

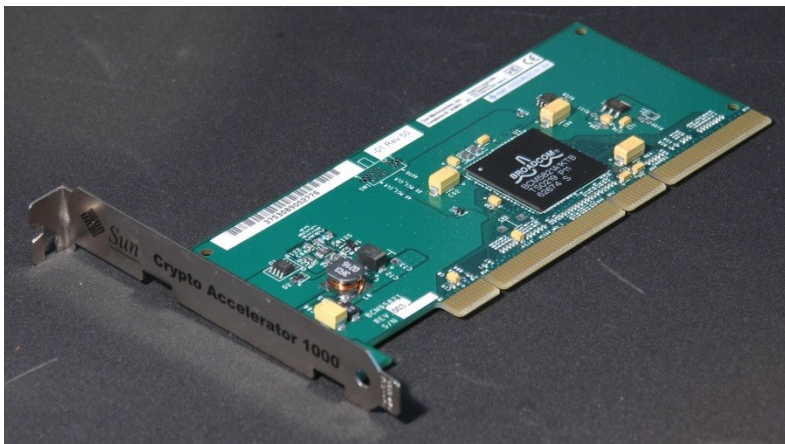
Структурное программирование

ЛЕКЦИЯ №5

3 ОКТЯБРЯ 2023



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](#).

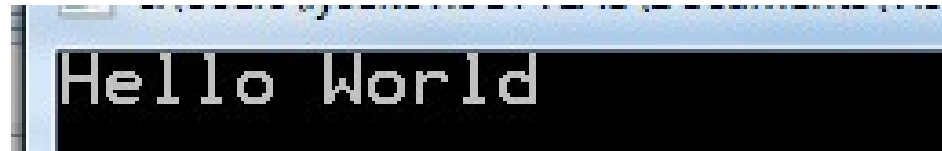
Строки в Си

```
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()
```

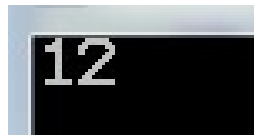
```
{
```

```
    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 ▶№: _

Утечки памяти

Если память регулярно выделяется, но не освобождается, то рано или поздно память кончится, «утечет».

Главное правило хорошего программиста на С: закончил работать с блоком памяти – освободи его!





ЛОВИМ утечки в VS

```
#define _CRTDBG_MAP_ALLOC
#include <stdlib.h>
#include <crtDBG.h>

int main()
{
    int a = 5;
    int* p;
    for (int i = 0; i < 5; i++)
    {
        p = malloc(5);
    }
    _CrtDumpMemoryLeaks();
}
```

