5. Структуры данных.

Структуры ("записи") представляют собой агрегаты разнородных данных (полей **разного** типа); в отличие от массивов, где все элементы имеют **один и тот же** тип.

```
    struct {
            int x, y;   /* два целых поля */
            char s[10]; /* и одно - для строки */
    } s1;
```

Структурный тип может иметь имя:

```
    struct XYS {
            int x, y;     /* два целых поля */
            char str[10]; /* и одно - для строки  */
    };
```

Здесь мы объявили тип, но не отвели ни одной переменной этого типа (хотя могли бы). Теперь опишем переменную этого типа и указатель на нее:

```
    struct XYS s2, *sptr = &s2;
```

Доступ к полям структуры производится по имени поля (а не по индексу, как у массивов):
имя_структурной_переменной.имя_поля
указатель_на_структуру -> **имя_поля**

то есть

```
       не                          а
    #define ВЕС  0            struct { int вес, рост; } x;
    #define РОСТ 1            x.рост = 175;
    int x[2]; x[РОСТ] = 175;
```

Например

```
    s1.x = 13;
    strcpy(s2.str, "Finish");
    sptr->y = 27;
```

Структура может содержать структуры **другого** типа в качестве полей:

```
    struct XYS_Z {
            struct XYS xys;
            int z;
    } a1;
    a1.xys.x = 71; a1.z = 12;
```

Структура **того же самого** типа не может содержаться в качестве поля - рекурсивные определения запрещены. Зато нередко используются поля - **ссылки** на структуры такого же типа (или другого). Это позволяет организовывать **списки** структур:

```
    struct node {
            int value;
```

```
        struct node *next;
};
```

Очень часто используются массивы структур:

```
struct XYS array[20]; int i = 5, j;
array[i].x = 12;
j = array[i].x;
```

Статические структуры можно описывать с инициализацией, перечисляя значения их полей в {} через запятую:

```
extern struct node n2;
struct node n1 = { 1, &n2  },
            n2 = { 2, &n1  },
            n3 = { 3, NULL };
```

В этом примере **n2** описано предварительно для того, чтобы &**n2** в строке инициализации **n1** было определено.

Структуры одинакового типа можно присваивать целиком (что соответствует присваиванию каждого из полей):

```
struct XYS s1, s2;  ...
s2 = s1;
```

в отличие от массивов, которые присваивать целиком нельзя:

```
int a[5], b[5];  a = b; /* ОШИБОЧНО ! */
```

Пример обращения к полям структуры:

```
typedef struct _Point {
    short x, y; /* координаты точки */
    char *s;    /* метка точки      */
} Point;
Point p; Point *pptr; short *iptr;
struct _Curve {
  Point points[25]; /* вершины ломанной */
  int color;        /* цвет линии       */
} aLine[10], *linePtr = & aLine[0];
        ...
pptr = &p; /* указатель на структуру p */
p.x = 1; p.y = 2; p.s = "Grue";
linePtr->points[2].x  = 54; aLine[5].points[0].y  = 17;
```

В ы р а ж е н и е				значение
p.x	pptr->x	(*pptr).x	(&p)->x	1
			&p->x	ошибка
iptr= &p.x	iptr= &pptr->x	iptr= &(pptr->x)		адрес поля
*pptr->s	*(pptr->s)	*p.s	p.s[0]	'G'
pptr->s[1]	(&p)->s[1]	p.s[1]		'r'
&p->s[1]				ошибка
(*pptr).s	pptr->s	p.s		"Grue"
*pptr.s				ошибка

```
        Вообще (&p)->field   =  p.field
               pptr->field   =  (*pptr).field
```

Объединения – это агрегаты данных, которые могут хранить в себе значения данных разных типов **на одном и том же месте.**

```
        struct a{ int x, y; char *s; } A;
        union  b{ int i; char *s; struct a aa; } B;

        Структура:

  _____________________
A: | A.x        int    |   Три поля
   ----------------------   расположены подряд.
   | A.y        int    |   Получается как бы
   ----------------------   "карточка" с графами.
   | A.s        char * |
   ----------------------
A у объединений поля расположены "параллельно",
на одном месте в памяти.

  ____________________________________________
B: | B.i  int | B.s  char *  | B.aa     : B.aa.x  int    |
   -----------|              | struct a : B.aa.y  int    |
              --------------|          : B.aa.s  char * |
                            |__________________________|
```

Это как бы "ящик" в который можно поместить значение любого типа из перечисленных, но не ВСЕ ВМЕСТЕ ("и то и это", как у структур), а ПО ОЧЕРЕДИ ("или/или"). Размер его достаточно велик, чтоб вместить самый большой из перечисленных типов данных.

