



# Факультатив по программированию на языке C

## Занятие 6 Устройство памяти (ч.2)



# План занятий

№	Тема	Описание
1	Введение в курс	Языки программирования. Основы работы с Linux.
2	Основы языка C	Написание и компиляция простейших программ с использованием gcc. Правила написания кода.
3	Компиляция	Разбиение программы на отдельные файлы. Make файлы. Компиляция.
4	Ввод данных. Библиотеки	Работа со вводом/выводом. Статические и динамические библиотеки.
5	Хранение данных. Память	Хранение процесса в памяти компьютера. Виртуальная память, сегментация. Секции программы.
6	Хранение данных.	Elf файлы. Указатели и массивы. Типы данных.
7	Обработка данных	Безопасные функции. Битовые операции – сдвиги, логические операции. Битовые поля.
8	Язык ассемблера	Основы анализа программ на языке ассемблер. Gdb и отладка
9	Программирование под встраиваемые ОС	Работа с микрокомпьютером Raspberry Pi

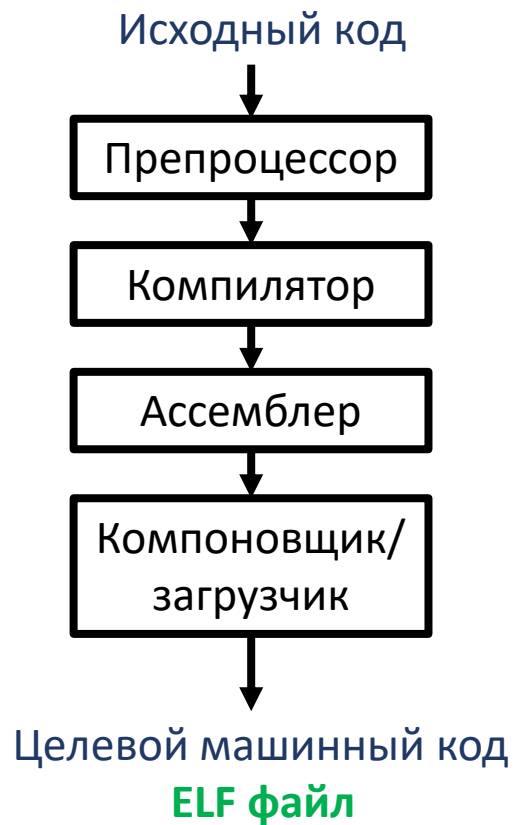


# Дерево языка





# Что?



**ELF** (Executable and Linkable Format) — формат двоичных файлов, определяющий структуру бинарных файлов, библиотек, и файлов ядра

**Процесс** — программа во время исполнения и все её элементы: адресное пространство, глобальные переменные, регистры, стек, счетчик команд, состояние, открытые файлы, дочерние процессы и т. д



# Устройство ELF файлов

Основные компоненты:

1) Заголовок

2) Описание секций

3) Описание сегментов



# Устройство ELF файлов

## Заголовок файла

```
typedef struct
{
    unsigned char e_ident[16];
    uint16_t e_type;
    uint16_t e_machine;
    uint32_t e_version;
    uint32_t e_entry;
    uint32_t e_phoff;
    uint32_t e_shoff;
    uint32_t e_flags;
    uint16_t e_ehsize;
    uint16_t e_phentsize;
    uint16_t e_phnum;
    uint16_t e_shentsize;
    uint16_t e_shnum;
    uint16_t e_shstrndx;
} Elf32_Ehdr;
```

Основные компоненты:

1) Заголовок

2) Описание секций

3) Описание сегментов



# Пример

## ELF Header:

Magic: 7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00  
Class: ELF32  
Data: 2's complement, little endian  
Version: 1 (current)  
OS/ABI: UNIX - System V  
ABI Version: 0  
Type: DYN (Shared object file)  
Machine: Intel 80386  
Version: 0x1  
Entry point address: 0x1070  
Start of program headers: 52 (bytes into file)  
Start of section headers: 18528 (bytes into file)  
Flags: 0x0  
Size of this header: 52 (bytes)  
Size of program headers: 32 (bytes)  
Number of program headers: 12  
Size of section headers: 40 (bytes)  
Number of section headers: 31  
Section header string table index: 30

`readelf memory -h`



# Устройство ELF файлов

## Основные компоненты:

1) Заголовок

2) Описание секций

3) Описание сегментов

Информация, хранящаяся в ELF-файле, организована в секции. Каждая секция имеет свое уникальное имя. Некоторые секции хранят служебную информацию ELF-файла (например, таблицы строк), другие секции хранят отладочную информацию, третьи секции хранят код или данные программы.