```
Detected memory leaks!
Dumping objects ->
C:\Users\vmbror\source\repos\Project4\Project4\Source.c(11) : {80} normal block at 0x009889B0, 5 bytes long.
Data: <    > CD CD CD CD CD
C:\Users\vmbror\source\repos\Project4\Project4\Source.c(11) : {79} normal block at 0x00984B90, 5 bytes long.
Data: <    > CD CD CD CD CD
C:\Users\vmbror\source\repos\Project4\Project4\Source.c(11) : {78} normal block at 0x00984B58, 5 bytes long.
Data: <    > CD CD CD CD CD
C:\Users\vmbror\source\repos\Project4\Project4\Source.c(11) : {77} normal block at 0x00985030, 5 bytes long.
Data: <    > CD CD CD CD CD
C:\Users\vmbror\source\repos\Project4\Project4\Source.c(11) : {76} normal block at 0x00984FF8, 5 bytes long.
Data: <    > CD CD CD CD CD
Object dump complete.
"Project4.exe" (Win32). Загружено "C:\Windows\SysWOW64\kernel.appcore.dll".
```

Список ошибок Вывод Результаты поиска символа

Фрагментация памяти



Все хорошо, все свободно

Фрагментация памяти



Все хорошо, все занято

Фрагментация памяти



Высокая фрагментация: 5 байт из 11 свободно, но выделить блок размером 2 байта не получится.

Изменение размера блока памяти

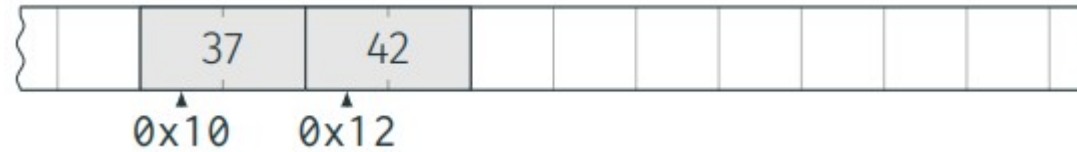
```
void* realloc(void* prt, size_t size);
```

Функция изменяет размер блока памяти до size байт. В случае успешного изменения размера возвращает указатель на начало блока, иначе NULL.

Функция может как уменьшать размер, так и увеличивать. Возможно перемещение содержимого памяти, при этом возвращается указатель на новое месторасположение.

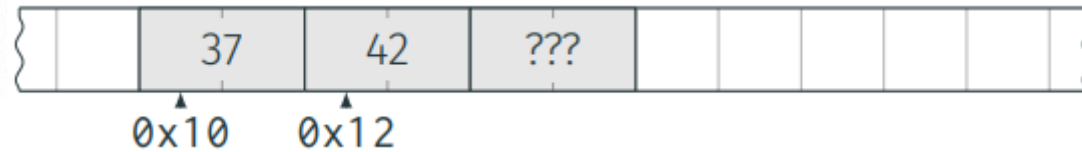
В случае неуспешного изменения размера, изначальный блок памяти не освобождается.

Пример работы realloc



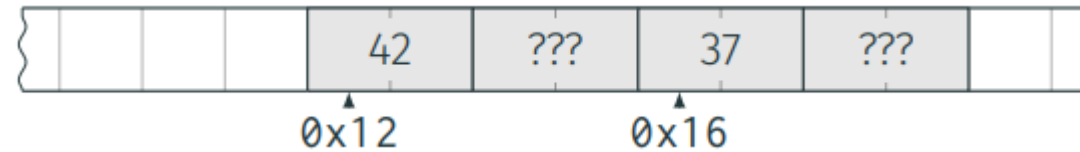
```
short* x = malloc(sizeof(short));  
short* y = malloc(sizeof(short));  
if (x == NULL || y == NULL) { return 0; }  
*x = 37;  
*y = 42;
```

Пример работы realloc



```
short* x = malloc(sizeof(short));
short* y = malloc(sizeof(short));
if (x == NULL || y == NULL) { return 0; }
*x = 37;
*y = 42;
short* y2 = realloc(y, 2 * sizeof(short));
```

Пример работы realloc



```
short* x = malloc(sizeof(short));
short* y = malloc(sizeof(short));
if (x == NULL || y == NULL) { return 0; }
*x = 37;
*y = 42;
short* y2 = realloc(y, 2 * sizeof(short));
short* x2 = realloc(x, 2 * sizeof(short));
if (x2 == NULL || y2 == NULL) { return 0; }
```


Изменяем размер правильно

```
char* x;  
x = malloc(2000);  
if (x == NULL) { return 0; }  
use(x);  
  
x = realloc(x, 4000);  
if(x==NULL){/* а больше нет нашего x */ }
```

Изменяем размер правильно

```
char* x;  
x = malloc(2000);  
if (x == NULL) { return 0; }  
use(x);
```

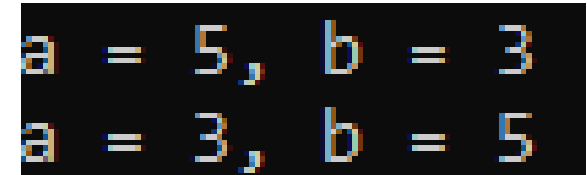
```
char* x2 = realloc(x, 4000);  
if (x2 == NULL) { free(x); return 0; }  
x = x2;  
use(x);
```



Используем указатели

```
void swap(void* a, void* b, size_t size) {
    char* tmp;
    tmp = (char*)malloc(size);
    memcpy(tmp, a, size);
    memcpy(a, b, size);
    memcpy(b, tmp, size);
    free(tmp);
}

int main()
{
    int a = 5, b = 3;
    printf("a = %d, b = %d\n", a, b);
    swap(&a, &b, sizeof(int));
    printf("a = %d, b = %d\n", a, b);
}
```

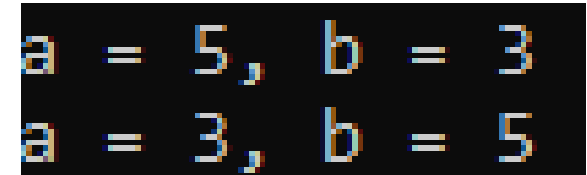


