

---

# Структуры. Массивы структур. Вложенные структуры. Указатели на структуры. Выравнивание в структурах. Примеры.

---

## Структуры

Структура — это объединение нескольких объектов, возможно, различного типа под одним именем, которое является типом структуры. В качестве объектов могут выступать переменные, массивы, указатели и другие структуры.

Структуры позволяют трактовать группу связанных между собой объектов не как множество отдельных элементов, а как единое целое. Структура представляет собой сложный тип данных, составленный из простых типов.

Объявление структуры:

```
struct ИмяСтруктуры
{
    тип ИмяЭлемента1;
    тип ИмяЭлемента2;
    . . .
    тип ИмяЭлементаn;
};
```

Пример:

```
struct Point
{
    int x;
    int y;
    int z;
};
```

Поля структуры располагаются в памяти в том порядке, в котором они объявлены:

# struct Point

1	2	3	4	5	6	7	8	9	10	11	12
int x				int y				int z			

После того как мы создали структуру, нужно создать её переменную в какой либо функции.

```
struct Point p1;
```

Затем можно инициализировать значения этой структуры вручную:

```
struct Point p1;  
  
p1.x = 0;  
p1.y = 3;  
p1.z = -10;
```

либо через `scanf()`:

```
struct Point p1;  
  
printf("Введите значение x: ");  
scanf("%d", &p1.x);  
  
printf("Введите значение y: ");  
scanf("%d", &p1.y);  
  
printf("Введите значение z: ");  
scanf("%d", &p1.z);
```

Дальше, значения структуры можно изменять, получать и так далее.

## Массив структур

Массив структур позволяет хранить набор однотипных структур.

```
    struct Point triangle[3] =
{
    {1, 3, 1},    // первая вершина треугольника
    {3, 0, 4},    // вторая вершина треугольника
    {0, 1, 5}     // третья вершина треугольника
};
printf("Точка 1 (%d, %d, %d)\n", triangle[0].x, triangle[1].y,
triangle[2].z);
```

---

## Вложенные структуры

Структура может содержать поля, которые сами являются структурами.

Пример:

```
    struct Point // объявляем
{
    int x;
    int y;
    int z;
};

struct Triangle // объявляем
{
    struct Point p1; //структура "треугольник" состоит из точек, что тоже
являются структурами
    struct Point p2;
    struct Point p3;
};

struct Triangle triangle = { // инициализируем
{1, 3, 1},
{3, 0, 4},
{0, 1, 5}
};

printf("вершина 1 с координатами (%d, %d, %d)\n", triangle.p1.x,
triangle.p1.y, triangle.p1.z);
```

---

## Указатели на структуры

Указатели на структуру предоставляют ряд преимуществ:

- Манипулировать указателями проще.
- Передача указателя эффективнее.
- Структуры могут содержать указатели на структуры  
Для доступа к полям структур через указатель используется оператор `->`.
- Простой пример:

```
struct Triangle triangle = {
    {1, 3, 1},
    {3, 0, 4},
    {0, 1, 5}
};

struct Triangle *ptr = &triangle;
printf("вершина 1 с координатами (%d, %d, %d)\n", ptr->p1.x, ptr->p1.y,
ptr->p1.z);
```

## Передача структуры через указатель

С помощью указателей можно изменять поля структуры, по такой же логике, что и с обычными переменными:

```
void movePoint(struct Triangle *triangle)
{
    triangle->p1.x+=3;
    triangle->p1.y+=9;
    triangle->p1.z+=1;
}

int main() {
    struct Triangle triangle = {
        {1, 3, 1},
        {3, 0, 4},
        {0, 1, 5}
    };

    struct Triangle *ptr = &triangle;

    printf("Вершина 1 с координатами (%d, %d, %d)\n", ptr->p1.x, ptr->p1.y,
ptr->p1.z);
    movePoint(&triangle);
    printf("Измененная вершина 1 с координатами (%d, %d, %d)\n", ptr->p1.x,
ptr->p1.y, ptr->p1.z);
```

```
    return 0;
}
```

Вывод:

```
Вершина 1 с координатами (1, 3, 1)
Измененная вершина 1 с координатами (4, 12, 2)
```

---

## Выравнивание в структурах

Выравнивание определяет, как данные структуры размещаются в памяти. Процесс выравнивания направлен на повышение производительности процессора, так как процессоры работают быстрее с данными, размещенными на адресах, кратных размеру данных.

### 1. Граница выравнивания:

- Поле структуры размещается в памяти на адресах, кратных его размеру (например, `int` часто должен начинаться с адреса, кратного 4).
- Размер структуры должен быть кратен размеру ее самого "требовательного" (по выравниванию) поля.

### 2. Пустое пространство

- Чтобы обеспечить корректное выравнивание, компилятор добавляет пустые байты между полями структуры.

Пример выравнивания:

```
#include <stdio.h>

struct Example {
    char c;    // 1 байт
    int i;     // 4 байта
    short s;   // 2 байта
};
```

- Поле `c` (1 байт) занимает 1-й байт.
  - Поле `i` (4 байта) должно быть выровнено на границу 4 байт, поэтому после `c` добавляется 3 байта.
  - Поле `s` (2 байта) занимает следующие 2 байта, и компилятор добавляет еще 2 байта для выравнивания общего размера структуры до кратности 4.
- Как итог, размер структуры будет 12 байт (1 + 3 + 4 + 2 + 2).

*При возможности и желании, лучше располагать поля так, чтобы при выравнивании добавлялось минимум байт.*

---

## Пример

```
#include <stdio.h>

struct Point
{
    int x;
    int y;
    int z;
};

struct Triangle
{
    struct Point points[3]; // массив структур
};

void displayCoords(struct Triangle *triangle)
{
    for (int i = 0; i < (sizeof triangle->points) / (sizeof triangle->points[0]); i++)
    {
        printf("Точка %d: (%d, %d, %d)\n", i+1, triangle->points[i].x, triangle->points[i].y, triangle->points[i].z);
    }
}

void movePoint(struct Triangle *triangle, int pointIndex)
{
    triangle->points[pointIndex].x+=3;
    triangle->points[pointIndex].y+=9;
    triangle->points[pointIndex].z+=1;
}

int main() {
    struct Triangle triangle = {
        { {1, 3, 1},
          {3, 0, 4},
          {0, 1, 5} }
    };
}
```

```
    struct Triangle *ptr = &triangle; // для удобства адрес сразу запишем

    displayCoords(ptr);
    printf("----Трах-тибидох---\n");
    movePoint(ptr, 1);
    displayCoords(ptr);

    return 0;
}
```

```
Точка 1: (1, 3, 1)
Точка 2: (3, 0, 4)
Точка 3: (0, 1, 5)
----Трах-тибидох---
Точка 1: (1, 3, 1)
Точка 2: (6, 9, 5)
Точка 3: (0, 1, 5)
```