



СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

ЛЕКЦИЯ № 1

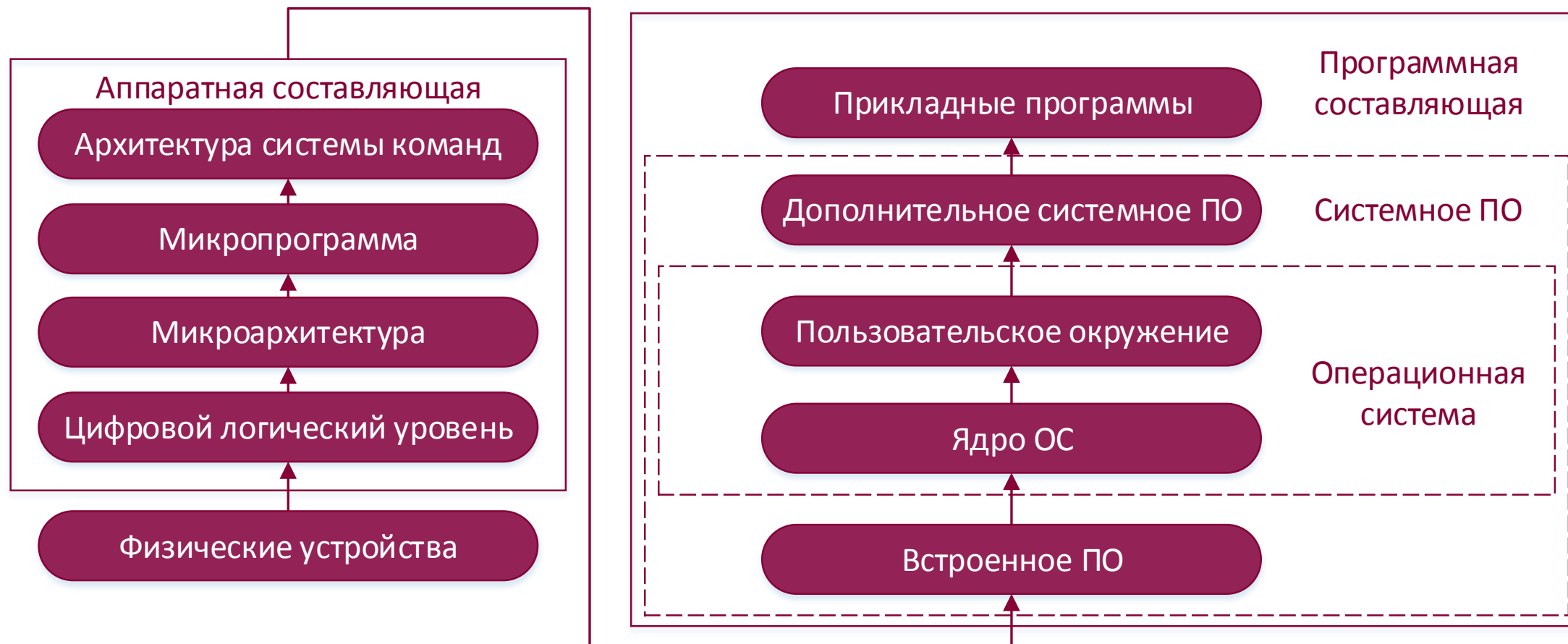
ПРЕПОДАВАТЕЛЬ: ХУСТОЧКА А.В.