```
a = 5, b = 3
a = 3, b = 5
```

Используем указатели

```
void swap(void* a, void* b, size_t size) {
    char* tmp;
    tmp = (char*)malloc(size);
    memcpy(tmp, a, size);
    memcpy(a, b, size);
    memcpy(b, tmp, size);
    free(tmp);
}

int main()
{
    int a = 5, b = 3;
    printf("a = %d, b = %d\n", a, b);
    swap(&a, &b, sizeof(a));
    printf("a = %d, b = %d\n", a, b);
}
```



```
a = 5, b = 3
a = 3, b = 5
```

Используем указатели

```
void swap(void* a, void* b, size_t size) {
    char tmp;
    size_t i;
    for (i = 0; i < size; i++) {
        tmp = *((char*)b + i);
        *((char*)b + i) = *((char*)a + i);
        *((char*)a + i) = tmp;
    }
}

int main()
{
    int a = 5, b = 3;
    printf("a = %d, b = %d\n", a, b);
    swap(&a, &b, sizeof(a));
    printf("a = %d, b = %d\n", a, b);
}
```

```
a = 5, b = 3
a = 3, b = 5
```





Функции как данные

В языке C с функциями можно работать как с данными:

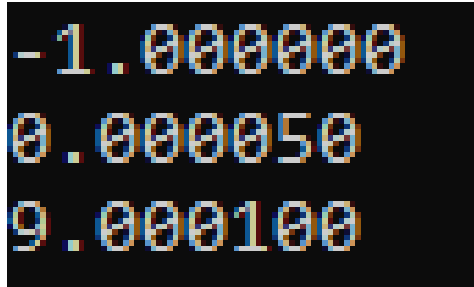
```
double (*my_func)(double, double) = pow;  
double x = my_func(3, 2);
```

Функция – набор байтов в памяти, кодирующих тело этой функции с помощью машинных команд. Значит, можно просто взять адрес этого «набора байтов»

Более формально:

```
double (*my_func)(double, double) = &pow;  
double x = (*my_func)(3, 2);
```

Пример



```
-1.000000  
0.000050  
9.000100
```

```
double diff(double x, double (*f)(double))  
{  
    return (f(x + 0.0001) - f(x)) / 0.0001;  
}  
  
double my_func(double x)  
{  
    return x * x + 5*x;  
}  
  
int main()  
{  
    printf("%lf\n", diff(M_PI, sin));  
    printf("%lf\n", diff(M_PI, cos));  
    printf("%lf\n", diff(2, my_func));  
}
```

Составные типы данных: структуры

Структура – объединение нескольких объектов (могут быть разного типа), под одним именем. (сложное из простого)

Объектами могут быть переменные, массивы, указатели и другие структуры.



Структуры: синтаксис

```
struct Название {  
    тип_поля название_поля;  
    тип_поля название_поля;  
};
```

```
int main()  
{  
    struct Название название_экземпляра = { значение, значение };  
    название_экземпляра.название_поля=...;  
}
```

Структуры: синтаксис

Инициализация полей структуры при ее объявлении:

```
struct Vector {  
    double x, y, z;  
};  
  
int main()  
{  
    struct Vector v = { 3,0,4 };  
    double len = sqrt(v.x * v.x + v.y * v.y + v.z * v.z);  
    return 0;  
}
```

Структуры: синтаксис

Инициализация полей структуры после ее объявления:

```
struct Vector {  
    double x, y, z;  
};  
  
int main()  
{  
    struct Vector v;  
    v.x=3;  
    v.y=0;  
    v.z=4;  
    return 0;  
}
```

Структуры: удобный синтаксис

`typedef СтарыйТип НовыйТип;` - объявление типа НовыйТип как синонима для Старый тип

