

## Оглавление по проклятью

Разбор массива байт .....	3
Порядок байт от младшего к старшему .....	3
Вариант №7 (test_tasks).....	3
Код: .....	4
Вывод: .....	5
Объяснение: .....	5
Порядок байт от старшего к младшему. ....	6
Вариант №28 (test_tasks).....	6
Код: .....	8
Вывод: .....	10
Объяснение: .....	10
Преобразования структуры data в массив байт .....	10
Порядок байт от младшего к старшему. ....	10
Вариант №16 (exam1) .....	10
Код: .....	12
Вывод: .....	13
Объяснение: .....	13
Вариант №19 (exam1) .....	14
Код: .....	15
Вывод: .....	17
Объяснение: .....	17
Порядок байт от старшего к младшему .....	18
Вариант №20 (exam1) .....	18
Код: .....	20
Вывод: .....	22
Объяснение: .....	22
Вариант №24 (exam1) .....	24

Код: .....	26
Вывод: .....	28
Объяснение: .....	28
Вариант №37 (exam1) .....	29
Код: .....	31
Вывод: .....	33
Объяснение: .....	33

# Разбор массива байт

## Порядок байт от младшего к старшему

### Вариант №7 (test\_tasks)

Написать программу на C/C++ для работы со структурой данных с учетом архитектурных особенностей некоторой платформы.

Информация о платформе представлена в таблице:

Тип	Размер (байт)	Знак	Выравнивание (байт)
char	1	Нет	1
unsigned short	2	Нет	8

Порядок байт от младшего к старшему.

Структура данных на целевой платформе имеет следующий вид:

```
struct data {  
    char field1[6];  
    char field2;  
    unsigned short field3;  
    char field4;  
};
```

Типы в приведенной структуре данных, возможно, придется адаптировать к используемому в решении задаче компилятору.

Написать функцию для разбора массива байт, содержащего структуру `data`. В реализации необходимо учесть особенности платформы. Вывести на экран значения полей, как показано в примерах ниже.

### Пример 1

Массив байт, содержащий значения структуры `data`:

```
unsigned char test_data1[] = {  
    0xf6, 0x51, 0xd0, 0x58, 0xc9, 0x6d, 0x75, 0x00,  
    0x0d, 0xb6, 0x03,  
};
```

Результат вывода значений полей `data` на экран:

```
246 81 208 88 201 109
117
46605
3
```

## Пример 2

Массив байт, содержащий значения структуры data:

```
unsigned char test_data2[] = {
    0xee, 0x79, 0x31, 0x8a, 0xfa, 0xbc, 0x13, 0x00,
    0x59, 0xeb, 0x1e,
};
```

Результат вывода значений полей data на экран:

```
238 121 49 138 250 188
19
60249
30
```

## Код:

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

struct data {
    // had to change from char to unsigned char
    unsigned char field1[6]; // 6
    unsigned char field2; // 1
    unsigned short field3; // 2
    char field4; // 1
};

unsigned char test_data1[] = {
    0xf6, 0x51, 0xd0, 0x58, 0xc9, 0x6d, 0x75, 0x00,
    0x0d, 0xb6, 0x03,
};

unsigned char test_data2[] = {
    0xee, 0x79, 0x31, 0x8a, 0xfa, 0xbc, 0x13, 0x00,
    0x59, 0xeb, 0x1e,
};

void process(struct data *s, unsigned char d[])
{
    memcpy(&s->field1, &d[0], 6); // from 0 to 5 not including 6
    memcpy(&s->field2, &d[6], 1);
    memcpy(&s->field3, &d[8], 2);
    memcpy(&s->field4, &d[10], 1);
}

void print_output(struct data *s)
{
    printf("%d %d %d %d %d %d \n", s->field1[0], s->field1[1], s->field1[2], s->field1[3], s->field1[4], s->field1[5]);
    printf("%d \n", s->field2);
    printf("%d \n", s->field3);
}
```

```

        printf("%d \n", s->field4);
    }

int main(void)
{
    struct data foo;

    process(&foo, test_data1);
    print_output(&foo);

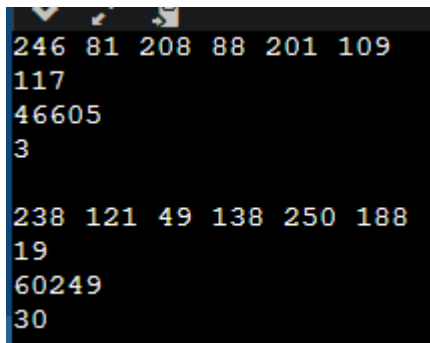
    printf("\n");

    process(&foo, test_data2);
    print_output(&foo);

    return 0;
}

```

## Вывод:



```

246 81 208 88 201 109
117
46605
3

238 121 49 138 250 188
19
60249
30

```

## Объяснение:

Мемсру (куда, откуда, размер байт) копирует (или перемещает) данные из одной структуры в другую.

Тип	Размер (байт)	Знак	Выравнивание (байт)
char	1	Нет	1
unsigned short	2	Нет	8

```

struct data {
    char field1[6];
    char field2;
    unsigned short field3;
    char field4;
};

memcpy(&s->field1, &d[0], 6);
memcpy(&s->field2, &d[6], 1);
memcpy(&s->field3, &d[8], 2);

```

```
memcpy(&s->field4, &d[10], 1);
```

field1 char имеет размер байт 1 и выравнивание 1, это означает, что при заполнении через memcpy, из test\_data1[] и test\_data2[], которые в случае memcpy являются &d[n], заполняются поля fieldN, где для field1 берутся значения из test\_data1 с индексом 1-5 (не включая 6), потому что char field1[6], что означает 6 пар по 1 байту (6\*1), они занимают 0 1 2 3 4 5 индексов по 1 байту, далее поле 2 начинается с 6 индекса и его и занимает (размер 1), далее поле 3 начинается с 8 индекса, почему не с 7? Потому что выравнивание 8 позволяет нам ставить значения поля 3 (unsigned short) только на индексы кратные 8 (0 8 16 24 32...), поэтому поле 3 занимает 8 9 индексы, далее поле 4 занимает индекс 10.