ОРГАНИЗАЦИОННЫЙ МОМЕНТ

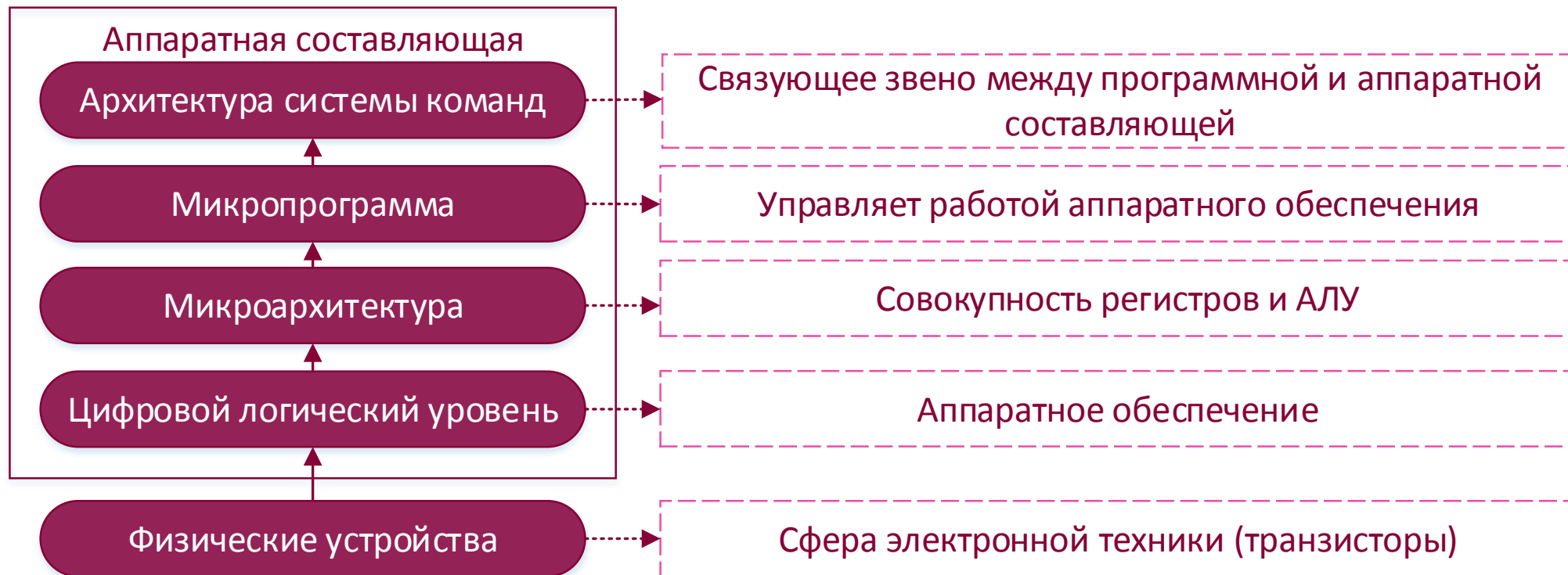
- Контакты: a.khustochka@mgutm.ru (почта) | @khustochka (telegram)
 - Вопросы
 - Лабораторные работы или отчёты по лабораторным
 - Список студентов от старост
- Возможность получения автоматов

