

Структурное программирование

ЛЕКЦИЯ №3

19 СЕНТЯБРЯ 2023



Передача массивов в функции

Часто, размер массива передается отдельным параметром:

```
int arr_sum(int arr[], int size)
{
    int sum = 0;
    for (int i = 0; i < size; i++)
    {
        sum += arr[i];
    }
    return sum;
}
```

Передача фиксированных массивов в функции

Массивы жестко фиксированных размерностей можно передать без дополнительных параметров

```
int matrix_sum(int matrix[3][3])
{
    int sum = 0;
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            sum += matrix[i][j];
        }
    }
    return sum;
}
```

Возвращение массивов из функций

Попробуем по аналогии с тем, как передавали массив

Получим ошибки ☹️

```
int[] reverse(int src[], int size)
{
    int dst[size];
    for (int i = 0; i < size; i++)
    {
        dst[i] = src[size - i - 1];
    }
    return dst;
}

int original[3] = { 1, 2, 3 };
int reversed[] = reverse(original, 3);
```

“Возвращение” массивов из функций

Правильный вариант:

Массив==указатель на его начало.

Передаем указатель=>

Работаем с оригиналом.

```
void reverse(int src[], int dst[], int size)
{
    for (int i = 0; i < size; i++)
    {
        dst[i] = src[size - i - 1];
    }
    return;
}
```

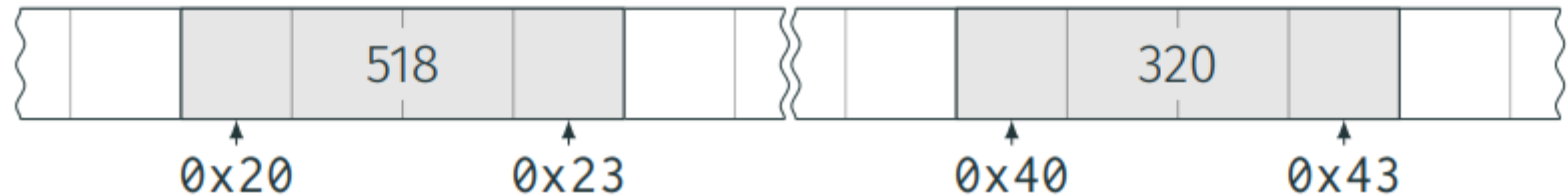
```
int original[3] = { 1, 2, 3 };
int reversed[3];
reverse(original, reversed, 3);
```

Указатель

Указатель – переменная, содержащая адрес объекта

- **Не** несет информации о содержимом объекта
- Содержит сведения о том, **где** размещен объект (его начало)
- `int* pA; float* pB;`
- Указатель – это тоже переменная, которая сидит в памяти

Указатель как адрес

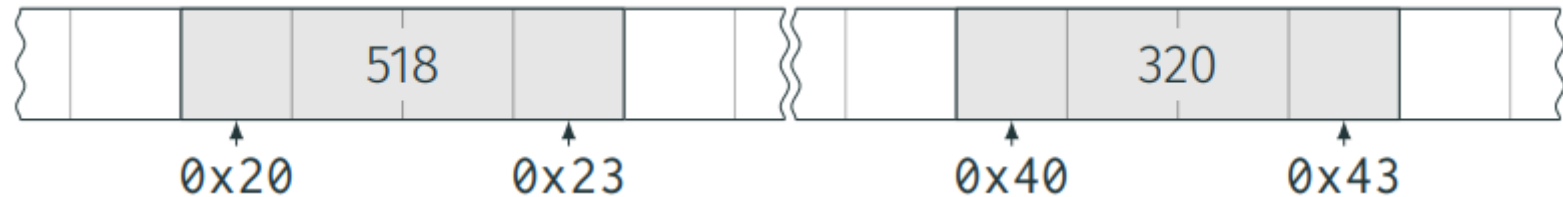


& - взятие адреса

* - разыменование указателя

```
int x = 518;      ptr = &x;
int y = 320;      printf("%p %d\n", ptr, *ptr);
int* ptr;         ptr = &y;
                  printf("%p %d\n", ptr, *ptr);
```

Указатель как адрес



& - взятие адреса

* - разыменование указателя

```
int x = 518;
```

```
int y = 320;
```

```
int* ptr;
```