```
void print_output(struct data *s)
{
    printf("%d %d %d %d %d %d \n", s->field1[0], s->field1[1], s->field1[2], s->field1[3], s->field1[4], s->field1[5]);
    printf("%d \n", s->field2);
    printf("%d \n", s->field3);
    printf("%d \n", s->field4);
}
```

Как и говорилось выше, поле 1 это массив данных, где всего 6 элементов по 1 байту, поэтому в выводе каждый из них нужно выводить отдельно.

## **Порядок байт от старшего к младшему.**

### **Вариант №28 (test\_tasks)**

Написать программу на C/C++ для работы со структурой данных с учетом архитектурных особенностей некоторой платформы.

Информация о платформе представлена в таблице:

Тип	Размер (байт)	Знак	Выравнивание (байт)
unsigned short	2	Нет	1

Тип	Размер (байт)	Знак	Выравнивание (байт)
unsigned long	4	Нет	8
char	1	Да	1

Порядок байт от старшего к младшему.

Структура данных на целевой платформе имеет следующий вид:

```
struct data {
    unsigned short field1;
    unsigned long field2;
    unsigned short field3[7];
    char field4;
};
```

Типы в приведенной структуре данных, возможно, придется адаптировать к используемому в решении задачи компилятору.

Написать функцию для разбора массива байт, содержащего структуру `data`. В реализации необходимо учесть особенности платформы. Вывести на экран значения полей, как показано в примерах ниже.

### Пример 1

Массив байт, содержащий значения структуры `data`:

```
unsigned char test_data1[] = {
    0xbb, 0x92, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x93, 0xb9, 0xcc, 0xca, 0xe7, 0xd9, 0x42, 0x91,
    0xc0, 0x27, 0x08, 0x5a, 0x58, 0x63, 0x69, 0x1c,
    0xd2, 0x1b, 0x01,
};
```

Результат вывода значений полей `data` на экран:

```
48018
2478427338
59353 17041 49191 2138 22627 26908 53787
1
```

### Пример 2

Массив байт, содержащий значения структуры `data`:

```
unsigned char test_data2[] = {
    0x02, 0x4a, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x1a, 0xae, 0xbd, 0x5f, 0x0c, 0xdb, 0x70, 0x0a,
};
```

```
    0xb2, 0x1d, 0x40, 0x54, 0xac, 0x9b, 0xf0, 0x65,  
    0x05, 0x75, 0xea,  
};
```

Результат вывода значений полей **data** на экран:

```
586  
447659359  
3291 28682 45597 16468 44187 61541 1397  
-22
```

## Код:

```
#include <stdio.h>  
#include <string.h>  
  
struct data  
{  
    unsigned short field1;  
    unsigned long field2;  
    unsigned short field3[7];  
    char field4;  
};  
  
void*  
revmemcpy(void* dest, const void* src, size_t len)  
{  
    char* d = dest + len - 1;  
    const char* s = src;  
    while (len--)  
        *d-- = *s++;  
    return dest;  
}  
  
unsigned char test_data1[] = {  
    0xbb, 0x92, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
    0x93, 0xb9, 0xcc, 0xca, 0xe7, 0xd9, 0x42, 0x91,  
    0xc0, 0x27, 0x08, 0x5a, 0x58, 0x63, 0x69, 0x1c,  
    0xd2, 0x1b, 0x01,  
};  
  
unsigned char test_data2[] = {  
    0x02, 0x4a, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
    0x1a, 0xae, 0xbd, 0x5f, 0x0c, 0xdb, 0x70, 0x0a,  
    0xb2, 0x1d, 0x40, 0x54, 0xac, 0x9b, 0xf0, 0x65,  
    0x05, 0x75, 0xea,  
};  
  
void process(struct data* s, unsigned char d[])  
{  
    revmemcpy(&s->field1, &d[0], 2);  
    revmemcpy(&s->field2, &d[8], 4);  
    revmemcpy(&s->field3, &d[12], 14);  
    revmemcpy(&s->field4, &d[26], 1);  
}  
  
void print_output(struct data* s)  
{  
    printf("%d \n", s->field1);  
    printf("%ld \n", s->field2);  
    printf("%d %d %d %d %d %d %d \n",  
        s->field3[6],
```



```
        s->field3[5],
        s->field3[4],
        s->field3[3],
        s->field3[2],
        s->field3[1],
        s->field3[0]
    );
    printf("%d \n", s->field4);
}

int main(void)
{
    struct data foo;
    process(&foo, test_data1);
    print_output(&foo);
    printf("\n");
    process(&foo, test_data2);
    print_output(&foo);
    return 0;
}
```

## Вывод:

```
/tmp/17Bw64B4nB.o
48018
2478427338
59353 17041 49191 2138 22627 26908 53787
1