ВЫЧИСЛИТЕЛЬНАЯ СИСТЕМА

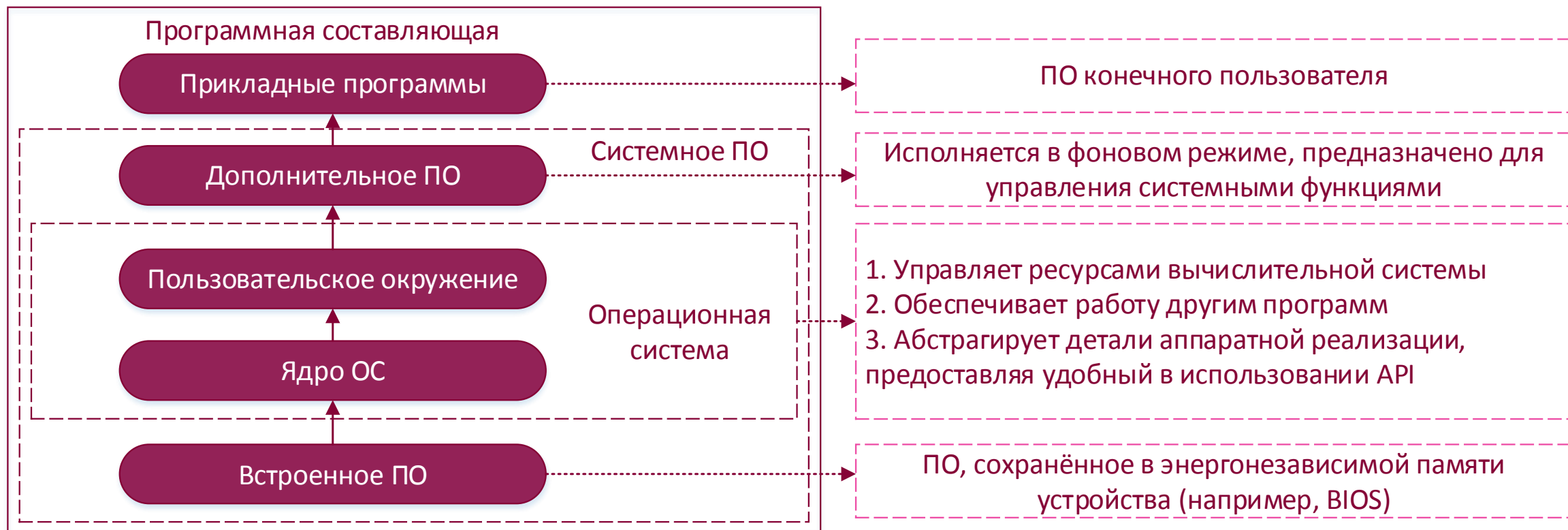


Основная задача любой вычислительной системы – обработка информации.

ФИЗИЧЕСКИЕ УСТРОЙСТВА И АППАРАТНАЯ СОСТАВЛЯЮЩАЯ



ПРОГРАММНАЯ СОСТАВЛЯЮЩАЯ



СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

Цель:

- Управление аппаратными ресурсами вычислительной системы
- Использование **API**, предоставляемого операционной системой, абстрагирующее программистов от аппаратного обеспечения

Наиболее популярные языки:

- Ассемблер
- C/C++

ЯЗЫК C

ТИПЫ ДАННЫХ

Тип данных	Описание	Размер
char	символьный	1 байт (8 бит)
int	целый	От 2 байт (16 бит). На практике обычно 4 байта.
float	вещественный	4 байта (32 бита)
double	вещественный двойной точности	8 байт (64 бита)
void	не имеющий значения	Зависит от системы: 16-ти битная = 2 байта 32-ти битная = 4 байта 64-ти битная = 8 байта

Базовый тип может быть модифицирован с помощью ключевых слов:

- unsigned
- signed
- short
- long

ЯЗЫК С

ПЕРЕПОЛНЕНИЕ ПЕРЕМЕННЫХ

```
#include <stdio.h>

int main()
{
    unsigned int a = 4294967295;
    int b = 2147483647;
    printf("unsigned int a = 4294967295      | %u\n", a);
    a++;
    printf("unsigned int a = 4294967295 + 1  | %u\n", a);
    printf("int          b = 2147483647      | %d\n", b);
    b++;
    printf("int          b = 2147483647 + 1  | %d\n", b);
}
```

unsigned int a = 4294967295		4294967295
unsigned int a = 4294967295 + 1		0
int b = 2147483647		2147483647
int b = 2147483647 + 1		-2147483648

ЯЗЫК С

ОПЕРАТОР ВЕТВЛЕНИЯ И ЛОГИЧЕСКОЕ ОПЕРАЦИИ

Оператор ветвления:

```
if (<условие>) {  
    <действия, которые необходимо выполнить,  
    когда условие выполняется>  
} else {  
    <действия, которые необходимо выполнить,  
    когда условие не выполняется>  
}
```

Логический оператор	Значение
>	Больше
<	Меньше
>=	Больше или равно
<=	Меньше или равно
==	Равно
!=	Не равно
!<условие>	Отрицание выражения

```
int main()  
{  
    unsigned int day = 0;  
    printf("Enter day: ");  
    scanf("%d", &day);  
    if (day > 7) {  
        day = day % 7;  
    }  
    if (day == 0) {  
        printf("Sunday");  
    } else if (day == 1) {  
        printf("Monday");  
    } else if (day == 2) {  
        printf("Tuesday");  
    } else if (day == 3) {  
        printf("Wednesday");  
    } else if (day == 4) {  
        printf("Thursday");  
    } else if (day == 5) {  
        printf("Friday");  
    } else if (day == 6) {  
        printf("Saturday");  
    }  
    return 0;  
}
```

Результат работы
программы:

```
Enter day: 10  
Wednesday
```

ЯЗЫК C

ОПЕРАТОР SWITCH

Оператор switch:

```
switch (<проверяемое значение>){  
    case <значение>:  
        <действия, которые необходимо сделать>;  
        break;  
        ...  
    case <значение>:  
        <действия, которые необходимо сделать>;  
        break;  
    default: break;  
}
```

```
int main()  
{  
    unsigned int day = 0;  
    printf("Enter day: ");  
    scanf("%d", &day);  
    if (day > 7) {  
        day = day % 7;  
    }  
  
    switch (day) {  
        case 0: printf("Sunday");      break;  
        case 1: printf("Monday");     break;  
        case 2: printf("Tuesday");    break;  
        case 3: printf("Wednesday");  break;  
        case 4: printf("Thursday");   break;  
        case 5: printf("Friday");     break;  
        case 6: printf("Saturday");   break;  
        default: printf("Invalid value!"); break;  
    }  
  
    return 0;  
}
```

