Структуры. Массивы структур.

Структуры в Си

В реальном мире существует множество объектов.

Объекты реального мира делятся на категории (типы, классы), у каждой категории (типа, класса) есть название и набор свойств.

Объекты одной категории (типа, класса) имеют одинаковые наборы свойств, но отличаются значениями этих свойств.

Структура в Си — конструкция (составной тип данных), которая позволяет объединять (агрегировать) данные различных типов под одним именем.

Для описания структуры нужно определить имя и набор ***полей***.

Описание структур

Для работы со структурами в языке Си нужно спроектировать структурный тип (шаблон) и описать переменную (объект) структурного типа.

Синтаксис описания:

```
struct <имя структурного типа>
{
   тип_поля1 имя_поля1;
   …
   тип_поляN имя_поляN;
};
```

Пример:

```
struct T
{
    int a;
    float b;
};
```

Описание структур

Описание глобального типа

```
struct T
{
    int a;
    float b;
}; /* тип struct T, «;» - обязательно!!  */

/*  ---------------- */

int main()
{
    struct T t1; /* локальная переменная типа struct T */
    …
    return 0;
}
```

Описание структур

Описание глобальной переменной (***Потенциальная уязвимость!***)

```
struct T
{
    int a;
    float b;
} t0; /* глобальная переменная типа struct T */

/* ---------------- */

int main()
{
    <Действия с t0>
    …
    return 0;
}
```

Описание структур

Описание локальных переменных локального типа

```
int main()
{
    struct T
    {
        int a;
        float b;
    } t0, t1; /* локальные переменные типа struct T */
    …
    return 0;
}
```

Описание структур

Описание нового типа данных (**typedef**, полный вариант)

```
struct T
{
    int a;
    float b;
};

typedef struct T typeT; /* тип данных typeT */

int main()
{
    typeT t1; /* переменная типа typeT */
    …
    return 0;
}
```

Описание структур

Описание нового типа данных (**typedef**, краткий вариант)

```
typedef struct T
{
    int a;
    float b;
} typeT; /* тип данных typeT */

int main()
{
    typeT t1; /* переменная типа typeT */
    …
    return 0;
}
```

Типы полей в структурах

- Символы (**`char`**)
- Числа (**`short, int, long, float, double …`**)
- Массивы (одномерные и многомерные)
- Строки
- Структуры с другими шаблонами
- Указатели на все перечисленное

Операции со структурами

Доступ к полям структуры

Пусть


```
int main()
{
    struct T
    {
        int a;
        float b;
    } t0, t1;

    t0.a = 10; /* оператор квалификации (составной) */
    t0.b = 12.34; /* затем – как с обычными переменными */
    return 0;
}
```

Операции со структурами

Доступ к полям структуры

Пусть

```
int main()
{
    struct T
    {
        int a;
        float b;
        char c[15];
    } t0, t1;

    t0.a = 10;
    t0.b = 12.34;
    t0.c[5] = 'z';

    return 0;
}
```

Операции со структурами

Присваивание структур

Пусть

```
int main()
{
    struct T
    {
        int a;
        float b;
    } t0, t1;

    t0.a = 10;
    t0.b = 12.34;

    t1 = t0; /* структуры можно присваивать! */
    return 0;
}
```

Операции со структурами

Поля структуры как переменные

Пусть

```
int main()
{
    struct T
    {
        int a;
        float b;
    } t0, t1;

    scanf("%d %f", &t0.a, &t0.b);
    …

    return 0;
}
```

Операции со структурами

Инициализация структур

Пусть

```
int main()
{
    struct T
    {
        int a;
        float b;
    } t0, t1;

    t0 = {10, 12.34};
    t1 = {5, 4.32};

    return 0;
}
```

Операции со структурами

Инициализация структур

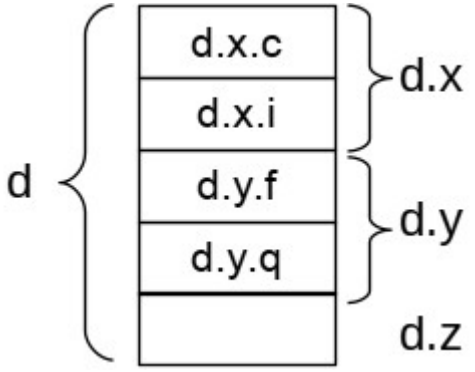
Пусть

```
struct T
{
    int a;
    float b;
};

int main()
{
    struct T t0 = {10, 12.34};
    struct T t1 = {5, 4.32};

    …
    return 0;
}
```