586
447659359
3291 28682 45597 16468 44187 61541 1397
-22
```

## Объяснение:

Принцип такой же, как в Варианте№7, только здесь была добавлена функция `revmemstru`, которая по сути инвертирует работу `memstru` для поскольку у нас здесь обратный порядок байт (от старшего к младшему).

## Преобразования структуры `data` в массив байт

### Порядок байт от младшего к старшему.

#### Вариант №16 (exam1)

Написать программу на C/C++ для работы со структурой данных с учетом архитектурных особенностей некоторой платформы.

Информация о платформе представлена в таблице:

Тип	Размер (байт)	Знак	Выравнивание (байт)
char	1	Да	1
long	4	Да	4

Порядок байт от младшего к старшему.

Структура данных на целевой платформе имеет следующий вид:

```
struct data {
    char field1;
    long field2;
    char field3[5];
};
```

```
char field4;  
};
```

Типы в приведенной структуре данных, возможно, придется адаптировать к используемому в решении задачи компилятору.

Написать функцию для преобразования структуры `data` в массив байт. В реализации необходимо учесть особенности платформы. Вывести на экран значения массива байт, как показано в примерах ниже:

### Пример 1

Функция для заполнения полей структуры `data`:

```
void test_code1(struct data *d) {  
    d->field1 = 18;  
    d->field2 = 550244718;  
    d->field3[0] = 54;  
    d->field3[1] = -75;  
    d->field3[2] = -41;  
    d->field3[3] = -29;  
    d->field3[4] = 26;  
    d->field4 = -26;  
}
```

Результат вывода массива байт на экран:

```
12 00 00 00 6E 11 CC 20  
36 B5 D7 E3 1A E6
```

### Пример 2

Функция для заполнения полей структуры `data`:

```
void test_code2(struct data *d) {  
    d->field1 = 11;  
    d->field2 = 870277776;  
    d->field3[0] = -76;  
    d->field3[1] = 105;  
    d->field3[2] = -111;  
    d->field3[3] = 56;  
    d->field3[4] = -35;  
    d->field4 = -45;  
}
```

Результат вывода массива байт на экран:

```
0B 00 00 00 90 62 DF 33  
B4 69 91 38 DD D3
```

## Код:

```
#include <stdio.h>

struct data
{
    char field1;    // 1
    long field2;    // 4
    char field3[5]; // 5
    char field4;    // 1
};

void test_code1(struct data* d)
{
    d->field1 = 18; // 0
    d->field2 = 550244718; // 4 - 7
    d->field3[0] = 54; // 8 - 12
    d->field3[1] = -75;
    d->field3[2] = -41;
    d->field3[3] = -29;
    d->field3[4] = 26;
    d->field4 = -26; // 13
}

void test_code2(struct data* d) {
    d->field1 = 11;
    d->field2 = 870277776;
    d->field3[0] = -76;
    d->field3[1] = 105;
    d->field3[2] = -111;
    d->field3[3] = 56;
    d->field3[4] = -35;
    d->field4 = -45;
}

void process(struct data* d)
{
    unsigned char buf[14] = {};
    buf[0] = d->field1; // одиночный элемент переносим как есть без циклов
    for (int i = 0; i < 4; i++) {
        buf[4 + i] = ((unsigned char*)&d->field2)[i]; // делаем field2 массивом с
        помощью ()
    }
    for (int i = 0; i < 5; i++) {
        buf[8 + i] = d->field3[i]; // field3 уже массив, получаем значения напрямую
    }
    buf[13] = d->field4; // одиночный элемент переносим как есть без циклов

    // Вывод
    for (int i = 0; i < 14; i++) {
        if (i % 8 == 0) {
            printf("\n");
        }
        printf("%02X ", buf[i]);
    }
}

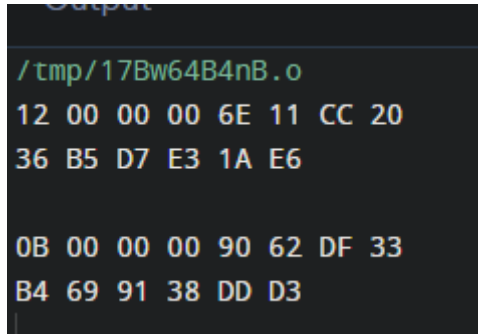
int main(void)
{
    struct data d;
    test_code1(&d);
    process(&d);
    printf("\n");
}
```

```

    test_code2(&d);
    process(&d);
    printf("\n");
    return (0);
}

```

## Вывод:



## Объяснение:

Принцип ввода данных и места в структуре, такой же как в [Варианте 7](#). Размер буфера считается исходя из записанных данных, в данном случае поле 1 занимает индекс 0, поле 2 – 4 5 6 7 (4 байт, выравнивание 4 (индекс кратен 4, 0 4 8 12 ...)) поле 3 – 8 9 10 11 12 (5 элементов массив по 1 байту), поле 4 – 13 (1 байт на 13 индексе) от 0 до 13 всего 14 элементов, то есть размер буфера 14, далее заполнение буфера и количество *i* формируется из значений которые были найдены (индексы). Кстати проверить себя можно по условию, буфер будет равняться количеству значений в выводе примера (результат вывода).

Дополнение: (`unsigned char*`) необходим для адекватной работы float и double и любой структуры, где создается массив скобами (поле 2).