```
struct Vector{  
double x, y, z;  
};
```

```
struct Vector v = { 3, 0, 4 };
```

```
typedef struct Vector{  
double x, y, z;  
} Vector;
```

```
Vector q = { 3, 0, 4 };
```

Еще немного typedef'a

```
void printok(void* a, void* b, int (*check)(void*, void*))
{
    if (check(a, b))
        printf("ok\n");
    else
        printf("ne ok\n");
}
```

```
typedef int (*CmpFunc)(void*, void*);

void printok(void* a, void* b, CmpFunc check)
{
    if (check(a, b))
        printf("ok\n");
    else
        printf("ne ok\n");
}
```



Структуры в структурах

```
typedef struct Vector{  
    double x, y, z;  
} Vector;
```

```
typedef struct Body{  
    Vector position;  
    Vector speed;  
} Body;
```

Структуры в структурах

```
typedef struct Group {  
    char *name;  
    int size;  
} Group;
```

Есть ли в таком подходе проблемы???

```
typedef struct Student{  
    char *name;  
    Group group;  
} Student;
```

Структуры в структурах

```
typedef struct Group {  
    char *name;  
    int size;  
    Student* starosta;  
} Group;
```

```
typedef struct Student{  
    char *name;  
    Group *group;  
} Student;
```

Иначе будет столько групп, сколько студентов...

Все, что не принадлежит этой структуре – делаем указателем.



Разбираемся с комбинацией структур и указателей

```
typedef struct test {  
    int a;  
    double b;  
} test;
```

```
void print_test(test data) {  
    printf("%d %lf\n", data.a, data.b);  
}
```

```
int main() {  
    test a = { 1, 3.4 };  
    print_test(a);  
    alter_test(&a);  
    print_test(a);  
}
```

```
void alter_test(test* data) {  
    (*data).a = 4;  
    (*data).b = 24.3;  
}
```



Разбираемся с комбинацией структур и указателей

```
typedef struct test {  
    int a;  
    double b;  
} test;
```

```
void print_test(test data) {  
    printf("%d %lf\n", data.a, data.b);  
}
```

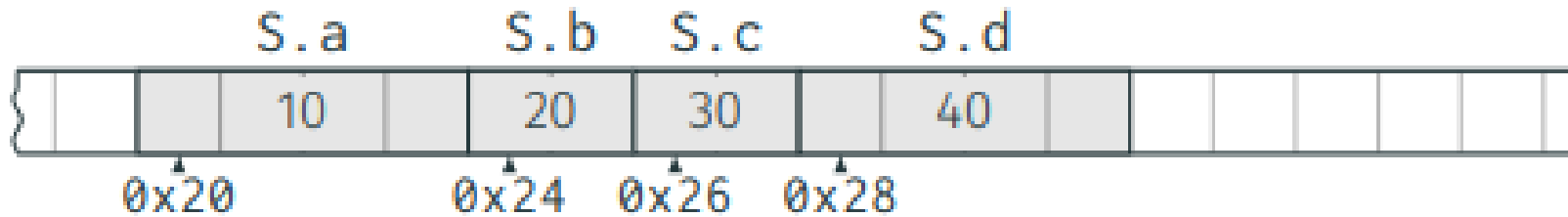
```
int main() {  
    test a = { 1, 3.4 };  
    print_test(a);  
    alter_test(&a);  
    print_test(a);  
}
```