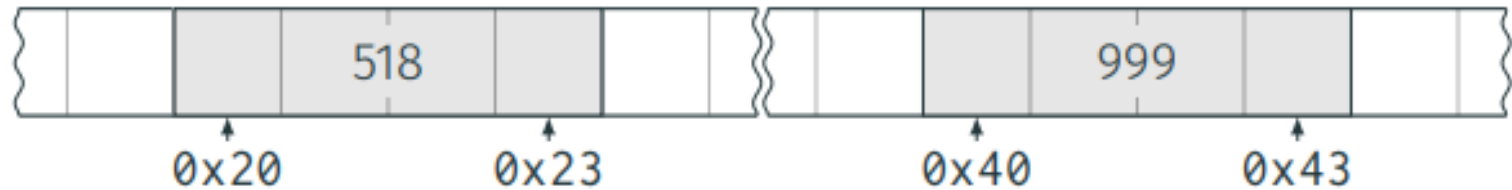
```
ptr = &x;
```

```
printf("%p %d\n", ptr, *ptr); //20 518
```

```
ptr = &y;
```

```
printf("%p %d\n", ptr, *ptr); //40 320
```


Указатель как адрес



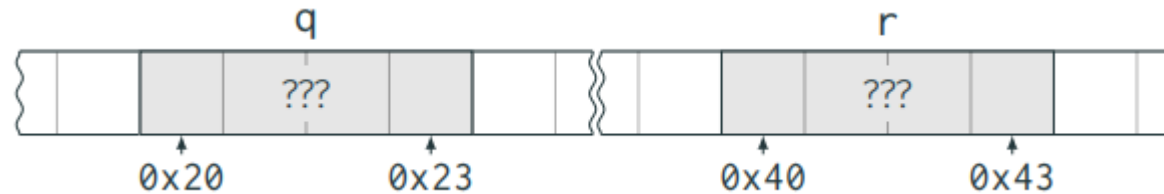
& - взятие адреса

* - разыменование указателя

```
int x = 518;
int y = 320;
int* ptr;

ptr = &x;
printf("%p %d\n", ptr, *ptr); //20 518
ptr = &y;
printf("%p %d\n", ptr, *ptr); //40 320
*ptr = 999;
printf("%d %d\n", x, y); //518 999
```

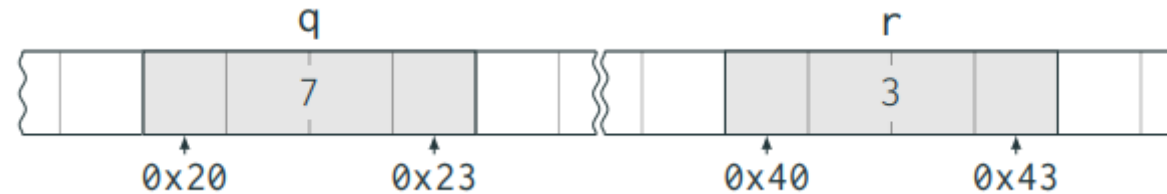
Возврат двух значений из функции



```
void divide(int a, int b, int* pq, int* pr)
{
    *pq = a / b;
    *pr = a % b;
    return;
}

int x = 73, y = 10;
int q, r;
divide(x, y, &q, &r);
```

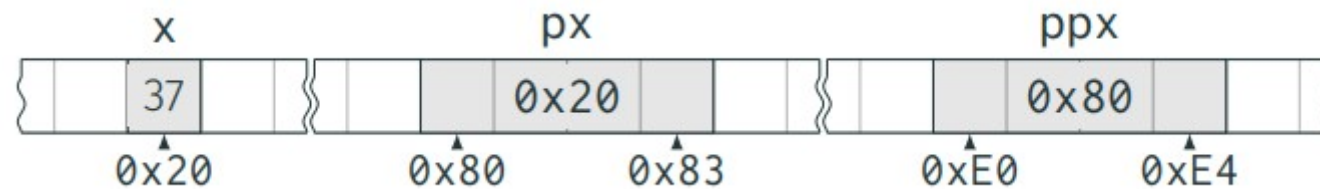
Возврат двух значений из функции



```
void divide(int a, int b, int* pq, int* pr)
{
    *pq = a / b; // *(0x20) = 7
    *pr = a % b; // *(0x30) = 3
    return;
}
```

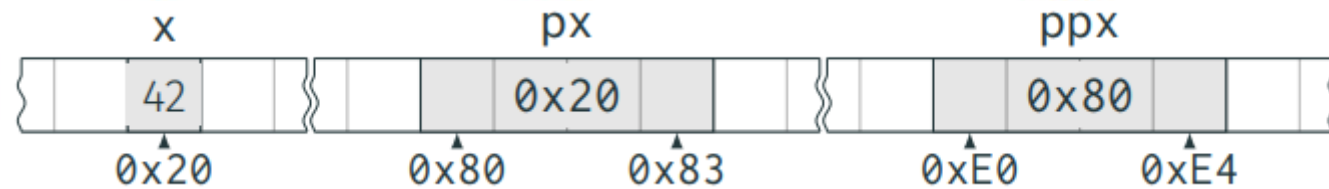
```
int x = 73, y = 10;
int q, r;
divide(x, y, &q, &r);
```