```

void test_code1(struct data* d)
{
    d->field1 = 18; // 0
    d->field2 = 550244718; // 4 - 7
    d->field3[0] = 54; // 8 - 12
    d->field3[1] = -75;
    d->field3[2] = -41;
    d->field3[3] = -29;
    d->field3[4] = 26;
    d->field4 = -26; // 13
}

```

Тип	Размер (байт)	Знак	Выравнивание (байт)
char	1	Да	1
long	4	Да	4

```

struct data {
    char field1;

```

```

long field2;
char field3[5];
char field4;
};

```

### Вариант №19 (exam1)

Написать программу на C/C++ для работы со структурой данных с учетом архитектурных особенностей некоторой платформы.

Информация о платформе представлена в таблице:

Тип	Размер (байт)	Знак	Выравнивание (байт)
char	1	Да	1
unsigned int	4	Нет	1
float	4	Да	8

Порядок байт от младшего к старшему.

Структура данных на целевой платформе имеет следующий вид:

```

struct data {
    char field1;
    unsigned int field2[4];
    unsigned int field3[3];
    float field4;
    unsigned int field5;
};

```

Типы в приведенной структуре данных, возможно, придется адаптировать к используемому в решении задачи компилятору.

Написать функцию для преобразования структуры `data` в массив байт. В реализации необходимо учесть особенности платформы. Вывести на экран значения массива байт, как показано в примерах ниже:

### Пример 1

Функция для заполнения полей структуры `data`:

```

void test_code1(struct data *d) {
    d->field1 = -64;
    d->field2[0] = 3358713824;
    d->field2[1] = 3988898738;
    d->field2[2] = 568596819;
    d->field2[3] = 2713713337;
    d->field3[0] = 2766330088;
    d->field3[1] = 988264849;
    d->field3[2] = 833855387;
}

```

```

d->field4 = -0.015566742978990078;
d->field5 = 3716914552;
}

```

Результат вывода массива байт на экран:

```

C0 E0 E7 31 C8 B2 C3 C1
ED 53 19 E4 21 B9 FA BF
A1 E8 D8 E2 A4 91 B9 E7
3A 9B 9F B3 31 00 00 00
A7 0B 7F BC 78 9D 8B DD

```

## Пример 2

Функция для заполнения полей структуры data:

```

void test_code2(struct data *d) {
    d->field1 = -114;
    d->field2[0] = 2241858822;
    d->field2[1] = 2170970126;
    d->field2[2] = 3183403288;
    d->field2[3] = 1863643585;
    d->field3[0] = 1043990085;
    d->field3[1] = 2966312094;
    d->field3[2] = 2528934039;
    d->field4 = 0.9702175855636597;
    d->field5 = 4156123881;
}

```

Результат вывода массива байт на экран:

```

8E 06 0D A0 85 0E 60 66
81 18 E1 BE BD C1 F1 14
6F 45 06 3A 3E 9E 54 CE
B0 97 78 BC 96 00 00 00
2E 60 78 3F E9 6A B9 F7

```

## Код:

```

#include <stdio.h>

struct data {
    char field1;
    unsigned int field2[4];
    unsigned int field3[3];
    float field4;
    unsigned int field5;
};

void test_code1(struct data* d) {
    d->field1 = -64; // 0
    d->field2[0] = 3358713824; // 1-16
    d->field2[1] = 3988898738;
    d->field2[2] = 568596819;
    d->field2[3] = 2713713337;
    d->field3[0] = 2766330088; // 17-28
    d->field3[1] = 988264849;
    d->field3[2] = 833855387;
}

```

```

    d->field4 = -0.015566742978990078; // 32-35
    d->field5 = 3716914552; // 36-39
}

void test_code2(struct data* d) {
    d->field1 = -114;
    d->field2[0] = 2241858822;
    d->field2[1] = 2170970126;
    d->field2[2] = 3183403288;
    d->field2[3] = 1863643585;
    d->field3[0] = 1043990085;
    d->field3[1] = 2966312094;
    d->field3[2] = 2528934039;
    d->field4 = 0.9702175855636597;
    d->field5 = 4156123881;
}

void process(struct data* d)
{
    unsigned char buf[40] = {};
    int size;
    // field1
    buf[0] = d->field1;
    // field2
    size = 4;
    for (int i = 0; i < 4; i++) { // элементы массива
        for (int j = 0; j < size; j++) { // байты элемента
            buf[1 + size * i + j] = ((unsigned char*)&d->field2[i])[j];
        }
    }
    // field3
    size = 4;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < size; j++) {
            buf[17 + size * i + j] = ((unsigned char*)&d->field3[i])[j];
        }
    }
    // field4
    for (int i = 0; i < 4; i++) {
        buf[32 + i] = ((unsigned char*)&d->field4)[i];
    }
    // field5
    for (int i = 0; i < 4; i++) {
        buf[36 + i] = ((unsigned char*)&d->field5)[i];
    }

    // Вывод
    for (int i = 0; i < 40; i++) {
        if (i % 8 == 0) {
            printf("\n");
        }
        printf("%02X ", buf[i]);
    }
}

int main(void)
{
    struct data d;
    test_code1(&d);
    process(&d);
    printf("\n");
    test_code2(&d);
    process(&d);
    printf("\n");
}

```



```
    return (0);  
}
```

## Вывод:

```
/tmp/17Bw64B4nB.o  
C0 E0 E7 31 C8 B2 C3 C1  
ED 53 19 E4 21 B9 FA BF  
A1 E8 D8 E2 A4 91 B9 E7  
3A 9B 9F B3 31 00 00 00  
A7 0B 7F BC 78 9D 8B DD  
  
