Шестое занятие

Структуры данных

Что такое структура?

- В языке Си, структура (struct) — композитный тип данных, инкапсулирующий без сокрытия набор значений различных типов. Порядок размещения значений в памяти задаётся при определении типа и сохраняется на протяжении времени жизни объектов, что даёт возможность косвенного доступа (например, через указатели)

Структура

- Имеет фиксированный размер
- Тот же набор байт только больше
- Создание на стеке
- Создание в сегменте данных
- Создание массивов
- Создание указателя на структуру
- Динамический массив

Пример использования

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

```
int main() {
    struct Point p = {1, 2};
    p.y = -1;
    printf("Point = {%d, %d}", p.x, p.y);
    return 0;
}
```

Либо

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

int main() {
    P.x = 2;
    printf("Point = {%d, %d}", P.x, P.y);
    return 0;
}
```

Структура в памяти

- Int a;
- float b;
- double c;

Int	float	double

Сплошная область памяти в 18 байт

Размещение структуры

Глобально

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

int main() {
    P.x = 2;
    printf("Point = {%d, %d}", P.x, P.y);
    return 0;
}
```

Локально

```
int main() {
    struct Point {
        int x;
        int y;
    } p;
    p.y = -1;
    printf("Point = {%d, %d}", p.x, p.y);
    return 0;
}
```

Первичная инициализация.

- Создание экземпляра структуры придерживается тем же правила что и создание переменных:
  - Глобальные объявления зануляются(т.к. выделены в сегменте данных), а значит все переменные внутри так же равняются нулю
  - При локальном объявлении(в теле функции) память резервируется в **стеке** а значит поля структуры будут инициализированы мусором.

Первичная инициализация

При такой инициализации, явно не инициализированные поля

будут приравнены к нулю

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

int main() {
    struct Point p = {1, 2};
    printf("Point = {%d, %d}", p.x, p.y);
    return 0;
}
```

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

int main() {
    struct Point p = {1};
    printf("Point = {%d, %p}", p.x, p.y);
    return 0;
}
```

Практика

- Напишем функцию которая распечатает данные структуры **Person** состоящую из полей: firstName, lastName, age, sex

#define

- Директива **#define** определяет идентификатор и последовательность символов, которой будет замещаться данный идентификатор при его обнаружении в тексте программы. Стандартный вид директивы следующий:

- #define имя_макроса последовательность_символов

#define

```c
#define Point struct Point_t

struct Point_t {
    int x;
    int* y;
};
```

```c
int main() {
    Point p;
    printf("Point = {%d, %p}", p.x, p.y);
    return 0;
}
```

typedef

- Объявление **typedef**, которое содержит имя, которое внутри своей области является синонимом для типа, указанного частью объявления **type-declaration**.

```
typedef type-declaration synonym;
```

typedef

```
struct Point {
    int x;
    int* y;
} typedef Point, MyPoint;

typedef struct Point Point, MyPoint;
```

```
int main() {
    Point p;
    MyPoint myP;
    printf("Point = {%d, %p}", p.x, p.y);
    return 0;
}
```

Динамическое выделение памяти под структуры

```
Point* p = (Point*) malloc(sizeof(Point));

printf("Point = {%d, %d}", (*p).x, (*p).y);

printf("Point = {%d, %d}", p->x, p->y);
```

Не все так очевидно

- Сколько весит структура?

8 байт

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

8 байт

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

8 байт

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

Выравнивание данных

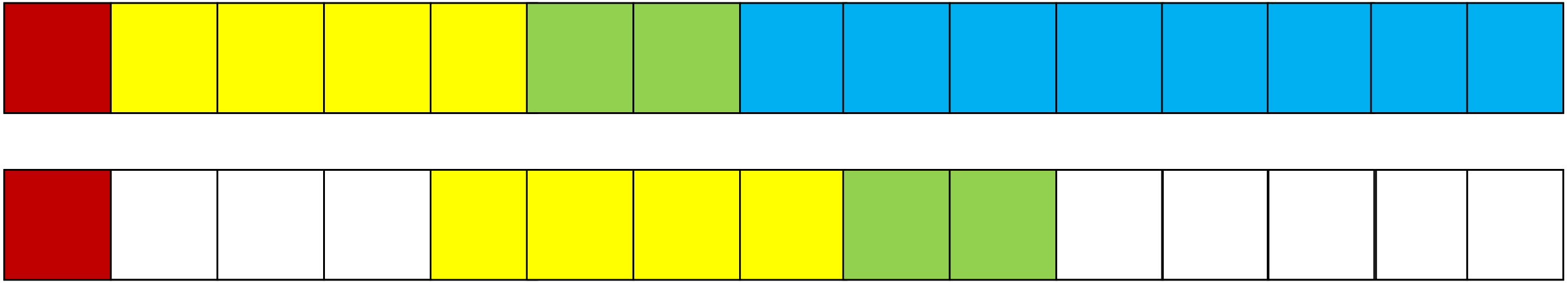
- Смещение данных в структуре до адреса кратного их размеру.

~~Скучный~~ Интересный факт

- На процессорах x86 и ARM примитивные типы не могут находиться в произвольной ячейке памяти. Каждый тип, кроме char, требует выравнивания. char может начинаться с любого адреса, однако двухбайтовый short должен начинаться только с четного адреса, четырехбайтный int или float — с адреса, кратного 4, восьмибайтные long или double — с адреса, кратного 8. Наличие или отсутствие знака значения не имеет. Указатели — 32-битные (4 байта) или 64-битные (8 байт) — также выравниваются.

Пример

```
struct Point {
    char a;
    int b;
    short c;
    double d;
};
```



Можно но не нужно

- Можно убрать выравнивание с помощью выражения
- #pragma pack(1)

Оптимизация

- Данные отсортированы по объему, от большего к меньшему

Списки

- Структуры не могу содержать в себе другие структуру того же типа, но могут содержать ссылки на них.

```
struct Point {
    int x;
    int y;
    Point* next;
};
```

Списки

- Список – цепочка элементов связанных между собой ссылками

Посмотрим на практике.

- Напишем программу в которую можно вводить не *ограниченное кол-во координат точек, концом ввода будет отрицательное число, после чего программа распечатывает все точки.

Виды списка

- Односвязные – каждый элемент списка имеет ссылку лишь на следующий элемент.
- Двусвязные – каждый элемент списка имеет ссылку на следующий и предыдущий элементы.

Список дел

- Напишем программу списка дел. Каждый элемент содержит в себе сообщение, свой порядковый номер, время выполнения. Элементы можно добавлять, удалять, изменять.