Строки/структуры и динамическое выделение памяти

(часть 2)

Структуры переменного размера

TLV (Type (или Tag) Length Value) - схема кодирования произвольных данных в некоторых телекоммуникационных протоколах.

Type – описание назначения данных.
Length – размер данных (обычно в байтах).
Value – данные.
Первые два поля имеют фиксированный размер.

Структуры переменного размера

TLV кодирование используется в:

- семействе протоколов TCP/IP
- спецификация PC/SC (smart cards)
- ASN.1
- …

Структуры переменного размера

Преимущества TLV кодирования:

- простота разбора;
- «тройки» TLV с неизвестным типом (тегом) могут быть пропущены при разборе;
- «тройки» TLV могут размещаться в произвольном порядке;
- «тройки» TLV обычно кодируются двоично, что позволяет выполнять разбор быстрее и требует меньше объема по сравнению с кодированием, основанном на текстовом представлении.

Flexible array member (C99)

```
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.

Flexible array member (C99)

```
struct s* create_s(int n, const double *d)
{
    assert(n >= 0);

    struct s *elem = malloc(sizeof(struct s) + n * sizeof(double));

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

    return elem;
}
```

Flexible array member до C99

```
struct s
{
    int n;
    double d[1];
};

struct s* create_s(int n, const double *d)
{
    assert(n >= 0);

    struct s *elem = calloc(sizeof(struct s) +
                                    (n > 1 ? (n - 1) * sizeof(double) : 0), 1);
    if (elem)
    {
        elem->n = n;
        memmove(elem->d, d, n * sizeof(double));
    }

    return elem;
}
```

"unwarranted chumminess with the C implementation"
(c) Dennis Ritchie

Flexible array member vs pointer field

- Экономия памяти.
- Локальность данных (data locality).
- Атомарность выделения памяти.
- Не требует «глубокого» копирования и освобождения.