8E 06 0D A0 85 0E 60 66  
81 18 E1 BE BD C1 F1 14  
6F 45 06 3A 3E 9E 54 CE  
B0 97 78 BC 96 00 00 00  
2E 60 78 3F E9 6A B9 F7
```

## Объяснение:

Начало тут точно такое же, как в [Варианте16](#), но добавлен данный пример был из-за следующих структур:

```
d->field2[0] = 3358713824; // 1-16  
d->field2[1] = 3988898738;  
d->field2[2] = 568596819;  
d->field2[3] = 2713713337;  
d->field3[0] = 2766330088; // 17-28  
d->field3[1] = 988264849;  
d->field3[2] = 833855387;
```

Которые формируют вложенные циклы:

```
// field2  
size = 4;  
for (int i = 0; i < 4; i++) { // элементы массива  
    for (int j = 0; j < size; j++) { // байты элемента  
        buf[1 + size * i + j] = ((unsigned char*)&d->field2[i])[j];  
    }  
}  
// field3  
size = 4;  
for (int i = 0; i < 3; i++) {  
    for (int j = 0; j < size; j++) {  
        buf[17 + size * i + j] = ((unsigned char*)&d->field3[i])[j];  
    }  
}
```

Где первый цикл отвечает за элементы массива (в случае поля 2 это 4 элемента по 4 байта, а поля 3 это 3 элемента по 4 байта), поэтому цикл с I

отвечает за шаг каждого элемента массива, а цикл с  $j$  за заполнение каждого элемента байтами, `size` у нас размер массива.

## Порядок байт от старшего к младшему

### Вариант №20 (exam1)

Написать программу на C/C++ для работы со структурой данных с учетом архитектурных особенностей некоторой платформы.

Информация о платформе представлена в таблице:

Тип	Размер (байт)	Знак	Выравнивание (байт)
double	8	Да	8
char	1	Да	1
long long	8	Да	8

Порядок байт от старшего к младшему.

Структура данных на целевой платформе имеет следующий вид:

```
struct data {  
    double field1;  
    double field2;  
    char field3;  
    char field4[8];  
    long long field5;  
    double field6;  
};
```

Типы в приведенной структуре данных, возможно, придется адаптировать к используемому в решении задаче компилятору.

Написать функцию для преобразования структуры `data` в массив байт. В реализации необходимо учесть особенности платформы. Вывести на экран значения массива байт, как показано в примерах ниже:

### Пример 1

Функция для заполнения полей структуры `data`:

```
void test_code1(struct data *d) {  
    d->field1 = -0.09270031390292233;  
    d->field2 = -0.2095092678469339;  
    d->field3 = -2;
```

```

d->field4[0] = -3;
d->field4[1] = 12;
d->field4[2] = 75;
d->field4[3] = 16;
d->field4[4] = -98;
d->field4[5] = 20;
d->field4[6] = 24;
d->field4[7] = 28;
d->field5 = -7513552878709493619;
d->field6 = -0.5188763099504088;
}

```

Результат вывода массива байт на экран:

```

BF B7 BB 35 30 8A BF 90
BF CA D1 33 1E CE 45 28
FE FD 0C 4B 10 9E 14 18
1C 00 00 00 00 00 00 00
97 BA 7F 77 1F 68 30 8D
BF E0 9A A2 7D BC FF 4E

```

## Пример 2

Функция для заполнения полей структуры data:

```

void test_code2(struct data *d) {
d->field1 = 0.16816283075653327;
d->field2 = 0.09251461845811049;
d->field3 = 127;
d->field4[0] = 35;
d->field4[1] = -10;
d->field4[2] = 78;
d->field4[3] = -28;
d->field4[4] = 33;
d->field4[5] = -74;
d->field4[6] = -48;
d->field4[7] = 114;
d->field5 = -5508108258527611492;
d->field6 = 0.8798291053898459;
}

```

Результат вывода массива байт на экран:

```

3F C5 86 5C 11 40 44 98
3F B7 AF 09 BC AD F3 E0
7F 23 F6 4E E4 21 B6 D0
72 00 00 00 00 00 00 00
B3 8F 44 B8 DA 10 F5 9C
3F EC 27 8F 5E 36 FC 86

```

## Код:

```
#include <stdio.h>

struct data {
    double field1;
    double field2;
    char field3;
    char field4[8];
    long long field5;
    double field6;
};

void test_code1(struct data* d) {
    d->field1 = -0.09270031390292233; // 0 - 7
    d->field2 = -0.2095092678469339; // 8 - 15
    d->field3 = -2; // 16
    d->field4[0] = -3; // 17-24
    d->field4[1] = 12;
    d->field4[2] = 75;
    d->field4[3] = 16;
    d->field4[4] = -98;
    d->field4[5] = 20;
    d->field4[6] = 24;
    d->field4[7] = 28;
    d->field5 = -7513552878709493619; // 32-39
    d->field6 = -0.5188763099504088; // 40 - 47
}

void test_code2(struct data* d) {
    d->field1 = 0.16816283075653327;
    d->field2 = 0.09251461845811049;
    d->field3 = 127;
    d->field4[0] = 35;
    d->field4[1] = -10;
    d->field4[2] = 78;
    d->field4[3] = -28;
    d->field4[4] = 33;
    d->field4[5] = -74;
    d->field4[6] = -48;
    d->field4[7] = 114;
    d->field5 = -5508108258527611492;
    d->field6 = 0.8798291053898459;
}

void process(struct data* d)
{
    unsigned char buf[48] = {};
    int size;
    // field1
    for (int i = 0; i < 8; i++) {
        buf[7 - i] = ((unsigned char*)&d->field1)[i];
    }
    // field2
    for (int i = 0; i < 8; i++) {
```