## Таблица заголовков секций

```
typedef struct
{
    uint32_t sh_name;
    uint32_t sh_type;
    uint32_t sh_flags;
    uint32_t sh_addr;
    uint32_t sh_offset;
    uint32_t sh_size;
    uint32_t sh_link;
    uint32_t sh_info;
    uint32_t sh_addralign;
    uint32_t sh_entsize;
} Elf32_Shdr;
```





# Пример

## Section Headers:

[Nr]	Name	Type	Addr	Off	Size	ES	Flg	Lk	Inf	Al
[ 0]		NULL	00000000	000000	000000	00		0	0	0
[ 1]	.interp	PROGBITS	000001b4	0001b4	000013	00	A	0	0	1
...										
[12]	.init	PROGBITS	00001000	001000	000024	00	AX	0	0	4
...										
[16]	.text	PROGBITS	00001070	001070	0001e9	00	AX	0	0	16
....										
[18]	.rodata	PROGBITS	00002000	002000	000e1c	00	A	0	0	32
....										
[25]	.data	PROGBITS	00005000	004000	000010	00	WA	0	0	4
[26]	.bss	NOBITS	00005010	004010	000008	00	WA	0	0	4
....										

readelf memory -S



# Устройство ELF файлов

## Основные компоненты:

1) Заголовок

2) Описание секций

3) Описание сегментов

Таблица заголовков программы содержит информацию, необходимую для загрузки программы на выполнение.

## Таблица заголовков программы

```
typedef struct
{
    uint32_t p_type;
    Elf32_Off p_offset;
    Elf32_Addr p_vaddr;
    Elf32_Addr p_paddr;
    uint32_t p_filesz;
    uint32_t p_memsz;
    uint32_t p_flags;
    uint32_t p_align;
} Elf32_Phdr;
```



## Пример

### Program Headers:

	Type	Offset	VirtAddr	PhysAddr	FileSiz	MemSiz	Flg	Align
0	PHDR	0x000034	0x00000034	0x00000034	0x00180	0x00180	R	0x4
1	INTERP	0x0001b4	0x000001b4	0x000001b4	0x00013	0x00013	R	0x1
2	LOAD	0x000000	0x00000000	0x00000000	0x003d4	0x003d4	R	0x1000
3	LOAD	0x001000	0x00001000	0x00001000	0x00274	0x00274	R E	0x1000
4	LOAD	0x002000	0x00002000	0x00002000	0x00f94	0x00f94	R	0x1000
5	LOAD	0x003edc	0x00004edc	0x00004edc	0x00134	0x0013c	RW	0x1000
6	DYNAMIC	0x003ee4	0x00004ee4	0x00004ee4	0x000f8	0x000f8	RW	0x4
...								

readelf memory -l



# Пример

## Program Headers:

	Type	Offset	VirtAddr	PhysAddr	FileSiz	MemSiz	Flg	Align
0	PHDR	0x000034	0x00000034	0x00000034	0x00180	0x00180	R	0x4
1	INTERP	0x0001b4	0x000001b4	0x000001b4	0x00013	0x00013	R	0x1
2	LOAD	0x000000	0x00000000	0x00000000	0x003d4	0x003d4	R	0x1000
3	LOAD	0x001000	0x00001000	0x00001000	0x00274	0x00274	R E	0x1000
4	LOAD	0x002000	0x00002000	0x00002000	0x00f94	0x00f94	R	0x1000
5	LOAD	0x003edc	0x00004edc	0x00004edc	0x00134	0x0013c	RW	0x1000
6	DYNAMIC	0x003ee4	0x00004ee4	0x00004ee4	0x000f8	0x000f8	RW	0x4
...								

## Segment Sections...

```
00
01 .interp
02 .interp .note.gnu.build-id .note.gnu.property .note.ABI-tag .gnu.hash .dynsym .dynstr .gnu.version
03 .init .plt .plt.got .plt.sec .text .fini
04 .rodata .eh_frame_hdr .eh_frame
05 .init_array .fini_array .dynamic .got .data .bss
06 .dynamic
```

...



## Пример со своим ELF

```
git clone https://github.com/SergeyBalabaev/Elective-C-Programming-Language
```

**Elective -> lesson6 -> ELF**

```
chmod +x make.sh  
./make.sh  
./Prog
```



# Пример со своим ELF

```
./make.sh
```

```
./Prog
```

Измените программу так, чтобы она выводила  
ваши имя и фамилию

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL



## Напоминание

1) Память поделена на **страницы**

2) Преобразование:

