Структуры, объединения.

Структуры

Структура представляет собой одну или несколько переменных (возможно разного типа), которые объединены под одним именем.

Структуры помогают в организации сложных данных, потому что позволяют описывать множество логически связанных между собой отдельных элементов как единое целое.

Способы определения переменных структурного типа

Раздельные определения типа и переменных

```
struct date
{
    int day;
    int month;
    int year;
};  // !

struct date birthday;
struct date exam;
```

Совмещенные определения типа и переменных

```
struct date
{
    int day;
    int month;
    int year;
} birthday, exam;
```

Тег структуры

Имя, которое располагается за ключевым словом struct, называется *тегом* структуры.

- Используется для краткого обозначения той части объявления, которая заключена в фигурные скобки.
- Тег может быть опущен (безымянный тип).

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

struct point a, b, c;
```

```
struct
{
    int x;
    int y;
} a, b, c;
```

Имена тегов и полей структур

- «Тело» структуры представляет собой самостоятельную область видимости: имена в этой области не конфликтуют с именами из других областей.
- Тег структуры не распознается без ключевого слова struct. Благодаря этому тег не конфликтует с другими именами в программе.

Имена тегов и полей структур

```
struct s_1
{
    int a;
    int b;
    char c;
};

struct s_2
{
    int a;
    int b;
    double d;
};

int main(void)
{
    struct s_1 a;
    struct s_2 b;
    int s_1;

    // ...
```

Поля структуры

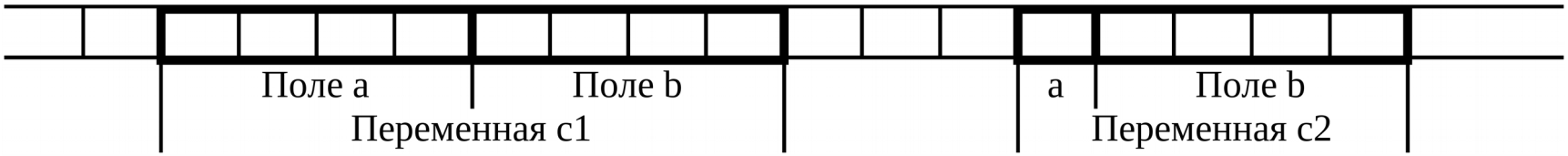
Перечисленные в структуре переменные называются *полями* структуры.

- Поля структуры располагаются в памяти в порядке описания.
- С целью оптимизации доступа компилятор может располагать поля в памяти не одно за другим, а по адресам кратным, например, размеру поля.
- Адрес первого поля совпадает с адресом переменной структурного типа.
- Поля структуры могут иметь любой тип, кроме типа этой же структуры, но могут быть указателями на него.

Поля структуры

```
struct
{
   char a;
   int b;
} c1;

sizeof(c1) == 8

(char*) &c1 == &c1.a
```

```
#pragma pack(push, 1)
struct
{
   char a;
   int b;
} c2;
#pragma pack(pop)

sizeof(c2) == 5

(char*) &c2 == &c2.a
```



Структуры и массивы

Структуры и массивы могут комбинироваться без каких-либо ограничений.

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

struct point triangle[3];
```

```
struct control_area
{
    ...

    char hash[32];
};
```

Инициализация

- Для инициализации переменной структурного типа необходимо указать список значений, заключенный в фигурные скобки.
- Значения в списке должны появляться в том же порядке, что и имена полей структуры.
- Если значений меньше, чем полей структуры, оставшиеся поля инициализируются нулями.