```

        buf[15 - i] = ((unsigned char*)&d->field2)[i];
    }
    // field3
    buf[16] = d->field3;

    // field4
    for (int i = 0; i < 8; i++) {
        buf[17 + i] = *(&d->field4[i]);
    }
    // field5
    for (int i = 0; i < 8; i++) {
        buf[39 - i] = ((unsigned char*)&d->field5)[i];
    }

    // field6
    for (int i = 0; i < 8; i++) {
        buf[47 - i] = ((unsigned char*)&d->field6)[i];
    }

    // Вывод
    for (int i = 0; i < 48; i++) {
        if (i % 8 == 0) {
            printf("\n");
        }
        printf("%02X ", buf[i]);
    }
}

int main(void)
{
    struct data d;
    test_code1(&d);
    process(&d);
    printf("\n");
    test_code2(&d);
    process(&d);
    printf("\n");
    return (0);
}

```

## Вывод:

```
Output

/tmp/17Bw64B4nB.o
BF B7 BB 35 30 8A BF 90
BF CA D1 33 1E CE 45 28
FE FD 0C 4B 10 9E 14 18
1C 00 00 00 00 00 00 00
97 BA 7F 77 1F 68 30 8D
BF E0 9A A2 7D BC FF 4E

3F C5 86 5C 11 40 44 98
3F B7 AF 09 BC AD F3 E0
7F 23 F6 4E E4 21 B6 D0
72 00 00 00 00 00 00 00
B3 8F 44 B8 DA 10 F5 9C
3F EC 27 8F 5E 36 FC 86
```

## Объяснение:

Все до начала цикла происходит аналогично объяснениям в [Варианте16](#) и [Варианте19](#), но далее в самих циклах начинает работать принцип от старшего к младшему, то есть обратному порядку байт:

```
// field1
for (int i = 0; i < 8; i++) {
    buf[7 - i] = ((unsigned char*)&d->field1)[i];
}
```

Где в циклах, мы наоборот не идем от индекса к верху, а заполняем значения наоборот от конца к началу (7-i), у поля 1 следующее заполнение:

```
d->field1 = -0.09270031390292233; // 0 - 7
```

То есть это индексы 0 1 2 3 4 5 6 7, но так как у нас обратный порядок байт мы наоборот заполняем буфер в порядке 7 6 5 4 3 2 1 0 (7-i).

ОСОБОЕ внимание прошу обратить на массив чаров:

```
d->field4[0] = -3; // 17-24
d->field4[1] = 12;
d->field4[2] = 75;
d->field4[3] = 16;
d->field4[4] = -98;
d->field4[5] = 20;
d->field4[6] = 24;
d->field4[7] = 28;
```

```
// field4  
  
for (int i = 0; i < 8; i++) {  
    buf[17 + i] = *(&d->field4[i]);  
}
```

И если в прямом порядке байт (от мл. к ст.) данный массив работает без проблем, то вот в обратном порядке есть свои особенности, которые и были реализованы тут (во-первых работа с указателем \*, чтобы не использовать strcpy, во вторых порядок тут остается прямой 17+i):

```
for (int i = 0; i < 8; i++) {  
    buf[17 + i] = *(&d->field4[i]);  
}
```

Все это происходит из-за особенностей чар массивов в си, если сделать по другому, начинают происходить аномалии, начиная от рандомных значений, заканчивая проблемами с конвертацией.

## Вариант №24 (exam1)

Написать программу на C/C++ для работы со структурой данных с учетом архитектурных особенностей некоторой платформы.

Информация о платформе представлена в таблице:

Тип	Размер (байт)	Знак	Выравнивание (байт)
float	4	Да	8
unsigned long	4	Нет	4
char	1	Нет	1
long	4	Да	4

Порядок байт от старшего к младшему.

Структура данных на целевой платформе имеет следующий вид:

```
struct data {  
    float field1;  
    unsigned long field2;  
    char field3[10];  
    unsigned long field4;  
    long field5;  
    float field6;  
};
```

Типы в приведенной структуре данных, возможно, придется адаптировать к используемому в решении задачи компилятору.

Написать функцию для преобразования структуры `data` в массив байт. В реализации необходимо учесть особенности платформы. Вывести на экран значения массива байт, как показано в примерах ниже:

### Пример 1

Функция для заполнения полей структуры `data`:

```
void test_code1(struct data *d) {  
    d->field1 = -0.25192779302597046;  
    d->field2 = 574905390;  
    d->field3[0] = 58;  
    d->field3[1] = 103;  
    d->field3[2] = 76;  
}
```



```

d->field3[3] = 149;
d->field3[4] = 22;
d->field3[5] = 63;
d->field3[6] = 202;
d->field3[7] = 75;
d->field3[8] = 7;
d->field3[9] = 77;
d->field4 = 3825010735;
d->field5 = 1420582020;
d->field6 = 0.5096840858459473;
}

```

Результат вывода массива байт на экран:

```

BE 80 FC AE 22 44 5C 2E
3A 67 4C 95 16 3F CA 4B
07 4D 00 00 E3 FD 08 2F
54 AC 5C 84 00 00 00 00
3F 02 7A A8

```

## Пример 2

Функция для заполнения полей структуры data:

```

void test_code2(struct data *d) {
d->field1 = 0.513469934463501;
d->field2 = 1846762717;
d->field3[0] = 196;
d->field3[1] = 118;
d->field3[2] = 33;
d->field3[3] = 58;
d->field3[4] = 155;
d->field3[5] = 99;
d->field3[6] = 10;
d->field3[7] = 104;
d->field3[8] = 168;
d->field3[9] = 99;
d->field4 = 3862930540;
d->field5 = -713071892;
d->field6 = -0.642917811870575;
}

```

Результат вывода массива байт на экран:

```

3F 03 72 C4 6E 13 5C DD
C4 76 21 3A 9B 63 0A 68
A8 63 00 00 E6 3F A4 6C
D5 7F 62 EC 00 00 00 00
BF 24 96 43

```

## Код:

```
#include <stdio.h>

struct data {
    float field1;
    unsigned long field2;
    unsigned char field3[10];
    unsigned long field4;
    long field5;
    float field6;
};

void test_code1(struct data* d) {
    d->field1 = -0.25192779302597046; // 0 - 3
    d->field2 = 574905390; // 4 - 7
    d->field3[0] = 58; // 8 - 17
    d->field3[1] = 103;
    d->field3[2] = 76;
    d->field3[3] = 149;
    d->field3[4] = 22;
    d->field3[5] = 63;
    d->field3[6] = 202;
    d->field3[7] = 75;
    d->field3[8] = 7;
    d->field3[9] = 77;
    d->field4 = 3825010735; // 20 - 23
    d->field5 = 1420582020; // 24 - 27
    d->field6 = 0.5096840858459473; // 32 - 35
}

void test_code2(struct data* d) {
    d->field1 = 0.513469934463501;
    d->field2 = 1846762717;
    d->field3[0] = 196;
    d->field3[1] = 118;
    d->field3[2] = 33;
    d->field3[3] = 58;
    d->field3[4] = 155;
    d->field3[5] = 99;
    d->field3[6] = 10;
    d->field3[7] = 104;
    d->field3[8] = 168;
    d->field3[9] = 99;
    d->field4 = 3862930540;
    d->field5 = -713071892;
    d->field6 = -0.642917811870575;
}

void process(struct data* d)
{
    unsigned char buf[35] = {};
    int size;
    // field1
    for (int i = 0; i < 4; i++) {
        buf[3 - i] = ((unsigned char*)&d->field1)[i];
    }
    // field2
    for (int i = 0; i < 4; i++) {
        buf[7 - i] = ((unsigned char*)&d->field2)[i];
    }
}
```

```

    }
    // field3
    for (int i = 0; i < 10; i++) {
        buf[8 + i] = *(&d->field3[i]);
    }
    // field4
    for (int i = 0; i < 4; i++) {
        buf[23 - i] = ((unsigned char*)&d->field4)[i];
    }
    // field5
    for (int i = 0; i < 4; i++) {
        buf[27 - i] = ((unsigned char*)&d->field5)[i];
    }

    // field6
    for (int i = 0; i < 4; i++) {
        buf[35 - i] = ((unsigned char*)&d->field6)[i];
    }

    // Вывод
    for (int i = 0; i < 36; i++)
    {
        if (i % 8 == 0) {
            printf("\n");
        }
        printf("%02X ", buf[i]);
    }
}

int main(void)
{
    struct data d;
    test_code1(&d);
    process(&d);
    printf("\n");
    test_code2(&d);
    process(&d);
    printf("\n");
    return (0);
}

```

## Вывод:

```
Output
/tmp/17Bw64B4nB.o
BE 80 FC AE 22 44 5C 2E
3A 67 4C 95 16 3F CA 4B
07 4D 00 00 E3 FD 08 2F
54 AC 5C 84 00 00 00 00
3F 02 7A A8

3F 03 72 C4 6E 13 5C DD
C4 76 21 3A 9B 63 0A 68
A8 63 00 00 E6 3F A4 6C
D5 7F 62 EC 00 00 00 00
BF 24 96 43
```

## Объяснение:

Читать [объяснение варианта 20](#).

## Вариант №37 (exam1)

Написать программу на C/C++ для работы со структурой данных с учетом архитектурных особенностей некоторой платформы.

Информация о платформе представлена в таблице:

Тип	Размер (байт)	Знак	Выравнивание (байт)
long	8	Да	1
int	8	Да	4
char	1	Да	1

Порядок байт от старшего к младшему.

Структура данных на целевой платформе имеет следующий вид:

```
struct data {  
    long field1;  
    int field2;  
    long field3;  
    char field4[10];  
    int field5;  
};
```

Типы в приведенной структуре данных, возможно, придется адаптировать к используемому в решении задачи компилятору.

Написать функцию для преобразования структуры `data` в массив байт. В реализации необходимо учесть особенности платформы. Вывести на экран значения массива байт, как показано в примерах ниже:

### Пример 1

Функция для заполнения полей структуры `data`:

```
void test_code1(struct data *d) {  
    d->field1 = -8418588989129825893;  
    d->field2 = -747860737140953660;  
    d->field3 = -8905682240807956907;  
    d->field4[0] = -11;  
    d->field4[1] = 76;  
    d->field4[2] = -21;  
    d->field4[3] = 85;  
    d->field4[4] = 16;  
    d->field4[5] = 83;  
    d->field4[6] = -40;  
    d->field4[7] = -78;  
    d->field4[8] = -117;  
}
```