Двойной указатель



```
char x = 37;  
char* px = &x;  
char** ppx = &px;
```

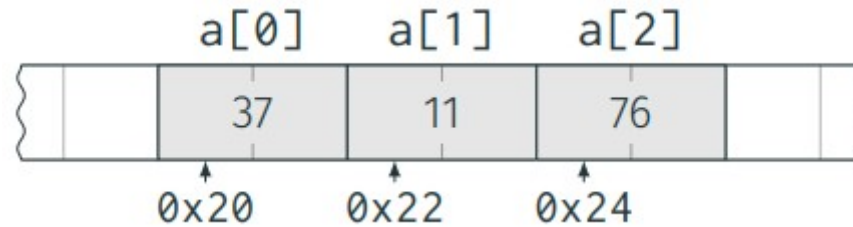
Двойной указатель



```
char x = 37;  
char* px = &x;  
char** ppx = &px;
```

```
**ppx = 42;
```

Массивы и указатели



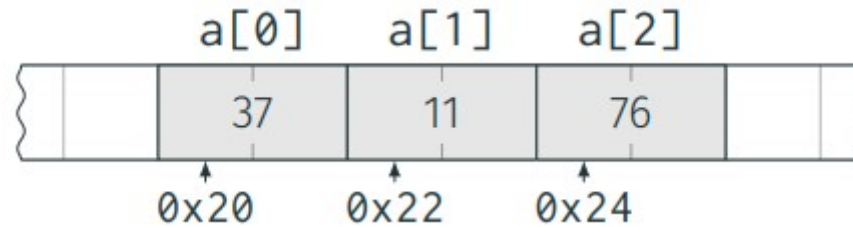
```
short a[3] = { 37, 11, 76 };
```

```
short* p = &(a[0]); ?
```

```
short* p1 = p + 1; ?
```

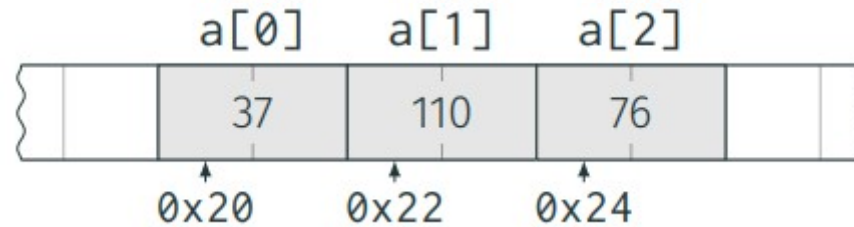
```
short* p2 = p + 2; ?
```

Массивы и указатели



```
short a[3] = { 37, 11, 76 };  
short* p = &(a[0]); // 0x20  
short* p1 = p + 1; // 0x22  
short* p2 = p + 2; // 0x24
```

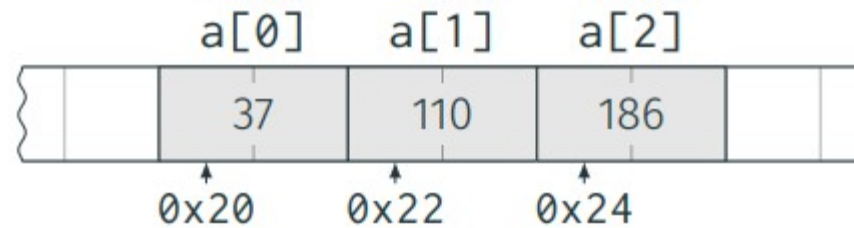
Массивы и указатели



```
short a[3] = { 37, 11, 76 };  
short* p = &(a[0]); // 0x20  
short* p1 = p + 1; // 0x22  
short* p2 = p + 2; // 0x24
```

```
*p1 = 110;
```

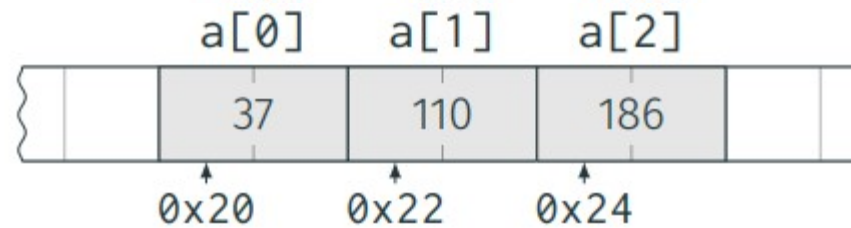

Массивы и указатели



```
short a[3] = { 37, 11, 76 };  
short* p = &(a[0]); // 0x20  
short* p1 = p + 1; // 0x22  
short* p2 = p + 2; // 0x24
```

```
*p1 = 110;  
*p2 = *p1 + *p2;
```