Мы можем занести в *union* значение и интерпретировать его как **другой** тип данных это иногда используется в машинно-зависимых программах. Вот пример, выясняющий порядок байтов в *short* числах:

```
 union lb {
        char s[2]; short i;
 } x;
 unsigned hi, lo;
 x.i = (02 << 8) | 01;
 hi = x.s[1]; lo = x.s[0];
 printf( "%d %d\n", hi, lo);
```

или так:

```
#include <stdio.h>
union {
        int i;
        unsigned char s[sizeof(int)];
} u;
void main(){
        unsigned char *p;
        int n;
        u.i = 0x12345678;
        for(n=0, p=u.s; n < sizeof(int); n++, p++){
                printf("%02X ", *p);
        }
        putchar('\n');
}
```

или порядок слов в *long* числах:

```
union xx {
  long l;
  struct ab {
    short a;  /* low word  */
    short b;  /* high word */
  } ab;
} c;
main(){ /* На IBM PC 80386 печатает 00020001 */
```

```
    c.ab.a = 1; c.ab.b = 2; printf("%08lx\n", c.l );
}
```

5.1.

Найдите ошибки в описании структурного шаблона:

```
structure { int arr[12],
            char string,
            int *sum
          }
```

5.2.

Разработайте структурный шаблон, который содержал бы название месяца, трехбуквенную аббревиатуру месяца, количество дней в месяце и номер месяца. Инициализируйте его для невисокосного года.

```
struct month {
       char name[10];  /* или char *name; */
       char abbrev[4]; /* или char *abbrev; */
       int  days;
       int  num;
};
struct month months[12] = {           /* индекс */
       {"Январь" , "Янв", 31, 1 },    /*  0     */
       {"Февраль", "Фев", 28, 2 },    /*  1     */
              ...
       {"Декабрь", "Дек", 31, 12},    /* 11     */
}, *mptr = & months[0];  /* или *mptr = months */
main(){
      struct month *mptr;
      printf( "%s\n", mptr[1].name );
      printf( "%s %d\n", mptr->name, mptr->num );
}
```

Напишите функцию, сохраняющую массив **months** в файл; функцию, считывающую его из файла. Используйте *fprintf* и *fscanf*.

В чем будет разница в функции чтения, когда поле **name** описано как char **name**[10] и как char ***name**?

Ответ: во втором случае для сохранения прочитанной строки надо заказывать память динамически при помощи *malloc*() и сохранять в ней строку при помощи *strcpy*(), т.к. память для хранения самой строки в структуре не зарезервирована (а только для указателя на нее).

Найдите ошибку в операторах функции *main*(). Почему печатается не "Февраль", а какой-то мусор? Указание: куда указывает указатель **mptr**, описанный в *main*()? Ответ: в "неизвестно куда" - это локальная переменная (причем не получившая начального значения - в ней содержится мусор), а не то же самое, что указатель **mptr**, описанный выше! Уберите описание **mptr** из *main*.

Заметим, что для распечатки всех или нескольких полей структуры следует ЯВНО перечислить в *printf*() все нужные поля и указать форматы, соответствующие типам этих полей. Не существует формата или стандартной функции, позволяющей распечатать все поля сразу (однако такая функция может быть написана вами для конкретного типа структур). Также не существует формата для *scanf*(), который вводил бы структуру целиком. Вводить можно только по частям - каждое поле отдельно.

5.3.

Напишите программу, которая по номеру месяца возвращает общее число дней года вплоть до этого месяца.

5.4.

Переделайте предыдущую программу таким образом, чтобы она по написанному буквами названию месяца возвращала общее число дней года вплоть до этого месяца. В программе используйте функцию *strcmp*().

5.5.

Переделайте предыдущую программу таким образом, чтобы она запрашивала у пользователя день, месяц, год и выдавала общее количество дней в году вплоть до данного дня. Месяц может обозначаться номером, названием месяца или его аббревиатурой.

5.6.

Составьте структуру для учетной картотеки служащего, которая содержала бы следующие сведения: фамилию, имя, отчество; год рождения; домашний адрес; место работы, должность; зарплату; дату поступления на работу.

5.7.