ЯЗЫК С

ТЕРНАРНЫЙ ОПЕРАТОР

`<переменная> = (<условие>) ?`

`<действия, которые необходимо выполнить, когда условие выполняется> :`

`<действия, которые необходимо выполнить, когда условие не выполняется>;`

Пример:

```
char* result = pin == 1234 ? "PIN-code is valid" : "PIN-code is invalid";
```

ЯЗЫК С

ЦИКЛЫ

```
int main()
{
    int i = 0;
    while (i < 5) {
        printf("i = %d\n", i);
        i++;
    }
    return 0;
}
```

Цикл с предусловием

```
int main()
{
    int i = 0;
    do {
        printf("i = %d\n", i);
        i++;
    } while (i < 5);
    return 0;
}
```

Цикл с постусловием

```
int main()
{
    for (int i = 0; i < 5; i++) {
        printf("i = %d\n", i);
    }
    return 0;
}
```

Цикл с о счётчиком

Результат работы любого цикла:

```
i = 0
i = 1
i = 2
i = 3
i = 4
```

ЯЗЫК С

МАССИВЫ

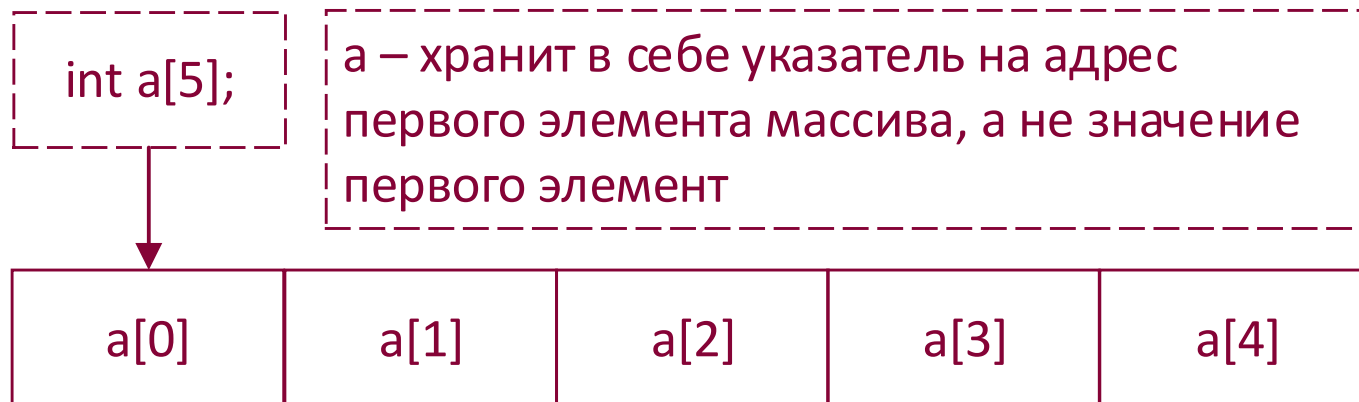
Объявление массива:

<тип> <переменная> [<размер>]

Пример:

```
int a[5] = { 0 };
```

Расположение в памяти:



[illegible]

3


```
00000000ca5ebf718 {0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000}
```

6

7

8

11

12

13

14

15

16

18


```
0x00000000CA5EBF718
```

[illegible]

ConsoleApplication2.cpp

ConsoleApplication2

(Глобальная область)

 main()

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

3

```
4 int main()
```

5

```
6 int a[5] = { 0 };
```

7

8

```
9  for (int i = 0; i < count_of_a; i++) {
```

19

```
11 } ≤ 1 мс прошло
```

12

```
13     for (int i = 0; i < count_of_a; i++) {
```

14

15

16

```
17         return 0;
```

18

Память 1

Адрес: 0x0000000CA5EBF718

0x0000000CA5EBF718	01	00	00	00	02	00	00	00	00	00	00	00	00	00	00	00	00	00	00	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc
0x0000000CA5EBF734	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	05	00	00	00	cc	cc	cc	cc	cc	cc	cc	
0x0000000CA5EBF750	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	01	00	00	00	cc	cc	cc	cc	
0x0000000CA5EBF76C	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	