Массивы и указатели

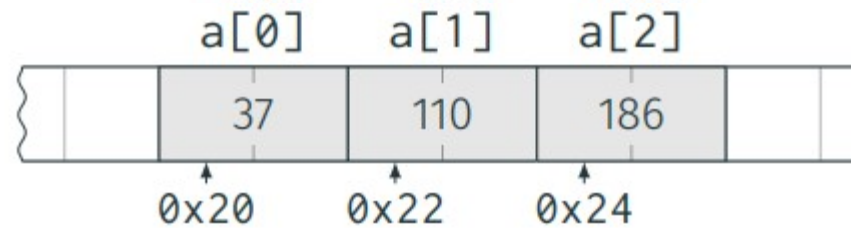


```
short a[3] = { 37, 11, 76 };  
short* p = &(a[0]); // 0x20  
short* p1 = p + 1; // 0x22  
short* p2 = p + 2; // 0x24
```

```
*p1 = 110;  
*(p+2) = *(p+1) + *(p+2);
```

```
*(x + y) == x[y]
```

Массивы и указатели

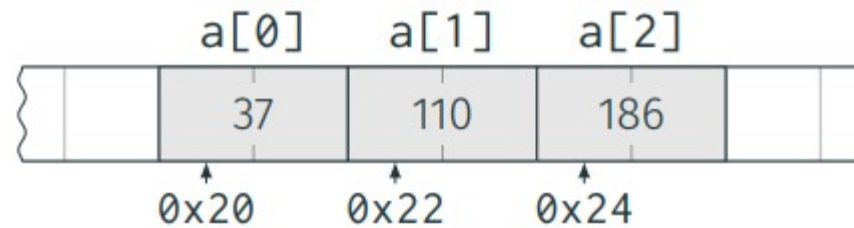


```
short a[3] = { 37, 11, 76 };  
short* p = &(a[0]); // 0x20  
short* p1 = p + 1; // 0x22  
short* p2 = p + 2; // 0x24
```

```
p[1] = 110;  
p[2] = p[1] + p[2];
```

$*(x + y) == x[y]$

Массивы и указатели



```
short a[3] = { 37, 11, 76 };
```

```
short* p = a; //0x20
```

```
short* p1 = p + 1; //0x22
```

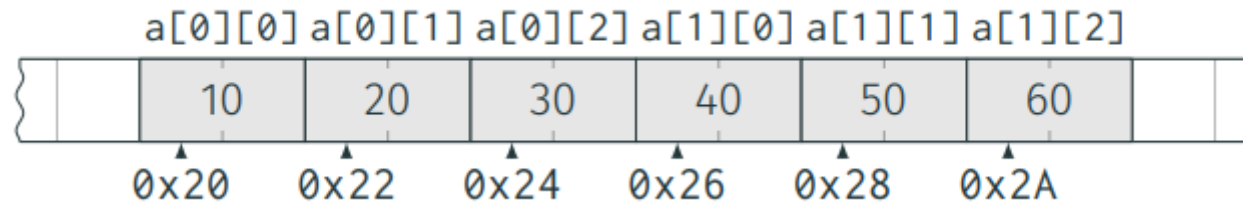
```
short* p2 = p + 2; //0x24
```

```
p[1] = 110;
```

```
p[2] = p[1] + p[2];
```

```
*(x + y) == x[y]
```

Двумерные массивы и указатели



```
short a[2][3] = { {10, 20, 30}, {40, 50, 60} };  
a[i][j] == *((short*)a + i * 3 + j);
```