Вложенные структуры

Вложенные (составные) структуры образуются, если одно или несколько полей структуры также являются структурами.

```
struct t1 { char c; int i; };
struct t2 { float f; int q; };
struct t3 { struct t1 x; struct t2 y; int z; } d;

d.x.c = 'a';
d.x.i = 2;
d.y.f = 3.62;
d.y.q = 3;
d.z = 5;
```



Анонимные структуры

```
#include <stdio.h>
#include <math.h>
int main()
{
    struct
    {
        float x;
        float y;
    } t0, t1;
    float r;

    r=sqrt(pow((t0.x-t1.x),2)+pow((t0.y-t1.y),2));
    printf("%.3f",r);
    return 0;
}
```

Имеет смысл только внутри функций (имя структурного типа отсутствует, тип использовать нельзя).
См. пример lect-10-01.c

Структуры в памяти

1. В памяти поля структуры размещаются в порядке, указанном при описании

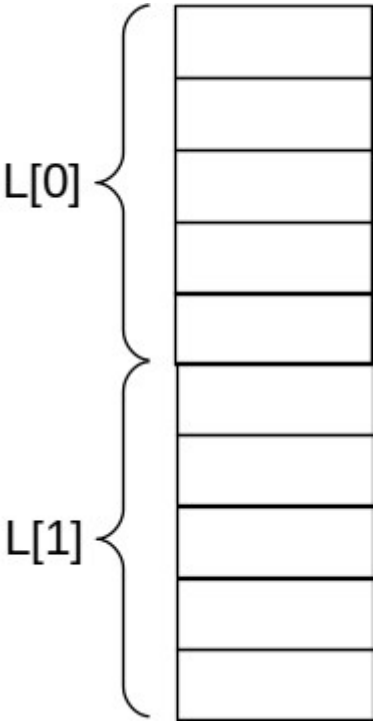
2. Для структур выделяются блоки, кратные 8 байтам (однако может зависеть от компилятора) – эффект «выравнивания».

См. пример lect-10-02.c (эксперименты с полями).

Массивы структур

Для структур типа `struct t3`

```
struct t3
{
   struct t1 x;
   struct t2 y;
   int z;
} L[5];
```



Обращение к полям:

```
L[0].x.c = 'a';
L[0].y.q = 3;
```

Возможно также

```
typedef struct t3 mas0; /* вне main() */
…
mas0 L[5]; /* в main() */
```

См. также пример lect-10-03.c

Массивы структур

Для формирования массивов структур разумно использовать файлы.

Каждая строка файла — элемент массива структур.

Данные для полей разделяются каким-либо символом-разделителем (***не пробелом!***).

Такой формат текстового файла называется CSV — «Comma separated values». Возможные разделители - **,** **;** **:** **|** …

См. пример lect-10-04.c

Задача в том, чтобы прочитать строку, разделить на элементы массива строк, а затем элементы массива строк преобразовать в значения полей элемента массива структур.

Структуры и файлы

1. Текстовый файл (**`fgets(), fprintf() …`**)

2. Бинарный файл (**`fwrite(), fread()`**)
Примеры: lect-10-05a.c, lect-10-05b.c

3. Позиционирование в файле
В **`stdio.h`** определен тип **`fpos_t`** (структура)
Этот тип используется в функциях **`fgetpos()`** и **`fsetpos()`** для получения и установки позиции в файле.
Примеры: lect-10-06.c, lect-10-07.c

Динамические массивы в полях структуры

См. пример lect-10-08.c

1. При описании структуры нельзя инициализировать указатели как NULL.

2. Имеет смысл проверять результат выделения памяти для всех полей сразу.

3. Нужно аккуратно обрабатывать ошибки выделения памяти и очистку памяти.

Структура как параметр функции

См. пример lect-10-09.c

1. В отличие от массива структуру как параметр функции можно передать по значению.