ConsoleApplication2.cpp

ConsoleApplication2

(Глобальная область)

main()

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

```
3
```

```
4  int main()
```

```
5  {
```

```
6      int a[5] = { 0 };
```

```
7      int count_of_a = sizeof(a) / sizeof(a[0]);
```

```
8
```

```
9      for (int i = 0; i < count_of_a; i++) {
```

```
10         a[i] = i + 1;
```

```
11     }
```

```
12
```

```
13     for (int i = 0; i < count_of_a; i++) {
```

```
14         printf("a[%d] = %d\n", i, a[i]);
```

```
15     }
```

```
16
```

```
17     return 0;
```

```
18 }
```

a | 0x0000000ca5ebf718 {0x00000001, 0x00000002, 0x00000000, 0x00000000, 0x00000000}

≤ 1 мс прошло

Память 1

Адрес: 0x0000000CA5EBF718

0x0000000CA5EBF718	01 00 00 00 02 00 00 00 03 00 00 00 04 00 00 00 05 00 00 00	cc cc
0x0000000CA5EBF734	cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc 05 00 00 00	cc cc
0x0000000CA5EBF750	cc cc	cc cc
0x0000000CA5EBF76C	cc cc	cc cc

ConsoleApplication2.cpp

ConsoleApplication2

(Глобальная область)

main()

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

```
3
```

```
4  int main()
```

```
5  {
```

```
6      int a[5] = { 0 };
```

```
7      int count_of_a = sizeof(a) / sizeof(a[0]);
```

```
8
```

```
9      for (int i = 0; i < count_of_a; i++) {
```

```
10         a[i] = i + 1;
```

```
11     }
```

```
12
```

```
13     for (int i = 0; i < count_of_a; i++) {
```

```
14         printf("a[%d] = %d\n", i, a[i]);
```

```
15     }
```

```
16
```

```
17     return 0;
```

```
18 }
```

a | 0x0000000ca5ebf718 {0x00000001, 0x00000002, 0x00000003, 0x00000004, 0x00000005}

Вывод программы:

a[0]	=	1
a[1]	=	2
a[2]	=	3
a[3]	=	4
a[4]	=	5

ЯЗЫК С

СТРОКИ

Память 1

Адрес: 0x00000077BE3AFCA4

0x00000077BE3AFCA4	6b 6f 74 69 6b 00 cc cc	kotik.MM
0x00000077BE3AFCAC	cc cc cc cc cc cc cc cc	MMMMMMMM
0x00000077BE3AFCB4	cc cc cc cc cc cc cc cc	MMMMMMMM
0x00000077BE3AFCBC	cc cc cc cc cc cc cc cc	MMMMMMMM

ConsoleApplication2.cpp

ConsoleApplication (Глобальная обла main())

```
2 #include <stdio.h>
3
4 int main()
5 {
6     char a[] = "kotik";
7     return 0;
8 }
9
```

а 0x00000077be3afca4 "kotik"

[0x00000000]	0x6b 'k'
[0x00000001]	0x6f 'o'
[0x00000002]	0x74 't'
[0x00000003]	0x69 'i'
[0x00000004]	0x6b 'k'
[0x00000005]	0x00 '\0'

Объявление строки:

`char <переменная> [<размер>]`

Особенности:

- Один элемент в памяти занимает 1 байт
- Символ – это число, которое отображается определённым образом, в зависимости от кодировки

Форматы представления строк:

- Нуль-терминированная строка
- Формат ANSI

ЯЗЫК С УКАЗАТЕЛИ

Объявление указателя:
<тип>* <переменная>

Пример:
`int* pointer = NULL;`

Две основные операции для работы с указателем:

- & – оператор взятия адреса
- * – оператор разыменования

ЯЗЫК С

УКАЗАТЕЛИ

```
int main()
{
    int a = 100;
    int* p = NULL;

    p = &a;
    printf("address: %p\n", p);
    printf("value p      | %d\n", *p);
    printf("value a      | %d\n", a);
    *p = 200;
    printf("new value p | %d\n", *p);
    printf("new value a | %d\n", a);
    a = 300;
    printf("new value p | %d\n", *p);
    printf("new value a | %d\n", a);

    return 0;
}
```

Результат работы программы:

```
address: 000000011C82FAE4
value p      | 100
value a      | 100
new value p  | 200
new value a  | 200
new value p  | 300
new value a  | 300
```