**Логический адрес -> Линейный адрес -> Физический адрес**



# Напоминание

Компоновщик



Загрузчик

LOAD



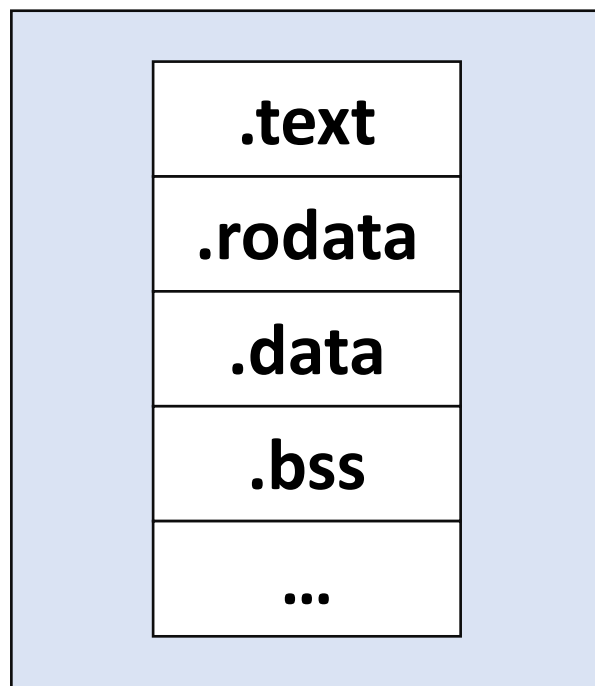


# Напоминание

Секции

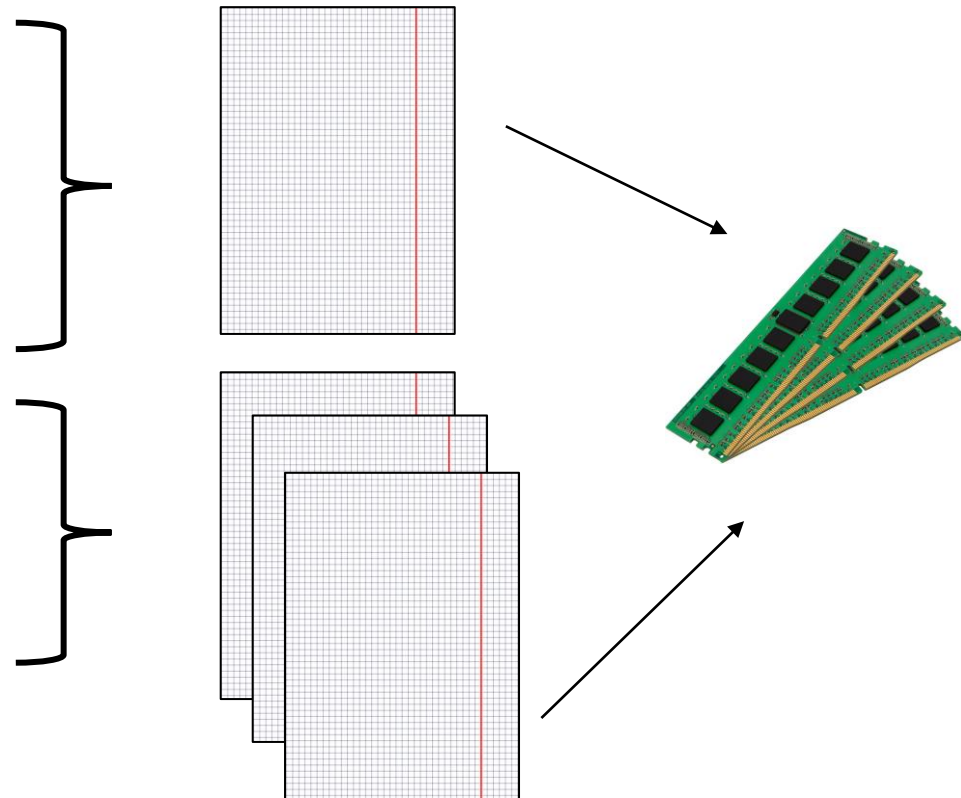
Сегменты

```
int main(void)
{
    ///
    return 0;
}
```



