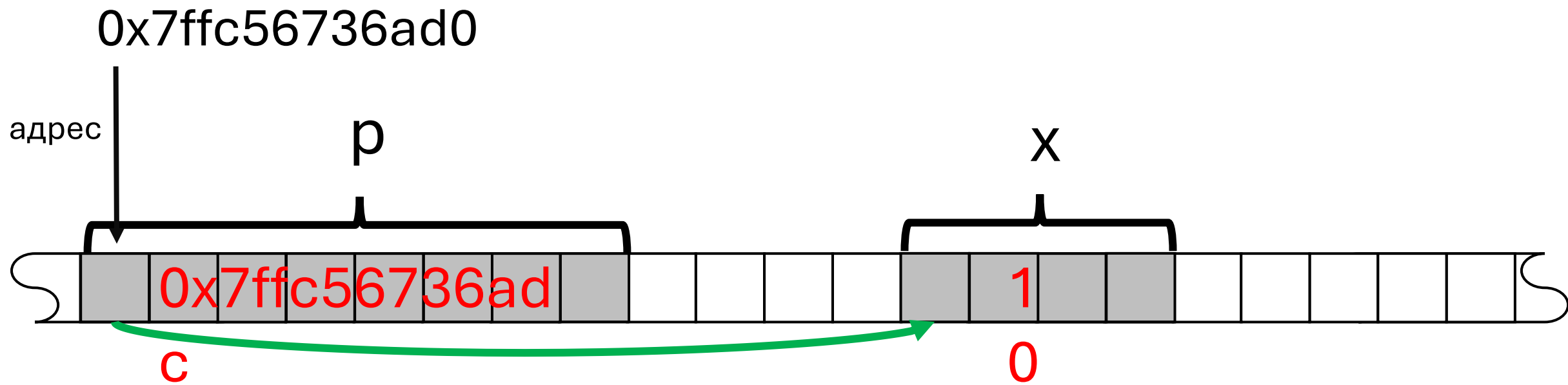
```
int x = 10;  
int* p;  
p = &x;
```





Указатели

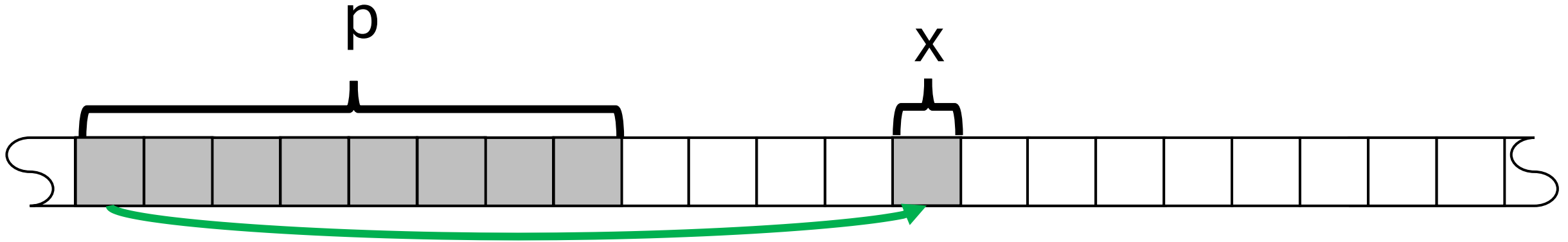
```
int x = 10;  
int* p;  
p = &x;
```





Указатели

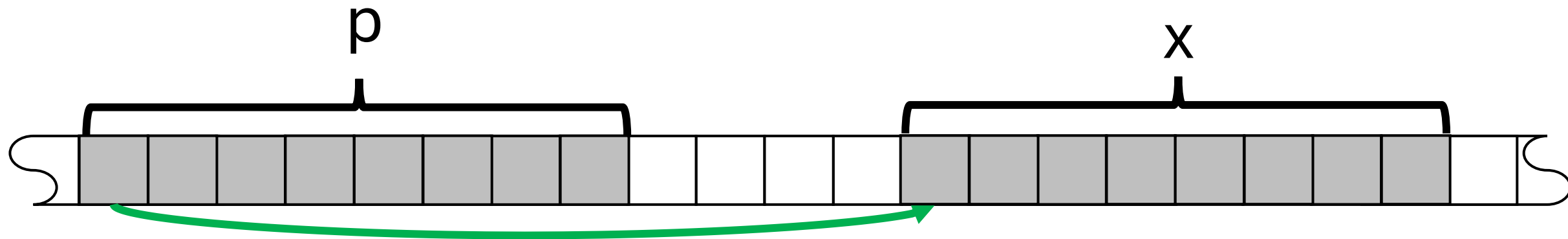
```
char x = 'A';  
char* p;  
p = &x;
```





Указатели

```
double x = 12.345678;  
double* p;  
p = &x;
```





Указатели

```
#include <stdio.h>
int main()
{
    int x = 10;
    int* y = &x;
    printf("x = %d\n", x);
    *y = *y + 1;
    printf("x = %d\n", x);
    return 0;
}
```

Что напечатает данная программа?