```
void alter_test(test* data) {  
    data->a = 4;  
    data->b = 24.3;  
}
```



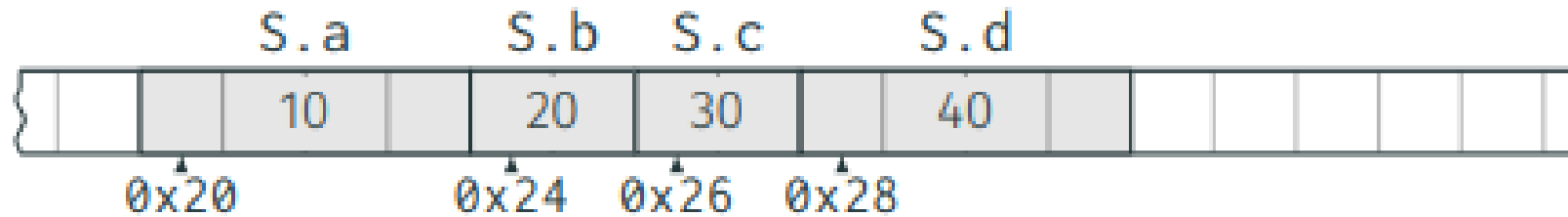
Используем -> для доступа к полю структуры по указателю

Структуры в памяти



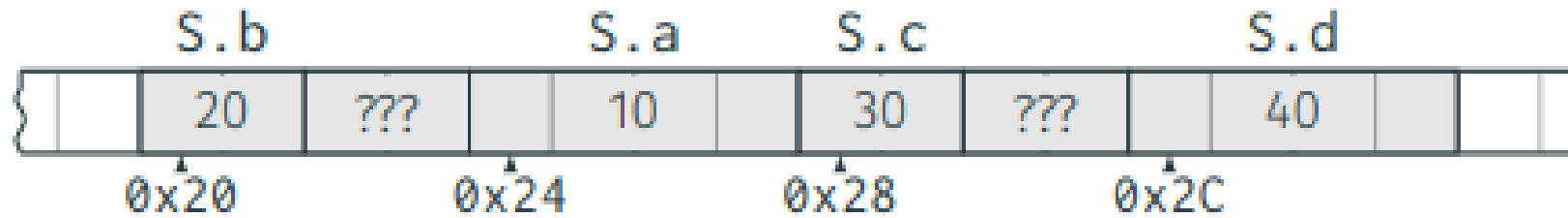
```
struct {  
    int a;  
    short b;  
    short c;  
    int d;  
} S = { 10, 20, 30, 40 };  
printf("%d", sizeof(S)); //12
```

Структуры в памяти



```
struct {  
    short b;  
    int a;  
    short c;  
    int d;  
} S = { 20, 10, 30, 40 };  
printf("%d", sizeof(S)); //??
```

Структуры в памяти



```
struct {  
    short b;  
    int a;  
    short c;  
    int d;  
} S = { 20, 10, 30, 40 };  
printf("%d", sizeof(S)); //16
```



Структуры в памяти



```
struct {  
    short b;  
    int a;  
    int d;  
    short c;  
} S = { 20, 10, 30, 40 };  
printf("%d", sizeof(S)); //16
```

Структуры в памяти

```
#pragma pack(push, 1)
struct {
    short b;
    int a;
    int d;
    short c;
} S = { 20, 10, 30, 40 };
printf("%d", sizeof(S)); //12
#pragma pack(pop)
```



Пример работы со структурой

```
typedef struct String{
    size_t len;
    char chars[];
} String;

String* str = malloc(sizeof(String)+27 * sizeof(char));
if (str != NULL)
    str->len = 27;
    for (size_t i = 0; i < str->len; i++)
        *(str->chars+i) = 'a'+i;
        str->chars[26] = '\0';
    printf("%s\n", str->chars);
    free(str);
```


Пример работы со структурой

```
typedef struct String{
    size_t len;
    char* chars;
} String;

String* str = malloc(sizeof(String));
if (str != NULL)
    str->len = 27;
    str->chars = malloc(sizeof(char)* 27);
    if (str->chars != NULL)
        for (size_t i = 0; i < str->len; i++)
            str->chars[i] = 'a'+i;
            str->chars[26] = '\0';
            printf("%s\n", str->chars);
            free(str->chars);
        free(str);
```

Бонус

```
typedef struct String {  
    size_t len;  
    char chars[];  
} String;
```