stack

heap





# Секции программы





# 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
```



# Типы данных

Type name	Usual size	Values stored
char	1 byte	integers
short	2 bytes	signed integers
int	4 bytes	signed integers
long	4 or 8 bytes	signed integers
long long	8 bytes	signed integers
float	4 bytes	signed real numbers
double	8 bytes	signed real numbers



## Типы данных

Type	Size (Bits)	Range
bit, _Bit	1	0 , 1
bool, _Bool	8	0 , 1
char	8	-128 to 127
unsigned char	8	0 to 255
signed char	8	-128 to 127
int	16	-32768 to 32767
short int	16	-32768 to 32767
unsigned int	16	0 to 65535
signed int	16	-32768 to 32767
long int	32	-2147483648 to 2147483647
unsigned long int	32	0 to 4294967295
signed long int	32	-2147483648 to 2147483647
float	32	$\pm 1.175e-38$ to $\pm 3.402e38$
double	32	$\pm 1.175e-38$ to $\pm 3.402e38$



# Память





# Память



Char



Int



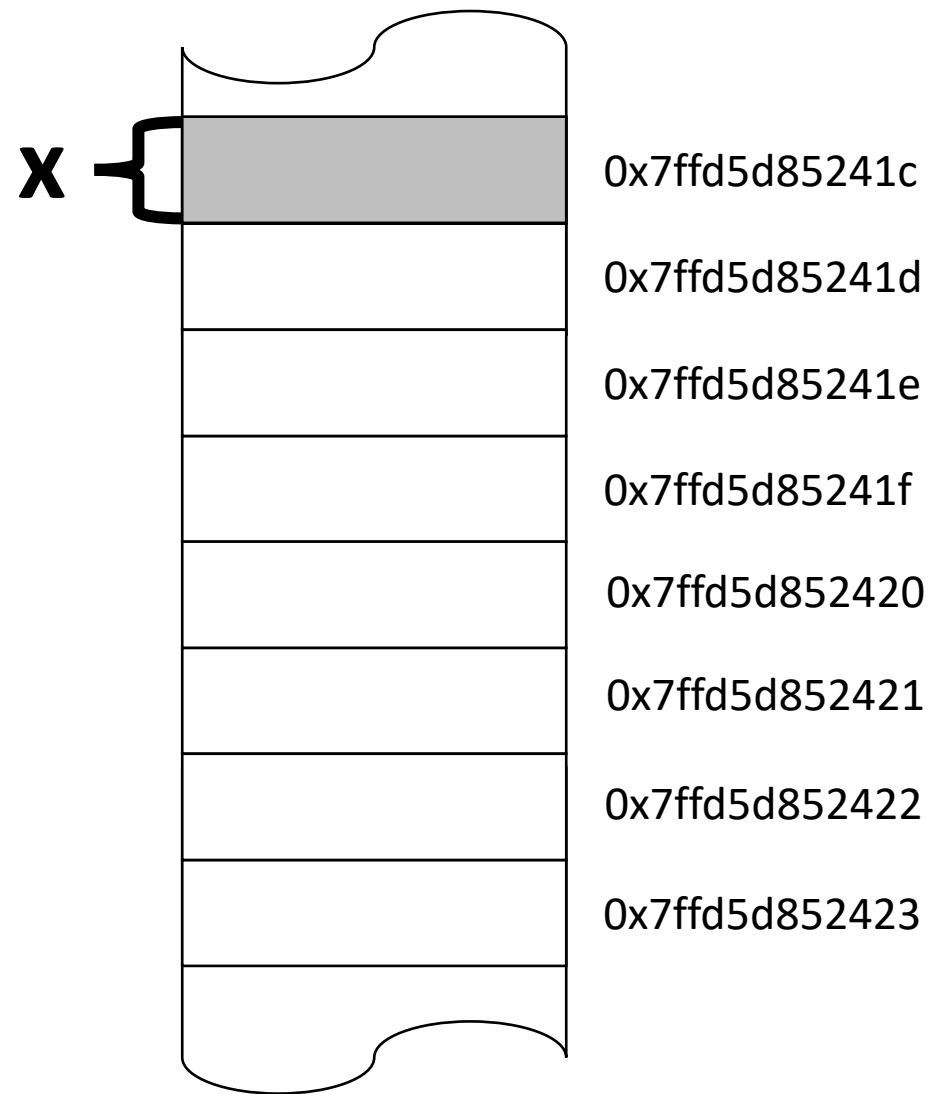
Double





# Адреса

char **x**