Что печатает программа?

```
struct man {
        char name[20];
        int salary;
} workers[] = {
        { "Иванов",  200 },
        { "Петров",  180 },
        { "Сидоров", 150 }
}, *wptr, chief = { "начальник", 550 };
main(){
   struct man *ptr, *cptr, save;
   ptr = wptr = workers + 1;
   cptr = &chief;
   save = workers[2]; workers[2] = *wptr; *wptr = save;
   wptr++; ptr--; ptr->salary = save.salary;
   printf( "%c %s %s %s %s\n%d %d %d %d\n%d %d %c\n",
     *workers[1].name, workers[2].name, cptr->name,
       ptr[1].name, save.name,
         wptr->salary, chief.salary,
           (*ptr).salary, workers->salary,
     wptr - ptr, wptr - workers, *ptr->name );
}
```

Ответ:

```
С Петров начальник Сидоров Сидоров
180 550 150 150
2 2 И
```

5.8.

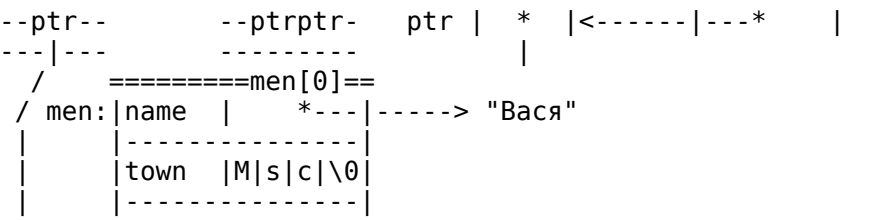
Разберите следующий пример:

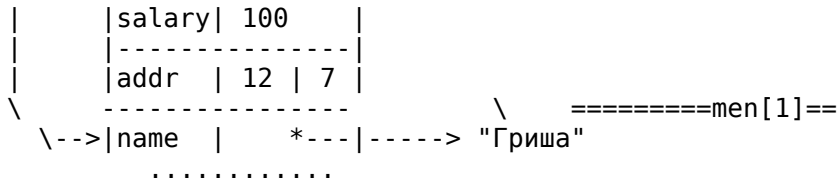
```
#include <stdio.h>
struct man{
        char *name, town[4]; int salary;
        int addr[2];
} men[] = {
        { "Вася",  "Msc",     100,  { 12,  7 } },
        { "Гриша", "Len",     120,  { 6,  51 } },
        { "Петя",  "Rig",     140,  { 23, 84 } },
        { NULL,    ""   ,      -1,  { -1, -1 } }
};
main(){
   struct man *ptr, **ptrptr;
   int i;
   ptrptr = &ptr;
   *ptrptr = &men[1];     /* men+1 */
   printf( "%s    %d    %s   %d   %c\n",
            ptr->name,
                  ptr->salary,
                        ptr->town,
                             ptr->addr[1],
                                  ptr[1].town[2] );
   (*ptrptr)++;
   /* копируем *ptr в men[0] */
   men[0].name    = ptr->name;       /* (char *) #1 */
   strcpy( men[0].town, ptr->town ); /* char []  #2 */
   men[0].salary  =     ptr->salary; /* int      #3 */
   for( i=0; i < 2; i++ )
       men[0].addr[i] = ptr->addr[i];  /* массив #4 */
   /* распечатываем массив структур */
   for(ptr=men; ptr->name; ptr++ )
       printf( "%s %s %d\n",
                ptr->name, ptr->town, ptr->addr[0]);
}
```

Обратите внимание на такие моменты:

1. Как производится работа с указателем на указатель (**ptrptr**).
2. При копировании структур отдельными полями, поля скалярных типов (int, char, long, ..., указатели) копируются операцией присваивания (см. строки с пометками #1 и #3). Поля векторных типов (массивы) копируются при помощи цикла, поэлементно пересылающего массив (строка #4). Строки (массивы букв) пересылаются стандартной функцией *strcpy* (строка #2). Все это относится не только к полям структур, но и к переменным таких типов. Структуры можно также копировать не по полям, а целиком: **men**[0]= ***ptr;**
3. Запись аргументов функции *printf*() лесенкой позволяет лучше видеть, какому формату соответствует каждый аргумент.
4. При распечатке массива структур мы печатаем не определенное их количество (равное размеру массива), а пользуемся указателем *NULL* в поле **name** последней структуры как признаком конца массива.
5. В поле **town** мы храним строки из 3х букв, однако выделяем для хранения массив из 4х байт. Это необходимо потому, что строка "Msc" состоит не из 3х, а из 4х байтов: 'M','s','c','\0'.

При работе со структурами и указателями большую помощь могут оказать рисунки. Вот как (например) можно нарисовать данные из этого примера (массив **men** изображен не весь):





5.9.

Составьте программу "справочник по таблице Менделеева", которая по названию химического элемента выдавала бы его характеристики. Таблицу инициализируйте массивом структур.

Назад | Содержание | Вперед

[Главная] [Гостевая]