```
size_t n = 5;  
String* str = malloc(sizeof(String) + n * sizeof(char));  
if (str != NULL) {  
    for (size_t i = 0; i < str->len; i++)  
        str->chars[i] = 'a' + i;  
    //...  
    free(str);  
}
```

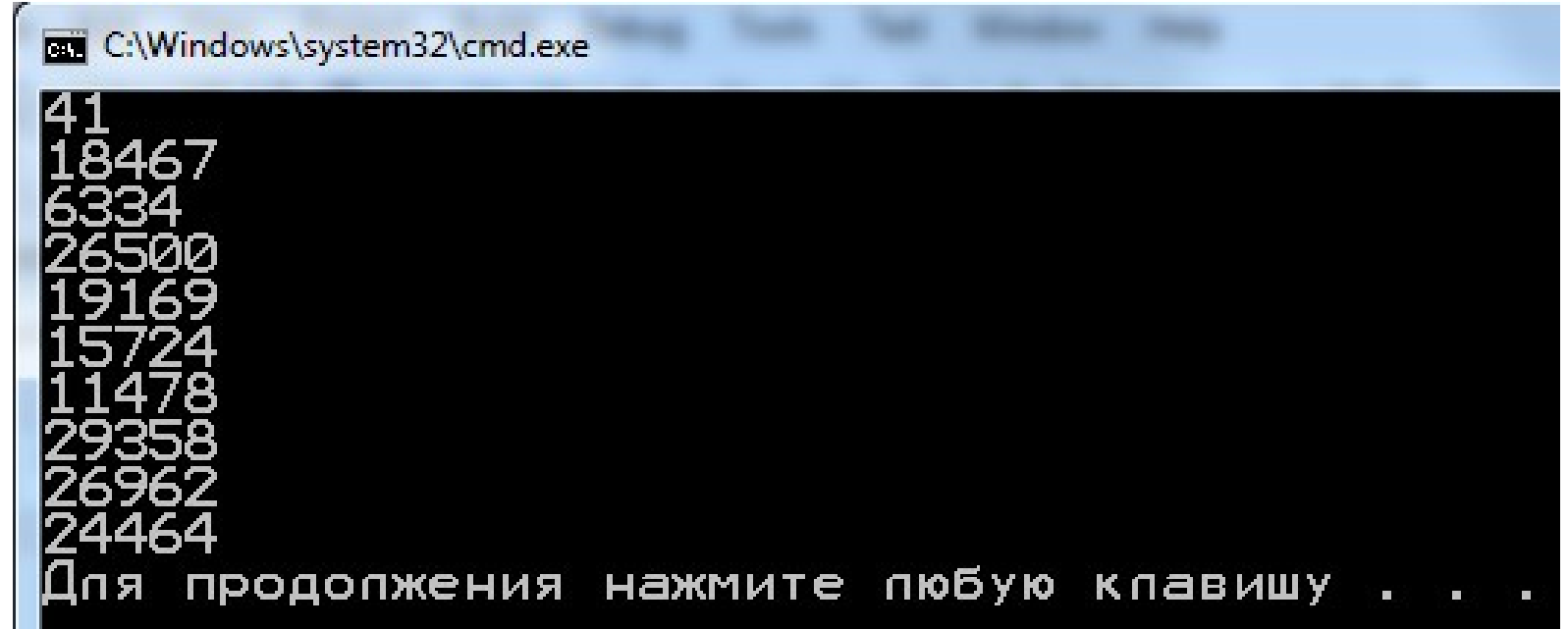


«Случайные» числа в Си

```
#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



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 потере 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).

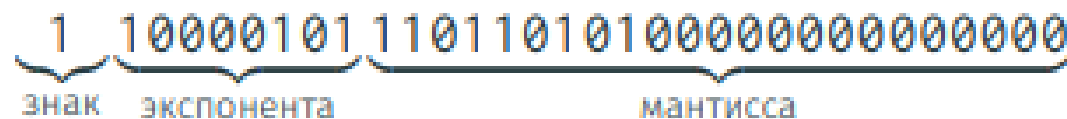
3.1415



Стандарт IEEE 754

Знак, экспонента и мантисса кодируются последовательно следующим количеством бит:

	Binary 32	Binary 64
Знак	1	1
Экспонента	8	11
Мантисса	23	52



Стандарт IEEE 754

Кодируемые числа:

Тип	Экспонента	Мантисса
± 0	0	0
Денормализованные числа	0	0,mmmmmm
Нормализованные числа	1...254	1,mmmmmm
\pm	255	0
Не числа (NaN)	255	Не 0

Вроде все

**ТЫ НЕ ПОЛУЧИШЬ ОШИБКУ
КОМПИЛЯЦИИ,**

**ЕСЛИ НЕ БУДЕШЬ
КОМПИЛИРОВАТЬ КОД**