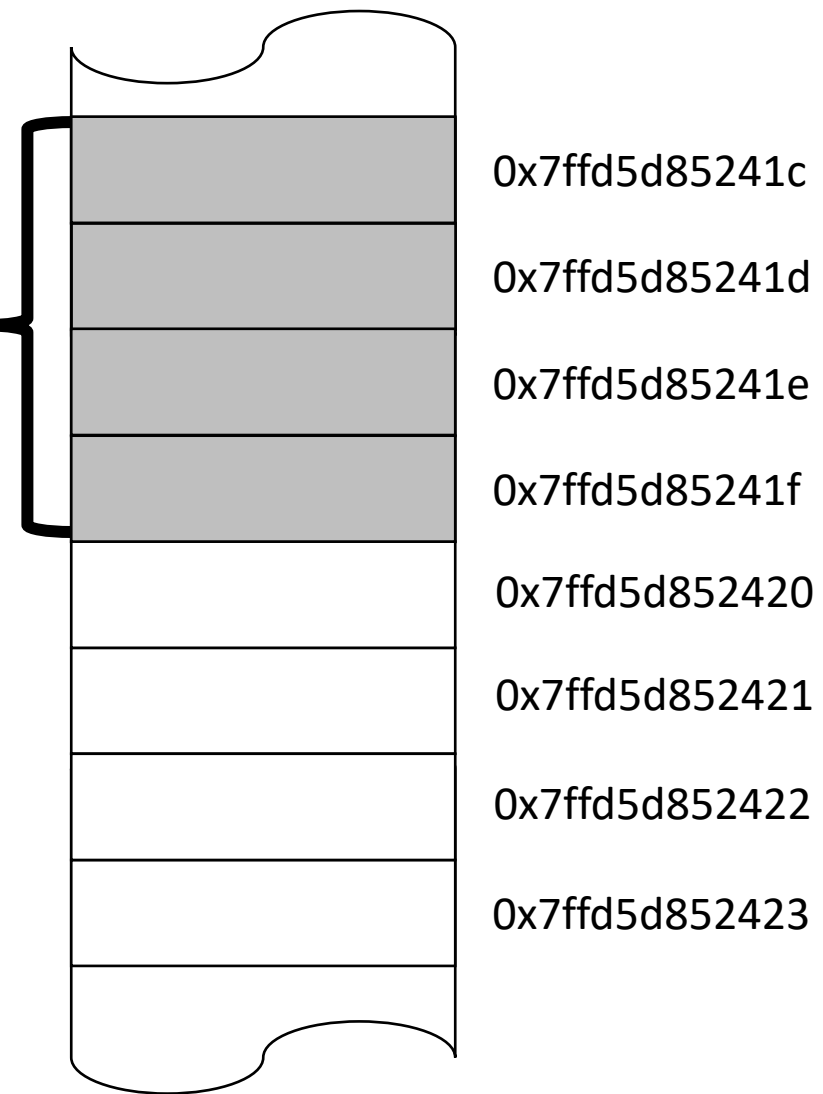


# Адреса

int x



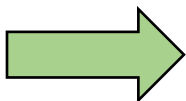
**x**



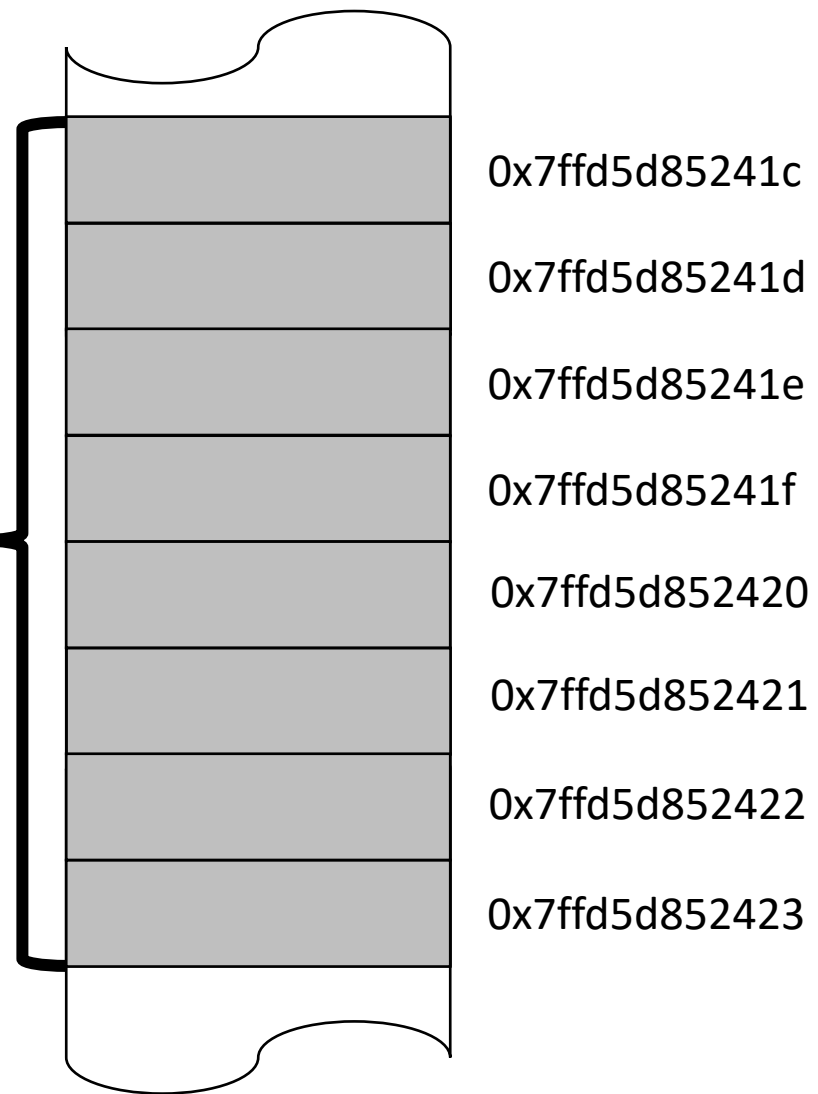


# Адреса

double **x**







**x**



Как хранить адреса?



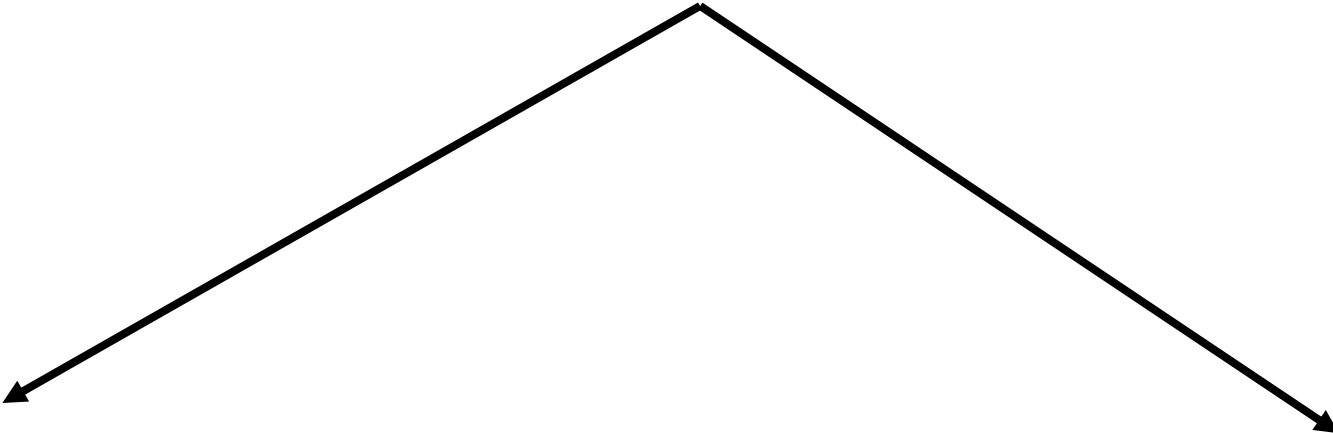
## Указатели

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

Указатель - это адрес переменной в памяти.



# Операции с указателями



Взятие адреса  
 $\&x$

Разыменование  
 $*x$



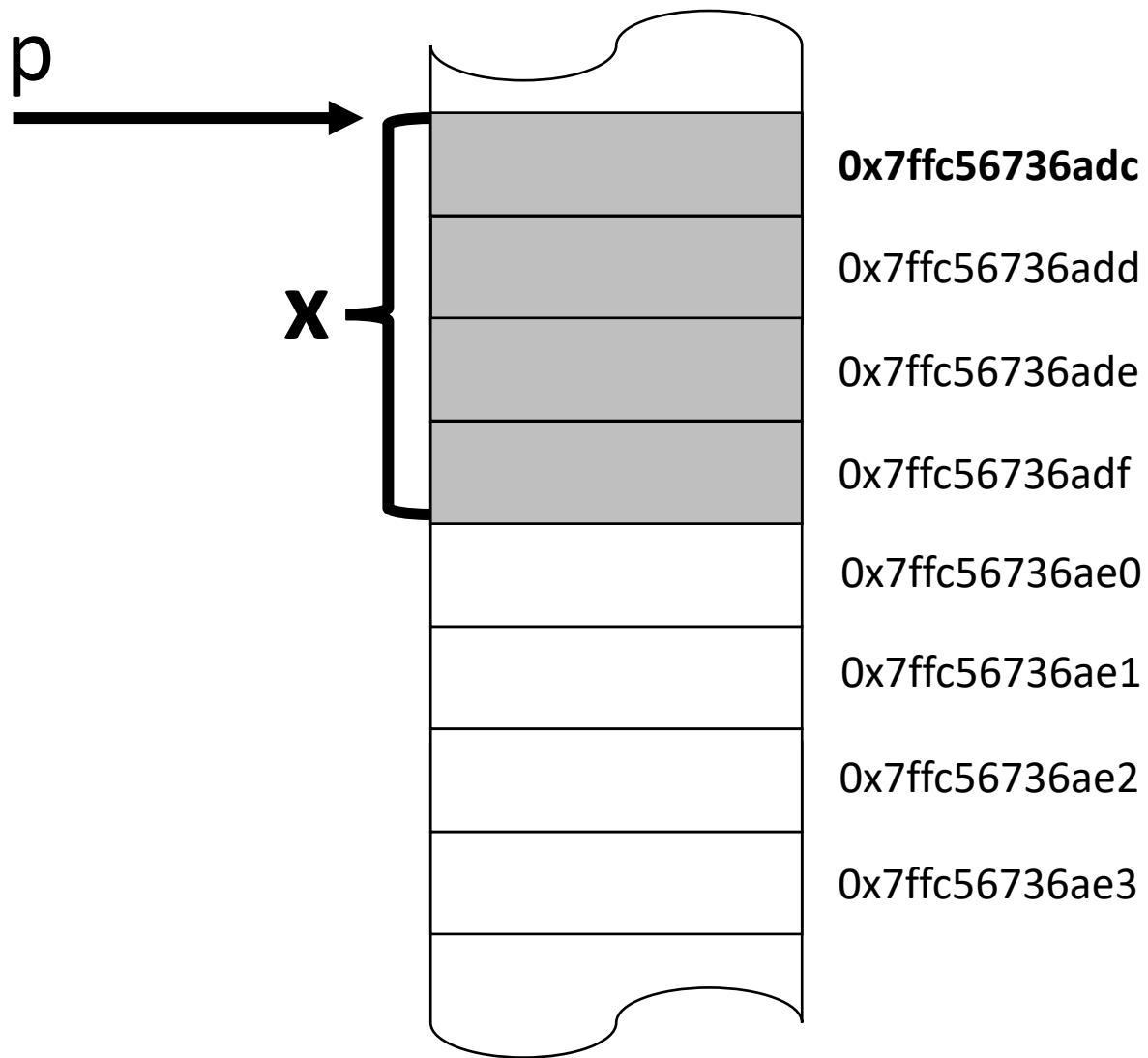
# Указатели