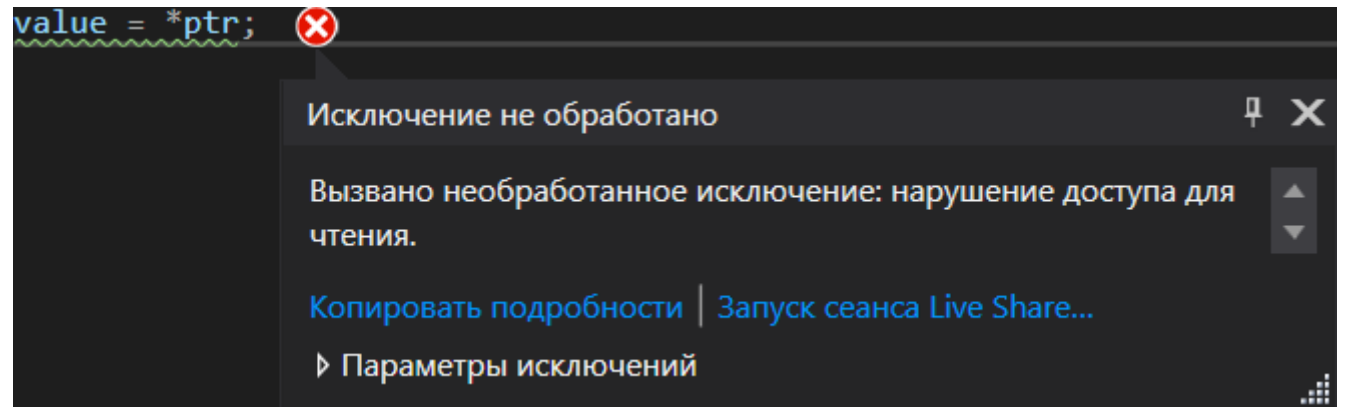
```
short* p = a;  
a[i][j] == p[i * 3 + j];
```

Указатель никуда

NULL – специальное константное значение, символизирующее, что указатель не указывает ни на какую память. (stdlib.h)

При обращении по такому указателю возникает «неопределенное поведение», чаще всего заканчивается фатальной ошибкой (segmentation fault, segfault, access violation, “ программа выполнила недопустимую операцию...»). Но может быть и хуже.

```
int* ptr = NULL;  
int value = *ptr; //беда  
*ptr = 37; //беда
```



Указатель хоть куда

`void*` - специальный тип указателя, который может указывать на любые данные в памяти. Может быть приведен к любому другому типу указателей и обратно.

```
double x = 37.5;  
double* px = &x;  
void* p = px;  
int* py = p;
```

Работа с динамической памятью

Для работы с динамической памятью используется библиотека `stdlib.h`

➤ `malloc`

➤ `calloc`

➤ `realloc`

➤ `free`

Выделение памяти

```
void* malloc(size_t size);
```

Функция выделяет блок памяти размером size байт и возвращает указатель на начало блока.

Если не получилось – NULL

```
void* calloc(size_t num, size_t size);
```

Функция выделяет блок памяти размером num*size байт, зануляет его и возвращает указатель на начало.

Если не получилось – NULL.

Освобождение блока памяти

```
void free(void* ptr);
```

Функция освобождает блок памяти, если `ptr == NULL`, ничего не делает.

После вызова значение указателя `ptr` остается прежним, но разыменовывать его нельзя. Повторно освободить тоже нельзя!

Небольшой пример

```
int n = 5;
short* p;
p = malloc(n * sizeof(short));
if (p == NULL)
    return 0;
p[0] = 0;
p[n - 1] = 1;
p[n / 2] = 2;
printf("%d %d %d", p[0], p[n/2], p[n-1]);
free(p);
```

0 2 1

C:\Users\vmbov\sou
Нажмите любую клавишу

Тонкость с инициализацией строк

test – указатель на константную строку.

```
char* test = "cat";  
test[2] = 'r';
```

test – массив, проинициализированный константной строкой.

```
char test[] = "cat"; // =  
{ 'c', 'a', 't', '\0' }  
test[2] = 'r';  
printf("%s\n", test); //car
```

#define

С помощью #define можно сделать все...

```
main.c [x]
(Global Scope)
#include <stdio.h>
#define kek "lol"
#define числоко int
#define главненькая_функция main
#define начало {
#define конец }
#define печатай printf
#define конец_строки ;

числоко главненькая_функция()
начало
    печатай("%s\n", kek) конец_строки
конец

C:\Windows\system32\cmd.exe
lol
Для продолжения нажмите любую клавишу . . .
```

Кусок файла math.h

```
#define M_E 2.71828182845904523536
#define M_LOG2E 1.44269504088896340736
#define M_LOG10E 0.434294481903251827651
#define M_LN2 0.693147180559945309417
#define M_LN10 2.30258509299404568402
#define M_PI 3.14159265358979323846
#define M_PI_2 1.57079632679489661923
#define M_PI_4 0.785398163397448309616
#define M_1_PI 0.318309886183790671538
#define M_2_PI 0.636619772367581343076
#define M_2_SQRTPI 1.12837916709551257390
#define M_SQRT2 1.41421356237309504880
#define M_SQRT1_2 0.707106781186547524401
```



Пересечение имен

А что если сделать так?

Какое значение выведет программа?

```
#include <stdio.h>

int a = 3;

int main()
{
    int a = 4;
    for (int i = 0; i < 1; i++)
    {
        int a = 5;
        printf("%d\n", a);
    }
    printf("%d\n", a);
    return 0;
}
```

```
#include <stdio.h>

int a = 3;

int main()
{
    int a = 4;
    printf("%d\n", a);
    return 0;
}
```



Пересечение имен

А что если сделать так?