Инициализация

```
struct date
{
    int day;
    int month;
    int year;
};

#define NAME_LEN 15

struct person
{
    char name[NAME_LEN+1];
    struct date birth;
};

int main(void)
{
    struct date today =
                {28, 10, 2015};
```

```
struct date day = {28};
struct date year = {, , 2015};
// error: expected expression before ',' token

struct person rector =
   {"Aleksandrov", {7, 4, 1951}};

struct date holidays[] =
{
    {1, 11, 2015},
    {3, 11, 2015},
    {8, 11, 2015}
};

//...
```

Инициализация в C99

```
struct date exam =
            {.date = 4, .month = 1, .year = 2016};
```

+ Такую инициализацию легче читать и понимать.
+ Значения могут идти в произвольном порядке.
+ Отсутствующие поля получают нулевые значения.
+ Возможна комбинация со старым способом.

```
struct date exam =
            {.date = 4, 1, .year = 2016};
```

Операции над структурами

- Доступ к полю структуры осуществляется с помощью операции ".", а если доступ к самой структуре осуществляется по указателю, то с помощью операции "->".

```
struct date today, *tomorrow, some_date;

today.day   = 28;
today.month = 10;
today.year  = 2015;

tomorrow = malloc(sizeof(struct date));
if (!tomorrow)
    return -1;

tomorrow->day = 29;
tomorrow->month = 10;
(*tomorrow).year = 2015;
```

Операции над структурами

- Структурные переменные одного типа можно присваивать друг другу (замечание: у разных безымянных типов тип разный).

```
some_date = today;
```

- Структуры нельзя сравнивать с помощью “==” и “!=”.

```
if (today == *tomorrow)
// error: invalid operands to binary == (have 'struct date'
// and 'struct date')
```

- Структуры могут передаваться в функцию как параметры и возвращаться из функции в качестве ее значения.

Операции над структурами

```
void print(struct date d)
{
    printf(“%02d.%02d.%04d”, d.day, d.month, d.year);
}

void print_ex(const struct date *d)
{
    printf(“%02d.%02d.%04d”, d->day, d->month, d->year);
}
```

Передача структур с помощью указателей

- **Эффективность (экономия стека и времени на копировании данных).**
- **Необходимость изменения переменной (например, FILE*).**

```
struct date get_student_date(void)
{
    struct date d = {25, 1, 2016};

    return d;
}
```

Операции над структурами (особенности)

```
struct s
{
   int a[5];
};

void f(struct s elem)
{
   for (int i=0; i<5; i++)
      elem.a[i] = 0;
}

// ...
struct s s_1 =
      {{1, 2, 3, 4, 5}};
struct s s_2;
```

```
// «присваивание» массивов
s_2 = s_1;

for(int i=0; i<5; i++)
  printf(“%d ”, s_2.a[i]);
// 1 2 3 4 5

// «передача» массива по значению
f(s_2);

for(int i=0; i<5; i++)
  printf(“%d ”, s_2.a[i]);
// 1 2 3 4 5
```

Особенности использования структур

```
#define NAME_LEN 20
struct person
{
    ...

    char name[NAME_LEN+1];
};
```

```
struct person
{
    ...

    char *name;
};
```

Первый вариант:

"+" простота присваивания;

"-" NAME_LEN и неэффективность использования памяти.

Второй вариант: все наоборот.

Особенности использования структур

```
// h-файл
...

typedef struct _date
{
    int day;
    int month;
    int year;

} date;

...
```

```
// c-файл
...

date a, b;

...
a = b;        // !
```

А если date содержит указатели?

C99: особенности использования структур - flexible array member

```
struct {int n, double d[]};
```

- Подобное поле должно быть последним.
- Нельзя создать массив структур с таким полем.
- Структура с таким полем не может использоваться как член в «середине» другой структуры.
- Операция sizeof не учитывает размер этого поля (возможно, за исключением выравнивания).
- Если в этом массиве нет элементов, то обращение к его элементам – неопределенное поведение.