ЯЗЫК С

УКАЗАТЕЛИ

```
int main()
{
    int a = 100;
    int* pointer_a = &a;
    double b = 2.3;
    double* pointer_b = &b;

    printf("size of 'a'          | %d\n", sizeof(a));
    printf("size of 'pointer_a' | %d\n", sizeof(pointer_a));
    printf("size of 'b'          | %d\n", sizeof(b));
    printf("size of 'pointer_b' | %d\n", sizeof(pointer_b));

    return 0;
}
```

size of 'a'	4	size of 'a'	4
size of 'pointer_a'	4	size of 'pointer_a'	8
size of 'b'	8	size of 'b'	8
size of 'pointer_b'	4	size of 'pointer_b'	8

x86

x64

Важные моменты:

- Размер указателя зависит от платформы
- Размер одного указателя любого типа всегда совпадает размеру указателя любого другого типа

Вопрос:

- Зачем указателю нужен тип?

ЯЗЫК С

АРИФМЕТИКА УКАЗАТЕЛЕЙ

```
int main()
{
    int a[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    int* int_pointer = a;
    double* double_pointer = (double*)a;

    printf(" # | a[] | int_pointer | double_pointer (as int) | double_pointer (as double)\n");
    printf("-----+-----+-----+-----+-----\n");
    for (int i = 0; i < 10; i++) {
        if (i < 5) {
            printf(" %d | %d | %d | %d | %lf\n",
                i, a[i], *int_pointer, *double_pointer, *double_pointer);
        } else {
            printf(" %d | %d | %d | %d | %lf\n",
                i, a[i], *int_pointer, *double_pointer, *double_pointer);
        }
        int_pointer++;
        double_pointer++;
    }
    printf("-----+-----+-----+-----+-----\n");

    return 0;
}
```

ЯЗЫК С

АРИФМЕТИКА УКАЗАТЕЛЕЙ

Результат работы программы с предыдущего слайда:

#	a[]	int_pointer	double_pointer (as int)	double_pointer (as double)
0	0	0	0	0.000000
1	1	1	2	0.000000
2	2	2	4	0.000000
3	3	3	6	0.000000
4	4	4	8	0.000000
5	5	5	-858993460	-92559631349317830736831783200707727132248687965119994463780864.000000
6	6	6	-858993460	-92559631349317830736831783200707727132248687965119994463780864.000000
7	7	7	-858993460	-92559631349317830736831783200707727132248687965119994463780864.000000
8	8	8	1873606808	0.000000
9	9	9	-858993460	-92559631349317830736831783200707727132248687965119994463780864.000000

ЯЗЫК С АРИФМЕТИКА УКАЗАТЕЛЕЙ

Арифметика указателей нужна:

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

Указатели поддерживают следующие операции:

- Сложение
- Вычитание
- Сравнение

ЯЗЫК С ДВОЙНОЙ УКАЗАТЕЛЬ

Объявление двойного указателя:

`<тип>** <переменная>`

Пример:

```
int a = 1;
```

```
int* pointer = &a;
```

```
int** pointer_1 = &pointer;
```

Нужен для работы с массивами разной размерности.

ЯЗЫК С

КЛЮЧЕВОЕ СЛОВО – CONST

`const` – ключевое слово, указывающее, что данный объект менять нельзя.

Типы неизменяемых объектов:

- Константный указатель
- Указатель-константа
- Указатель на константу

КОНСТАНТНЫЙ УКАЗАТЕЛЬ

```
int main()
{
    const char* const_p = NULL;
    char* p = NULL;
    unsigned length = 0;
    char str[] = "meow :3";

    p = str;
    const_p = str;
    while (*const_p++) {
        length++;
    }
    printf("length (\"%s\") = %d\n", str, length);

    *p = '!';
    *const_p = '1';

    return 0;
}
```

```
length ("meow :3") = 7
pointer      : "meow :3"
const pointer : "meow :3"
pointer      : "cat says 'meow'"
const pointer : "cat says 'meow'"
```

```
int main()
{
    const char* const_p = NULL;
    char* p = NULL;
    unsigned length = 0;
    char str[] = "meow :3";
    char new_str[] = "cat says 'meow'";

    p = str;
    const_p = str;
    while (*const_p++) {
        length++;
    }
    const_p = str;
    printf("length (\"%s\") = %d\n", str, length);
    printf("pointer      : \"%s\"\n", p);
    printf("const pointer : \"%s\"\n", const_p);

    p = new_str;
    const_p = new_str;
    printf("pointer      : \"%s\"\n", p);
    printf("const pointer : \"%s\"\n", const_p);

    return 0;
}
```