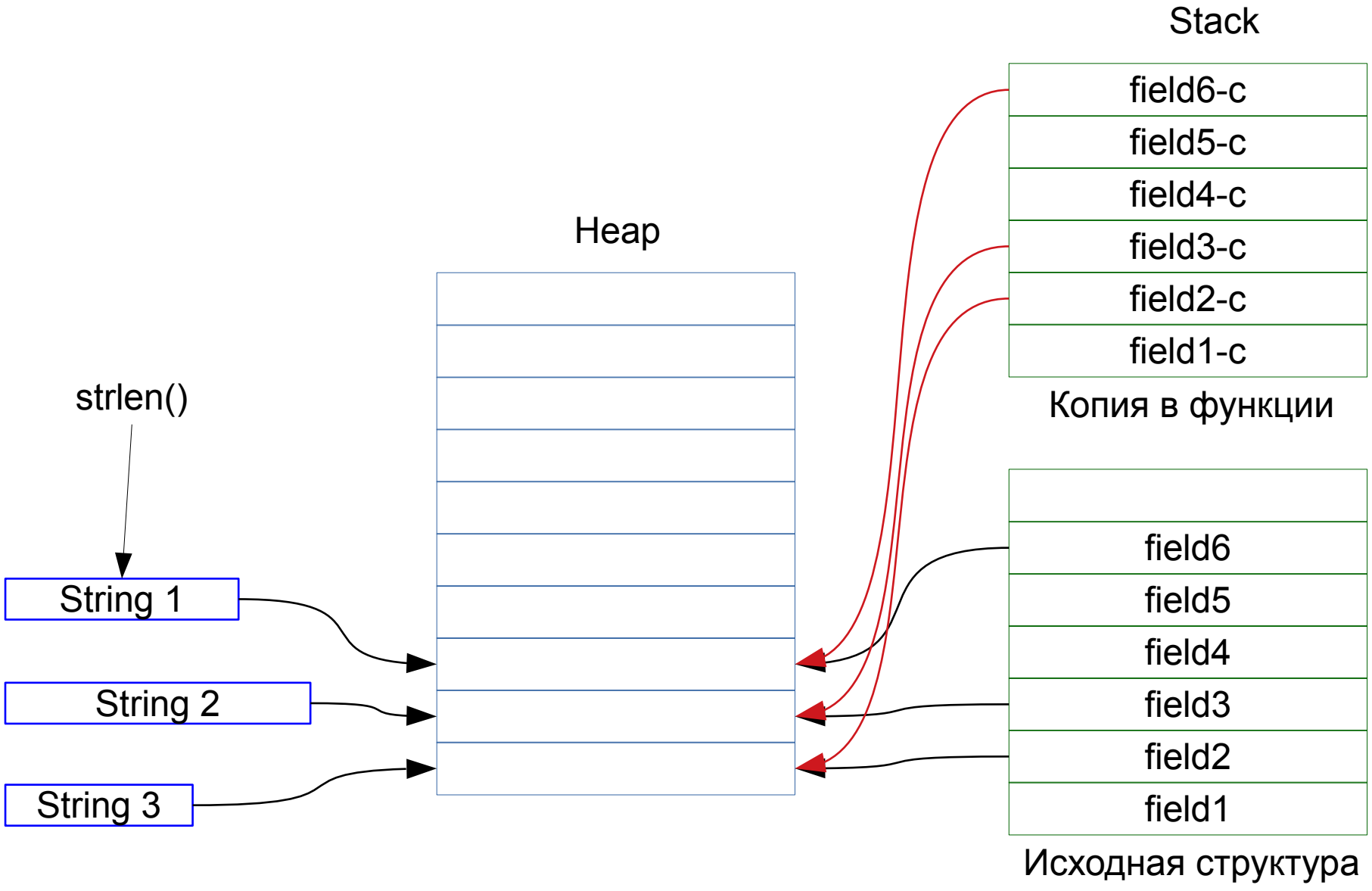
2. Функция может вернуть структуру по имени (функция должна иметь соответствующий тип).

3. Массив структур как параметр функции передается по ссылке.

4. Объявление типа должно быть глобальным (вне функций), иначе для функции тип будет неизвестным.

Д/З: используя пример lect-10-09.c, написать функцию **`fill_struct()`**, заполняющую описанную структуру значениями из элементов массива строк.

Структура как параметр функции



Упаковка структур

Директива **`#pragma`** определяет действия, зависящие от реализации компилятора.

Если препроцессор не распознает конкретный вариант директивы **`#pragma`**, он ее просто игнорирует.

`#pragma pack(`*`N`*`)` позволяет изменять режим выравнивания в структурах при их размещении в памяти.

`#pragma pack(1)` — выравнивание по байтам (самая «плотная» упаковка)

`#pragma pack(2)` — выравнивание по «словам» (скорее всего, по 4 байта)

`#pragma pack(4)` — выравнивание по «двойным словам» (скорее всего, по 8 байтов, самый «рыхлый» вариант).