C99: особенности использования структур - flexible array member

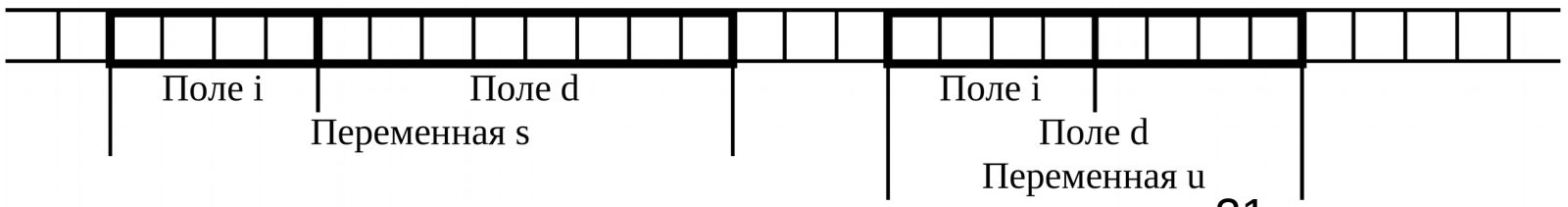
```
struct s* create_s(int n, const double *d)
{
    struct s *elem = malloc(sizeof(struct s) + n * sizeof(double));
    struct s *elem = malloc(sizeof(struct s) + sizeof(double [n]));

    if (elem)
    {
        elem->n = n;
        memmove(elem->d, d, n * sizeof(double));
    }

    return elem;
}

struct {int n, double d[1]};  // workaround для cc ≤ C99
```

Объединения

Объединение, как и структура, содержит одно или несколько полей возможно разного типа. Однако все поля объединения разделяют одну и ту же область памяти.

```
struct
{
    int i;
    double d;
} s;
```

```
union
{
    int i;
    double d;
} u;
```



Объединения

Присвоение значения одному члену объединения обычно изменит значение других членов.

```
printf("u.i %d, u.d %g\n", u.i, u.d);
// u.i 2293664, u.d 1.7926e-307

u.i = 5;
printf("u.i %d, u.d %g\n", u.i, u.d);
// u.i 5, u.d 1.79255e-307

u.d = 5.25;
printf("u.i %d, u.d %g\n", u.i, u.d);
// u.i 0, u.d 5.25
```

Инициализация объединений

```
struct s_t
{
    int i;
    double d;
};

...

struct s_t s = {1, 5.25};
```

```
union u_t
{
    int i;
    double d;
};

...

union u_t  u_1 = {1};

// только c99
union u_t  u_2 = { .d = 5.25 };
```


23

Использование объединений

- Экономия места.

```
struct library_item {
    int number;
    int item_type;
    union {
        struct {
            char author[NAME_LEN + 1];
            char title[TITLE_LEN + 1];
            char publisher[PUBLISHER_LEN + 1];
            int year;
        } book;
        struct {
            char title[TITLE_LEN + 1];
            int year;
            int volume;
        } magazine;
    } item;
};
```

Использование объединений

(«развернутая запись» предыдущего примера)

```
#define NAME_LEN        20
#define TITLE_LEN       20
#define PUBLISHER_LEN   10

struct book_t {
 char author[NAME_LEN+1];
 char title[TITLE_LEN+1];
 char publisher[PUBLISHER_LEN+1];
 int  year;
};

struct magazine_t {
 char title[TITLE_LEN+1];
 int  year;
 int  volume;
};
```

```
union item_t {
 struct book_t     book;
 struct magazine_t magazine;
};

typedef enum
 {KIND_BOOK, KIND_MAGAZINE}
                  kind_item_t;

struct library_item_t {
 int           number;
 kind_item_t   kind;
 union item_t item;
};
```


25

Использование объединений

- Создание структур данных из разных типов.

```
typedef enum { KIND_INT, KIND_DOUBLE } kind_num_t;

typedef struct
{
    kind_num_t kind;
    union {
        int i;
        double d;
    } u;
} number_t;

number_t arr[10];
```

Использование объединений

- Разный взгляд на одни и те же данные (машинно-зависимо).

```
union word
{
    unsigned short word;
    struct word_parts
    {
        unsigned char lo;
        unsigned char hi;
    } parts;
} a;

a.word = 0xABCD;

printf("word 0x%4x, hi part 0x%2x, lo part 0x%2x",
                              a.word, a.parts.hi, a.parts.lo);
```