Код Описание ▾

E0137 выражение должно быть допустимым для изменения левосторонним значением

C3892 const_p: невозможно присваивать значения переменной, которая объявлена как константа

ЯЗЫК С

УКАЗАТЕЛЬ КОНСТАНТА

Результат работы программы:

```
x      = 10
p      = 10
x new  = 20
p new  = 20
```

Константный указатель на константу:

```
int main()
{
    int x = 10;
    int* const p = &x;

    printf("x      = %d\n", x);
    printf("p      = %d\n", *p);
    *p = 20;
    printf("x new  = %d\n", x);
    printf("p new  = %d\n", *p);
    //p = &x;
    //p++;

    return 0;
}
```

```
int main()
{
    int x = 10;
    const int* const p = &x;

    printf("x      = %d\n", x);
    printf("p      = %d\n", *p);
    *p = 20;
    p = &x;
    p++;
}
```

const int *const p

выражение должно быть допустимым для изменения левосторонним значением

ЯЗЫК С

УКАЗАТЕЛЬ НА КОНСТАНТУ

Программа, приводящая к ошибке компилятора:

```
int main()
{
    const int x = 10;
    const int* p = &x;

    printf("x = %d\n", x);
    printf("p = %d\n", *p);

    return 0;
}
```

Результат работы программы:

```
x = 10
p = 10
```

```
int main()
{
    const int x = 10;
    const int* p = &x;
    int* p_1 = &x;

    printf("x = ");
    printf("p = ");

    x = 15;
    *p = &x;

    return 0;
}
```

значение типа "const int *" нельзя использовать для инициализации сущности типа "int *"

ЯЗЫК С

УКАЗАТЕЛИ И ОДНОМЕРНЫЕ МАССИВЫ

```
int main()
{
    int a[5] = { 1, 2, 3, 4, 5 };
    int* p = a;

    printf("a[3]      = %d\n", a[3]);
    printf("*(a + 3) = %d\n", *(a + 3));
    printf("*(p + 3) = %d\n", *(p + 3));

    return 0;
}
```

Отработает ли следующий код и почему:

```
int a[5] = { 1, 2, 3, 4, 5 };
printf("%d\n", a[3]);
printf("%d\n", 3[a]);
```

Результат работы программы:

```
a[3]      = 4
*(a + 3) = 4
*(p + 3) = 4
```

ЯЗЫК C

МАКРОСЫ

```
#define PRINT_MAX(x, y) { \
    if (x > y) { \
        printf("max number is: %d", x); \
    } else { \
        printf("max number is: %d", y); \
    } \
}

int main()
{
    PRINT_MAX(5, 7);
    return 0;
}
```

```
#define SOME_TEXT "print number:"
#define INT_FORMAT "%d"

int main()
{
    printf(SOME_TEXT " " INT_FORMAT, 10);
    return 0;
}
```

```
max number is: 7
```

ЯЗЫК С

СТРУКТУРЫ И ПЕРЕЧИСЛЯЕМЫЙ ТИП

Объявление структуры:

```
struct <имя> {  
    <тип 1> <поле 1>;  
    ...  
    <тип N> <поле N>;  
};
```

Объявление перечисляемого типа:

```
enum <имя> {  
    <элемент 1>,  
    ...  
    <элемент N>,  
};
```

```
typedef enum PONY_TYPE {  
    LITTLE_PONY,  
    BIG_PONY  
};  
  
typedef struct {  
    int age;  
    const char* name;  
    PONY_TYPE type;  
} MY_PONY;  
  
int main()  
{  
    MY_PONY ignat;  
    ignat.age = 20;  
    ignat.type = LITTLE_PONY;  
    ignat.name = "Ignat";  
  
    printf("I have a pony.\nHis name is %s.\nHi is %d years old.\nHe is a %s.",  
        ignat.name, ignat.age, ignat.type == LITTLE_PONY ? "little pony" : "big pony");  
  
    return 0;  
}
```

Вывод программы:

```
I have a pony.  
His name is Ignat.  
Hi is 20 years old.  
He is a little pony.
```