Какое значение выведет программа?

```
#include <stdio.h>

int a = 3;

int main()
{
    int a = 4;
    for (int i = 0; i < 1; i++)
    {
        int a = 5;
        printf("%d\n", a);
    }
    printf("%d\n", a);
    return 0;
}
```

```
#include <stdio.h>

int a = 3;

int main()
{
    int a = 4;
    printf("%d\n", a);
    return 0;
}
```



```
C:\Windows\system32\cmd.exe
5
4
Для продолжения нажмите любую клавишу . . .
```

Локальная переменная важнее чем какая-то там глобальная!

1) Переменные живут от фигурной скобки до парной закрывающей скобки

2) Переменная «заходит» во все внутренние блоки кода, если внутри там нет «местной» переменной

Прототип функции

Как написать функцию?

Есть 2 способа варианта:

```
#include <stdio.h>

int summ(int a, int b);

int main()
{
    int a = 4;
    int b = 5;
    printf("%d\n", summ(a, b));
}

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

Иначе получим странные ошибки

- ⚠ 1 warning C4013: 'summ' undefined; assuming extern returning int
- ✖ 2 error C2371: 'summ' : redefinition; different basic types

```
#include <stdio.h>

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

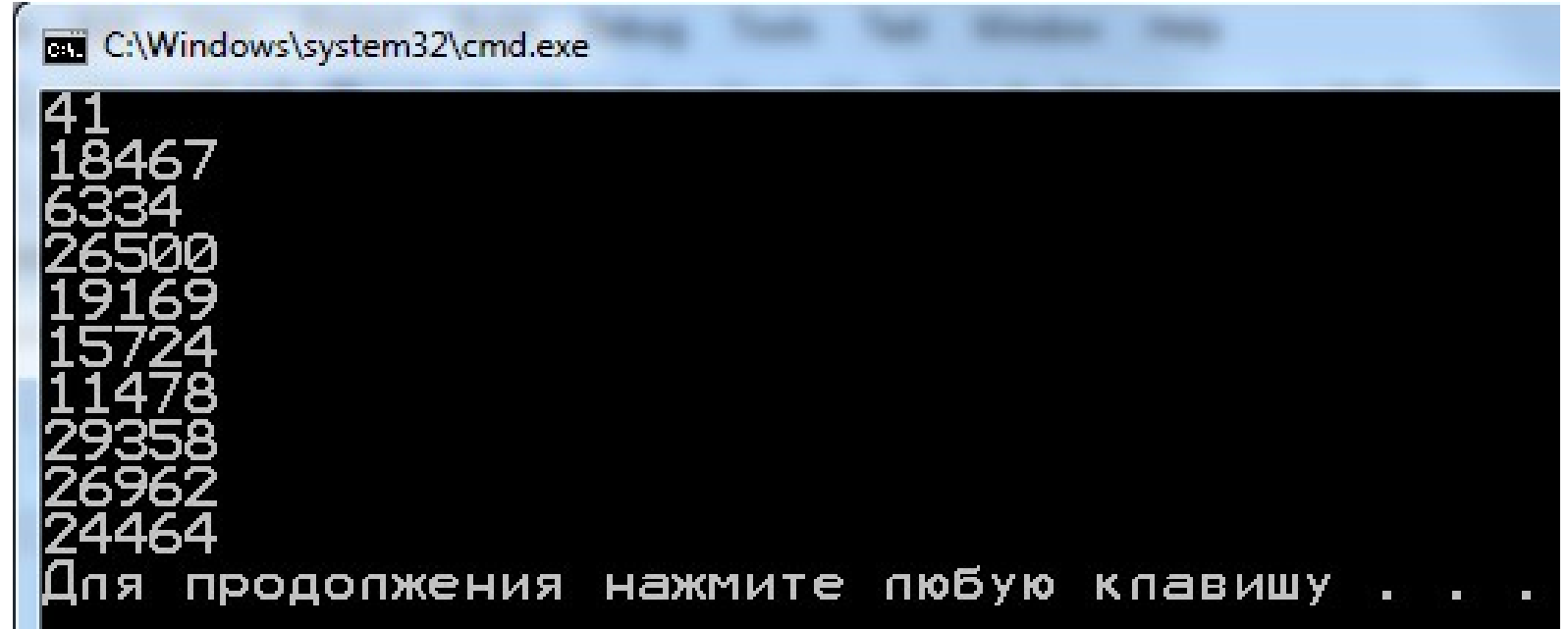
int main()
{
    int a = 4;
    int b = 5;
    printf("%d\n", summ(a, b));
}
```


«Случайные» числа в Си

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

void main()
{
    int i, r;

    for (i = 0; i < 10; i++)
    {
        r = rand();
        printf("%d\n", r);
    }
}
```