Указатели

```
#include <stdio.h>
int main()
{
    int x = 10;
    int* y = &x;
    printf("x = %d\n", x);
    *y = *y + 1;
    printf("x = %d\n", x);
    return 0;
}
```

Что напечатает данная программа?

```
x = 10
x = 11
```




Адресная арифметика

Сложение: (адрес в указателе) + (значение `int_выражения`)*sizeof(<тип>)

```
int x = 10;  
int* ptr;  
ptr = &x;  
ptr = ptr + 1;
```

```
//ptr = 0x0000002ABDBAF974
```

Возвращаемое значение – адрес!



Адресная арифметика

Сложение: (адрес в указателе) + (значение `int_выражения`)*sizeof(<тип>)

```
int x = 10;  
int* ptr;  
ptr = &x;           //ptr = 0x0000002ABDBAF974  
ptr = ptr + 1;      //ptr = 0x0000002ABDBAF978
```

$$0x0000002ABDBAF978 = 0x0000002ABDBAF974 + 1 * 4$$



Адресная арифметика

Вычитание: (адрес в указателе) - (значение int_выражения)*sizeof(<тип>)

```
int x = 10;  
int* ptr;  
ptr = &x;  
ptr = ptr - 1;
```

```
//ptr = 0x0000003E8D2FF834  
//ptr = 0x0000003E8D2FF830
```

$$0x0000003E8D2FF830 = 0x0000003E8D2FF834 - 1*4$$

Возвращаемое значение – адрес!



Адресная арифметика

Индексация: $\text{*((адрес в указателе) + (значение индекса) * sizeof(<тип>))}$

```
int x = 10;  
int* ptr;  
ptr = &x; //ptr = 0x0000001F33AFFB24  
ptr = &ptr[1]; //ptr = 0x0000001F33AFFB28
```

1. ЧИСЛО = $\text{*(0x0000001F33AFFB24 + 1*4)}$

2. $\text{0x0000001F33AFFB28 = \&(ЧИСЛО)}$

Напомнила ли вам что-нибудь
подобная запись?

Возвращаемое значение – значение по адресу!



Память

```
int x[4];
```





Тест

Какие из предложенных вариантов
обращения к элементу массива верные?
`int x[3] = {1,2,3};`

`x[1]`

`*(&x[1])`

`*(x+1)`

`1[x]`



Тест

Какие из предложенных вариантов
обращения к элементу массива верные?
`int x[3] = {1,2,3};`

`x[1]`

`*(&x[1])`

`*(x+1)`

`1[x]`



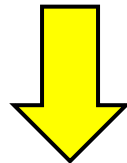
Указатели и массивы

Заметим!

```
ptr = &ptr[1];
```



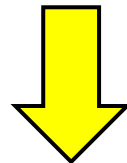
```
ptr = ptr + 1;
```



```
int x = ptr[1];
```



```
int x = *(ptr + 1);
```



Массивы

?

Указатели



Указатели и массивы

```
#include <stdio.h>
int main()
{
    int x[5];
    for(int i = 0; i < 5; i++)
        x[i] = i + 1;

    for (int i = 0; i < 5; i++)
        printf("%d ", x[i]);
    printf("\n");

    x[2] = 10;

    for (int i = 0; i < 5; i++)
        printf("%d ", x[i]);
    printf("\n");

    *(x + 2) = 3;
    for (int i = 0; i < 5; i++)
        printf("%d ", *(x + i));

    return 0;
}
```



Указатели и массивы

```
#include <stdio.h>
int main()
{
    int x[5];
    for(int i = 0; i < 5; i++)
        x[i] = i + 1;

    for (int i = 0; i < 5; i++)
        printf("%d ", x[i]);
    printf("\n");

    x[2] = 10;

    for (int i = 0; i < 5; i++)
        printf("%d ", x[i]);
    printf("\n");

    *(x + 2) = 3;
    for (int i = 0; i < 5; i++)
        printf("%d ", *(x + i));

    return 0;
}
```