```

d->field4[9] = -76;
d->field5 = 3658691376754472353;
}

```

Результат вывода массива байт на экран:

```

8B 2B 29 EB 00 E6 E1 9B
F5 9F 10 9F 35 40 6D C4
84 68 A9 2F B2 23 0E 55
F5 4C EB 55 10 53 D8 B2
8B B4 00 00 32 C6 48 6C
2E 3C 35 A1

```

## Пример 2

Функция для заполнения полей структуры data:

```

void test_code2(struct data *d) {
    d->field1 = 1531015128964307103;
    d->field2 = -2473208637284675586;
    d->field3 = 5188822503015179259;
    d->field4[0] = -43;
    d->field4[1] = 56;
    d->field4[2] = 43;
    d->field4[3] = -72;
    d->field4[4] = -55;
    d->field4[5] = 48;
    d->field4[6] = -79;
    d->field4[7] = 21;
    d->field4[8] = 74;
    d->field4[9] = 45;
    d->field5 = 4630072673857499224;
}

```

Результат вывода массива байт на экран:

```

15 3F 42 25 EB AA 98 9F
DD AD 65 D9 17 38 0F FE
48 02 66 93 2E C5 E3 FB
D5 38 2B B8 C9 30 B1 15
4A 2D 00 00 40 41 52 90
D1 84 F0 58

```

## Код:

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

struct data
{
    long int field1;
    long int field2;
    long int field3;
    char field4[10];
    long int field5;
};

void test_code1(struct data* d)
{
    d->field1 = -8418588989129825893; // 0 - 7
    d->field2 = -747860737140953660; // 8 - 15
    d->field3 = -8905682240807956907; // 16 - 23
    d->field4[0] = -11; // 24 - 33
    d->field4[1] = 76;
    d->field4[2] = -21;
    d->field4[3] = 85;
    d->field4[4] = 16;
    d->field4[5] = 83;
    d->field4[6] = -40;
    d->field4[7] = -78;
    d->field4[8] = -117;
    d->field4[9] = -76;
    d->field5 = 3658691376754472353; // 36 - 43
}

void test_code2(struct data* d) {
    d->field1 = 1531015128964307103;
    d->field2 = -2473208637284675586;
    d->field3 = 5188822503015179259;
    d->field4[0] = -43;
    d->field4[1] = 56;
    d->field4[2] = 43;
    d->field4[3] = -72;
    d->field4[4] = -55;
    d->field4[5] = 48;
    d->field4[6] = -79;
    d->field4[7] = 21;
    d->field4[8] = 74;
    d->field4[9] = 45;
    d->field5 = 4630072673857499224;
}

void process(struct data* d)
{
    unsigned char buf[43] = {};

    // field1
    for (int i = 0; i < 8; i++)
    {
        buf[7 - i] = ((unsigned char*)&d->field1)[i];
    }
    // field2
    for (int i = 0; i < 8; i++)
    {
        buf[15 - i] = ((unsigned char*)&d->field2)[i];
    }
}
```

```

    }
    // field3

    for (int i = 0; i < 8; i++)
    {
        buf[23 - i] = ((unsigned char*)&d->field3)[i];
    }

    // field4

    for (int i = 0; i < 10; i++)
    {
        buf[24 + i] = *(&d->field4[i]);
        //strcpy(buf[32 - i], (&d->field4)[i]);
    }
    // field5

    for (int i = 0; i < 8; i++)
    {
        buf[43 - i] = ((unsigned char*)&d->field5)[i];
    }

    // Вывод
    for (int i = 0; i < 44; i++)
    {
        if (i % 8 == 0)
        {
            printf("\n");
        }
        printf("%02X ", buf[i]);
    }
}

int main(void)
{
    struct data d;
    test_code1(&d);
    process(&d);
    printf("\n");
    test_code2(&d);
    process(&d);
    printf("\n");
    return (0);
}

```



## Вывод:

```
Output
/tmp/17Bw64B4nB.o
8B 2B 29 EB 00 E6 E1 9B
F5 9F 10 9F 35 40 6D C4
84 68 A9 2F B2 23 0E 55
F5 4C EB 55 10 53 D8 B2
8B B4 00 00 32 C6 48 6C
2E 3C 35 A1

15 3F 42 25 EB AA 98 9F
DD AD 65 D9 17 38 0F FE
48 02 66 93 2E C5 E3 FB
D5 38 2B B8 C9 30 B1 15
4A 2D 00 00 40 41 52 90
D1 84 F0 58
```

## Объяснение:

Читать [объяснение варианта 20](#). Но стоит добавить, что порой в:

```
struct data {
    long field1;
    int field2;
    long field3;
    char field4[10];
    int field5;
};
```

Даются значения, которые не соответствуют по размеру или знаку в заполняемой части, в данном случае это:

```
d->field1 = -8418588989129825893; // 0 - 7
d->field2 = -747860737140953660;  // 8 - 15
d->field3 = -8905682240807956907; // 16 - 23
```

Где по понятным причинам, поле 1, 2 и 3 не влезают в long, int и long, что требует задать такие типы данных в самом коде:

```
struct data
{
    long int field1;
    long int field2;
    long int field3;
    char field4[10];
    long int field5;
};
```