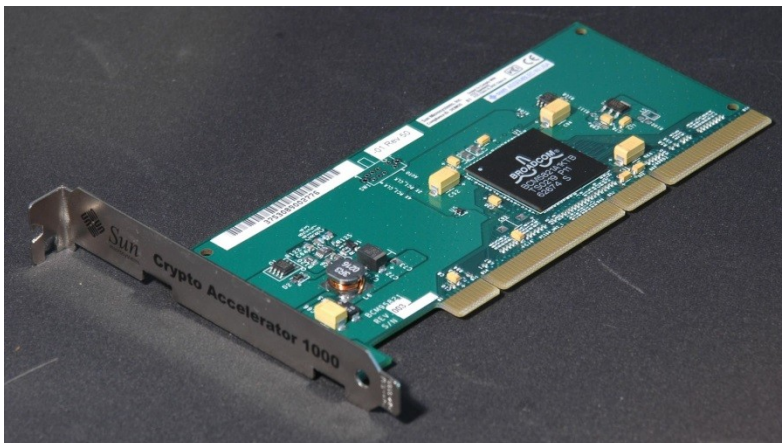


```
cmd.exe C:\Windows\system32\cmd.exe
41
18467
6334
26500
19169
15724
11478
29358
26962
24464
Для продолжения нажмите любую клавишу . . .
```

```
#include <stdlib.h>
int rand( void );
```

Функция RAND() генерирует положительное целое число от 0 до RAND_MAX

Случайные числа



Использование надёжных источников энтропии, таких, как тепловой шум, дробовой шум, фотоэлектрический эффект, квантовые явления и т. д.

- Если в качестве источника энтропии использовать текущее время, то для получения целого числа от 0 до N достаточно вычислить остаток от деления текущего времени в миллисекундах на число N+1.
- Недостатком этого ГСЧ является то, что в течение одной миллисекунды он выдает одно и то же число.

Псевдослучайные числа

«особенные алгоритмы»

Никакой детерминированный алгоритм не может генерировать полностью случайные числа, он может только аппроксимировать некоторые их свойства.

Инициализация генератора «случайных» чисел в Си

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

```
void srand( unsigned int seed );
```

Функция `srand` выполняет инициализацию генератора случайных чисел `rand`. Генератор псевдо-случайных чисел инициализируется с помощью аргумента `seed`.

```
#include <time.h>
```

```
Тип данных time_t
```

Этот тип данных используется для представления целого числа — количества секунд, прошедших после полуночи 00:00 , 1 января 1970 года в формате GMT. Это обусловлено историческими причинами, связанными со становлением платформы UNIX.

```
#include <time.h>
```

```
time_t time( time_t * timeptr );
```

```
time_t time( );
```

Функция возвращает текущее календарное значение времени в секундах. Если аргумент не является нулевым указателем, ей передается значение времени [типа time_t](#).

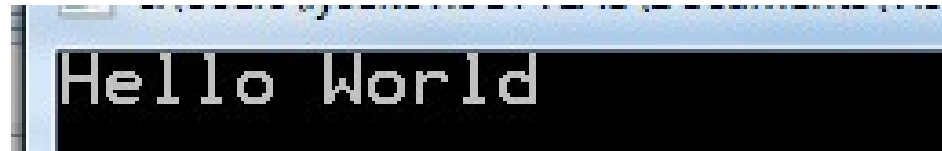
Строки в Си

```
printf("Hello World");
```

0	1	2	3	4	5	6	7	8	9	10	11	12
H	e	l	l	o		w	o	r	l	d		

*Строка в Си – одномерный массив символов,
но с особенностями*

```
void main()  
{  
    int i;  
    char str[] = "Hello World";  
  
    for(i = 0; i < 11; i++)  
        printf("%c", str[i]);  
}
```



Строки в Си

```
void main()
```

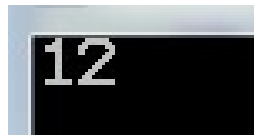
```
{
```

```
    char str[] = "Hello World";
```

```
    printf("%d", sizeof(str));
```

```
}
```

0	1	2	3	4	5	6	7	8	9	10	11	12
H	e	l	l	o		w	o	r	l	d		



12

нуль-символ '\0'

при объявлении строковой константы
нуль-символ добавляется автоматически

0	1	2	3	4	5	6	7	8	9	10	11	12
H	e	l	l	o		w	o	r	l	d	\0	

```
void main()
```

```
{
```

```
    char str[] = "Hello World";
```

```
    printf("%s", str);
```

```
}
```