Результат

1	2	3	4	5
1	2	10	4	5
1	2	3	4	5



The Ksplice Pointer Challenge

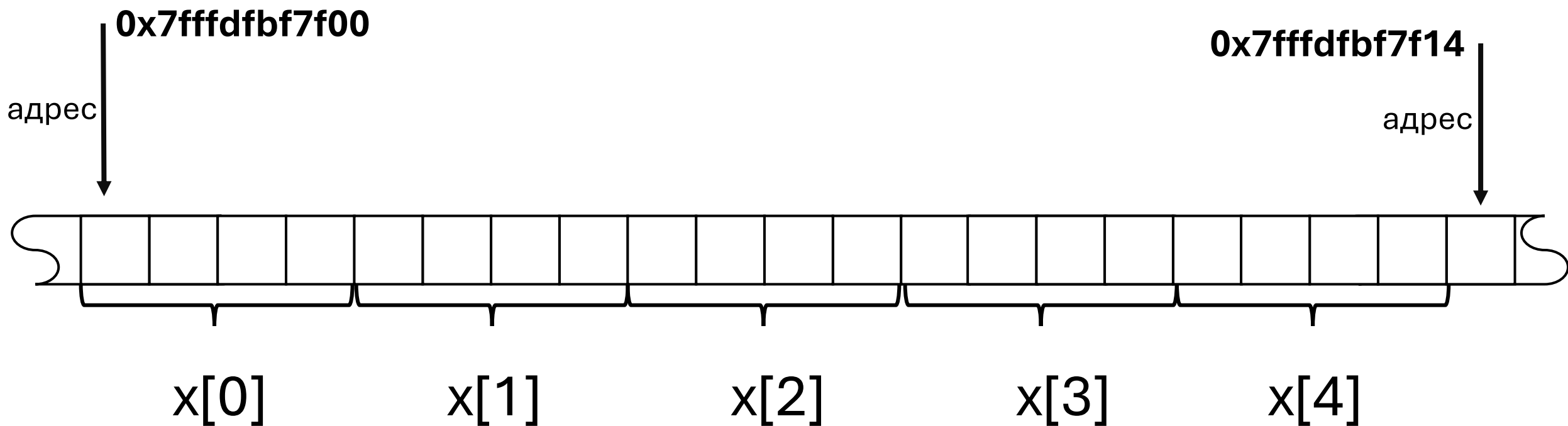
```
#include <stdio.h>
```

```
int main()
{
    int x[5];
    printf("x=\t%p\n", x);
    printf("x+1=\t%p\n", x + 1);
    printf("&x=\t%p\n", &x);
    printf("&x+1\t%p\n", &x + 1);
    return 0;
}
```

Что напечатает данная программа?



The Ksplice Pointer Challenge





The Ksplice Pointer Challenge

```
#include <stdio.h>

int main()
{
    int x[5];
    printf("x=\t%p\n",      x);
    printf("x+1=\t%p\n",    x + 1);
    printf("&x=\t%p\n",      &x);
    printf("&x+1\t%p\n",     &x + 1);
    return 0;
}
```



The Ksplice Pointer Challenge

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

```
int main()  
{
```

```
    int x[5];
```

```
    printf("x=\t%p\n", x);
```

```
    printf("x+1=\t%p\n", x + 1);
```

```
    printf("&x=\t%p\n", &x);
```

```
    printf("&x+1\t%p\n", &x + 1);
```

```
    return 0;
```

```
}
```

$x + 0 * \text{sizeof}(\text{int})$

$x + 1 * \text{sizeof}(\text{int})$

$\&x$

$x + 1 * \text{sizeof}(x)$

Работает
как
указатель
Работает
как **массив**



The Ksplice Pointer Challenge

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

```
int main()  
{
```

```
    int x[5];
```

```
    printf("x=\t%p\n", x);
```

```
    printf("x+1=\t%p\n", x + 1);
```

```
    printf("&x=\t%p\n", &x);
```

```
    printf("&x+1\t%p\n", &x + 1);
```

```
    return 0;
```

```
}
```

```
x=          000000139ACFFB48  
x+1=        000000139ACFFB4C  
&x=         000000139ACFFB48  
&x+1        000000139ACFFB5C
```



Указатели и массивы

```
#include <stdio.h>
int main() {
    int x[5] = { 1,2,3,4,5 };
    int(*y) = &x;
    int(*z)[5] = &x;
    printf("x: \t%p\n", x);
    printf("x+1: \t%p\n", x + 1);
    printf("y: \t%p\n", y);
    printf("y+1: \t%p\n", y + 1);
    printf("z: \t%p\n", z);
    printf("z+1: \t%p\n", z + 1);
    printf("*****\n");
    printf("*x: \t%p\n", *x);
    printf("*x+1: \t%p\n", *x + 1);
    printf("*y: \t%p\n", *y);
    printf("*y+1: \t%p\n", *y + 1);
    printf("*z: \t%p\n", *z);
    printf("*z+1: \t%p\n", *z + 1);
    return 0;
}
```

```
x:      0x7ffc8d858330
x+1:    0x7ffc8d858334
y:      0x7ffc8d858330
y+1:    0x7ffc8d858334
z:      0x7ffc8d858330
z+1:    0x7ffc8d858344
*****
*x:      0x1
*x+1:    0x2
*y:      0x1
*y+1:    0x2
*z:      0x7ffc8d858330
*z+1:    0x7ffc8d858334
```




Передача параметров в функцию по указателю

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

```
void swap(int *a, int *b)
{
    int tmp = *b;
    *b = *a;
    *a = tmp;
}
```

```
int main()
{
    int a = 10, b = 15;
    printf("a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("a = %d, b = %d\n", a, b);
    return 0;
}
```

Результат

```
a = 10, b = 15
a = 15, b = 10
```