```
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

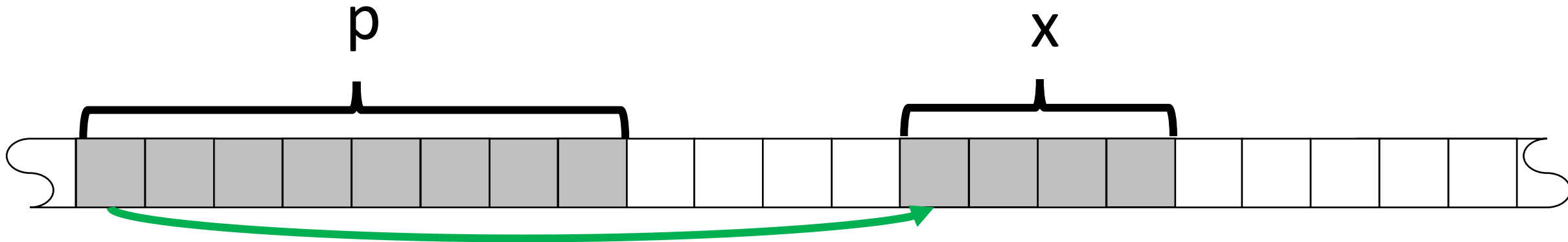


Чему равен размер указателя?