Автоматический контроль
окончания строки:
вывод происходит до символа '\0'

Ввод и вывод строк в Си

() Строка в языке Си – одномерный массив символов, заканчивающийся '\0'*

%s – спецификатор работы со строками при вводе и выводе в языке Си

```
void main()
{
    char str[80];
    scanf("%s", str);
    printf("%s", str);
}
```

Контроль размера массива
количество элементов всегда должно быть больше
количества хранящихся символов

Присвоение значений строкам в Си

```
char str[] = "Hello World";
```

- Автоматическое вычисление размера массива
- Автоматическое добавление '\0'

```
char str[12] = "Hello World";
```

- Автоматическое добавление '\0' (*)
- **отсутствие контроля корректности размера**

```
void main()  
{  
    char str[10] = "Hello World";  
    printf("%s", str);  
}  
  
void main()  
{  
    char str[3];  
    str[0] = 'H';  
    str[1] = 'i';  
    str[2] = '\0';  
    printf("%s", str);  
}
```

- **Ручной контроль**



Hello WorlHHHHHH2Y ▶№: _



Запись чисел с фиксированной точкой

Есть фиксированное количество бит для целой и дробной частей

00000001, 01010001
целая часть дробная часть

Запись чисел с фиксированной точкой

- Больше точность – меньше диапазон.
- Больше диапазон – меньше точность.
- Округление при представлении чисел (отбрасывание дробной части), но предсказуемо
- Умножение и деление могут привести к потери точности, но предсказуемо.
- Вычисления быстры, как вычисления с целыми числами.

BCD с фиксированной точкой

Кодирование в BCD с четырьмя знаками после запятой:

Метод используется для хранения денежных величин\в калькуляторах (точное представление десятичных дробей).

Числа с плавающей точкой

m – мантисса (значащая часть)

b – основание степени (обычно 2 или 10)

e – экспонента (порядок).



Числа с плавающей точкой

m – мантисса (значащая часть)

b – основание степени (обычно 2 или 10)

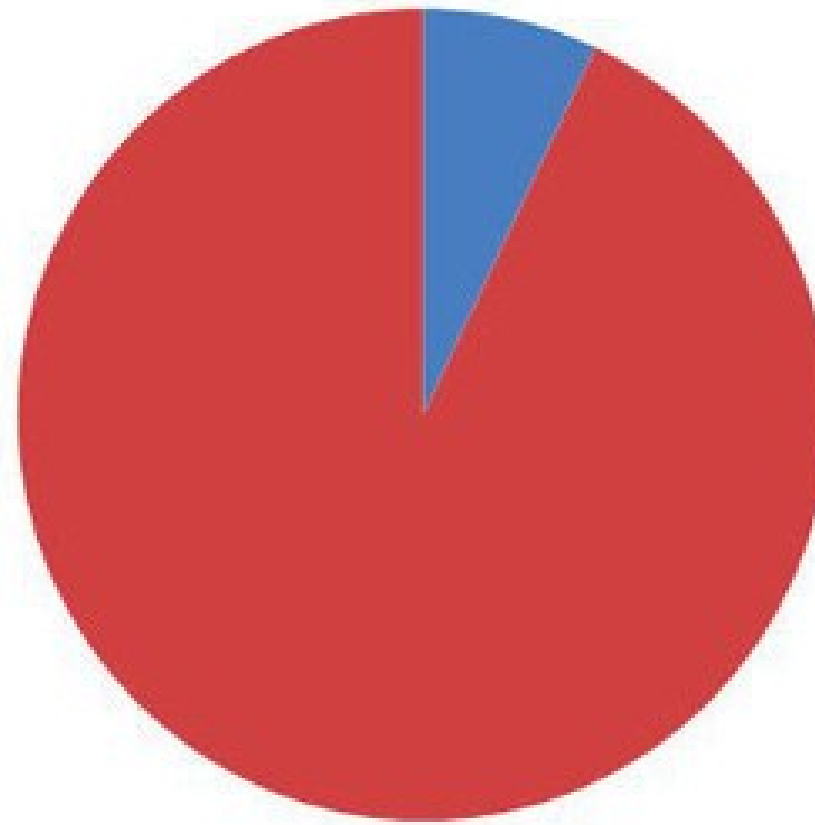
e – экспонента (порядок).

3.1415



Вроде все

**ЕСТЬ ЛИ У ТЕБЯ ШАНСЫ УМЕСТИТЬ
В ПАМЯТИ ВСЁ, ЧТО ТЕБЕ НУЖНО?**



■ НЕТ

■ ТОЖЕ НЕТ, НО СИНЕГО ЦВЕТА