# Указатели

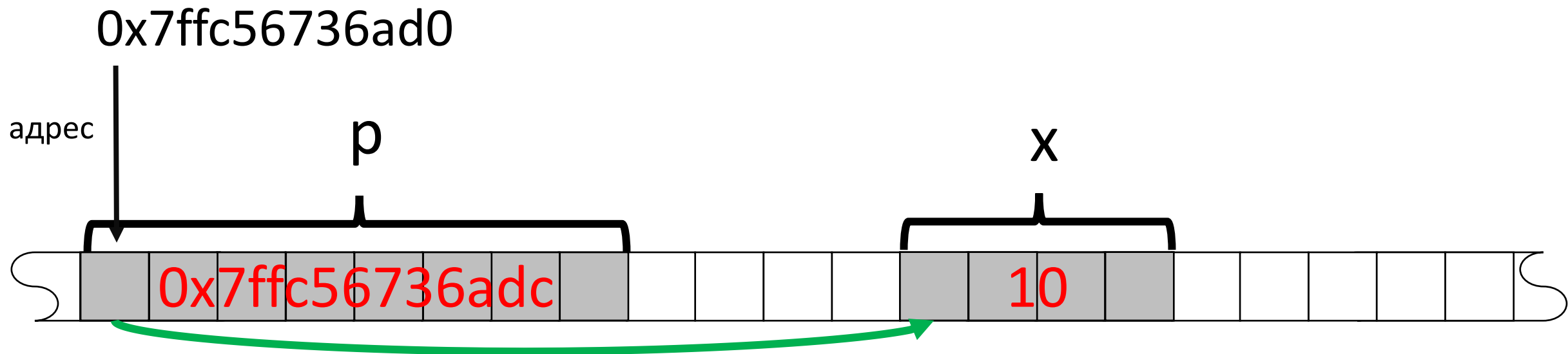
```
int x = 10;  
int* p;  
p = &x;
```





# Указатели

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

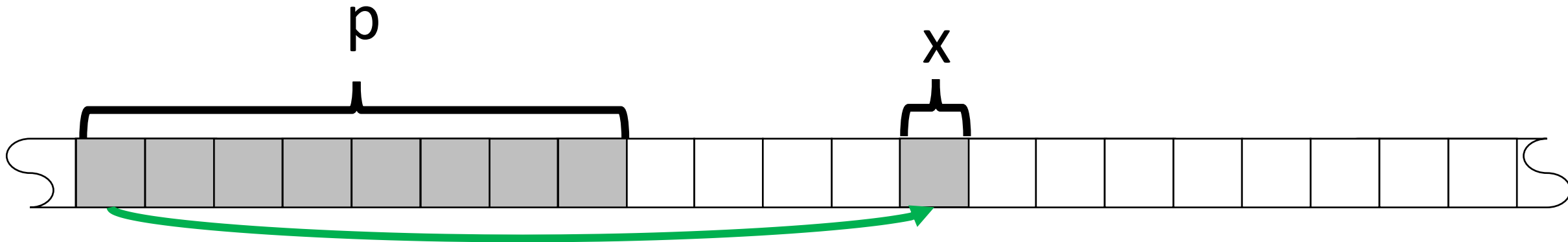






# Указатели

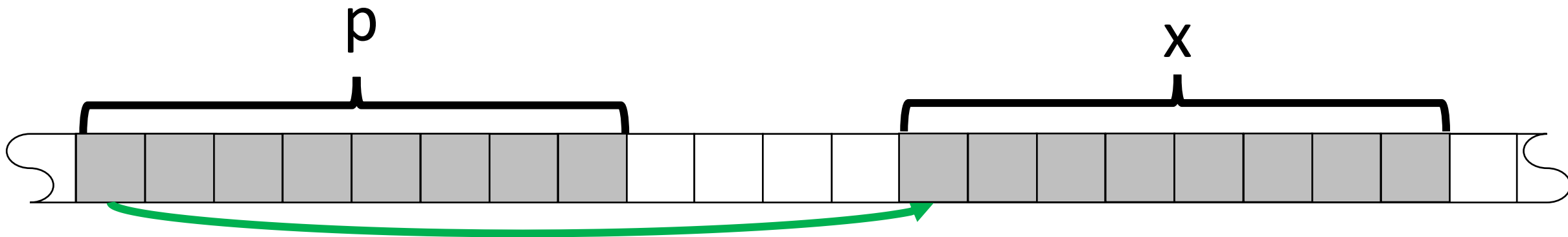
```
char x = 'A';  
char* p;  
p = &x;
```





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

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]`



# Тест

Убедимся в этом!

**Elective -> lesson6 -> Pointers**

```
gcc test.c -o test  
./test
```



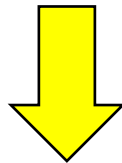
# Указатели и массивы

Заметим!

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



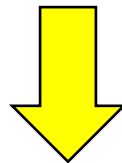
```
ptr = ptr + 1;
```



```
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>
```

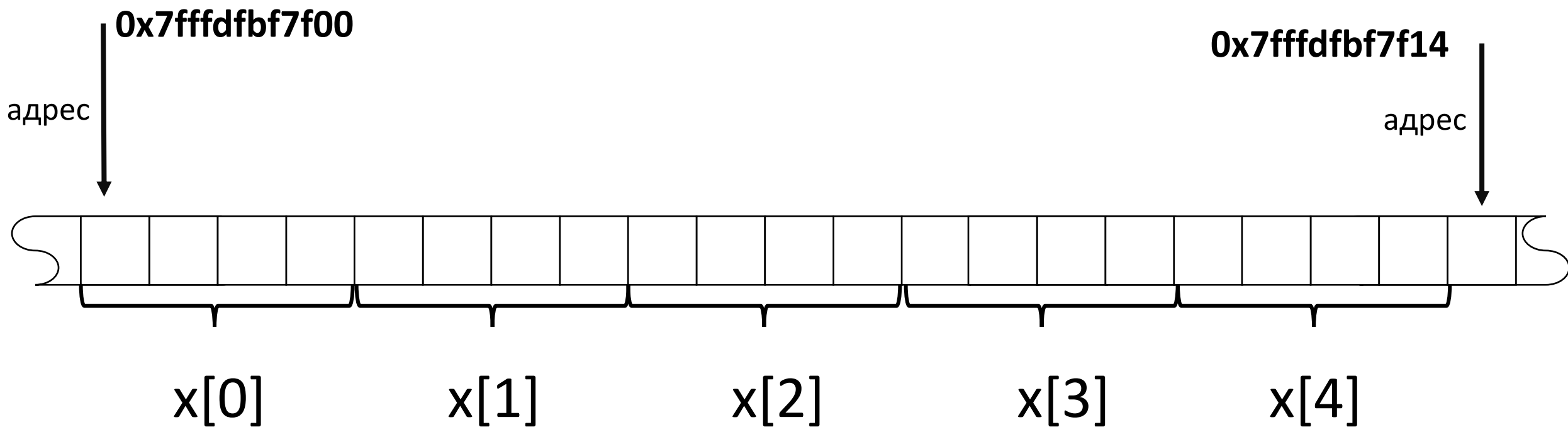
```
int main()
{
    int x[5];
    printf("x=\t%p\n", x);
    printf("x+1=\t%p\n", x + 1);
    printf("&x=\t%p\n", &x);
    printf("&x+1\t%p\n", &x + 1);
    return 0;
}
```

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;
```

```
}
```

```
gcc Challenge.c -o Challenge  
./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;
```

```
}
```

```
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)[5] = &x;
    int(*z)      = &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);
    return 0;
}
```

```
gcc pointerVSarray.c -o pointerVSarray
./pointerVSarray
```



# Указатели и массивы

```
#include <stdio.h>
int main() {
    int x[5] = { 1,2,3,4,5 };
    int(*y)[5] = &x;
    int(*z) = &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);
    return 0;
}
```

```
x:      0x7ffd9f310140
x+1:    0x7ffd9f310144
*y:     0x7ffd9f310140
*y+1:   0x7ffd9f310144
*z:     0x1
*z+1:   0x2
```



# 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
```





## Пример

```
gcc -g gdb_vals.c -o gdb_vals  
gdb gdb_vals
```



## GDB

```
list
break 32
run
info registers
info locals
ptype a
print a
print &a
print sizeof(a)
set var a = 512
print a
```



# GDB

`x/4xb &a`

Вывести

4 байта

в 16 формате

побайтово

с адреса переменной a



x/80xb &b

[illegible]