Структуры, объединения и нестандартные типы данных

Многие задачи удобно и экономно программируются с применением составных конструкций данных языка C, которые называются *структурами*. Структура представляет собой нестандартный объект для хранения разнородных данных, создаваемый программистом для своих целей. В этой связи будут рассмотрены следующие вопросы.

- Простые и сложные структуры
- Объявление и определение структур
- Обращение к данным в структурах
- Создание структур, содержащих массивы, и массивов структур
- Объявление указателей внутри структур и на структуры
- Передача структур в качестве аргументов функций
- Объявление, определение и использование объединений
- Определение новых структурных типов

Простейшие структуры

Новый термин

Структура — это совокупность нескольких переменных под единым именем, рассматриваемых как одно целое. В отличие от массива, переменные в структуре могут иметь различные типы. Переменные внутри структуры называются ее *членами* или *полями*. Структуры могут содержать данные любых мыслимых типов C, в том числе массивы и другие структуры. В следующем разделе приведен простой пример.

В языке C допускаются структуры любой сложности, но мы начнем с самых простых.

Объявление и определение структур

Если вы пишете программу с графикой, она обязательно должна работать с координатами точек экрана. Экранные координаты записываются в виде двух целых чисел: `x` описывает горизонтальное положение точки, а `y` — вертикальное. Для этой цели можно определить структуру под именем `coord`, которая будет содержать сразу и `x`, и `y`:

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

Ключевое слово `struct` указывает начало определения структуры. За этим ключевым словом должно следовать имя структуры. Это же правило относится и к другим нестандартным типам данных, которые создаются программистом. Имя структуры называют также ее *меткой* (*tag*), а также *именем структурного типа*. Позже мы рассмотрим использование этой метки.

После метки структуры должна стоять открывающая фигурная скобка. В скобках после имени структуры находится список переменных — полей структуры. Каждый элемент структуры должен быть объявлен со своим именем и типом.

В приведенном выше примере определяется структурный тип под названием `coord`, содержащий два целочисленных поля `x` и `y`. Однако это объявление структуры и ее членов само по себе не создает фактического объекта-структуры `coord` или ее переменных `x` и `y`. Другими словами, объявление структурного типа — это еще не объявление экземпляров этого типа, т.е. при этом не выделяется память для фактических структур. Объявить фактическую структуру можно двумя способами. Один из них состоит в том, чтобы поставить список имен переменных сразу после определения структурного типа, как показано ниже:

```
struct coord {
    int x;
    int y;
} first, second;
```

В этом операторе определяется структурный тип `coord`, и объявляются две структуры этого типа — `first` и `second`. Эти две структуры являются *экземплярами* типа `coord` — каждая из них содержит две целочисленных переменных `x` и `y`.

При таком способе объявления структур определение структурного типа совмещается с объявлением его экземпляров. Другой способ состоит в том, что объявление экземпляров структур помещается в исходном коде отдельно от определения типа. В следующем примере также объявляются два экземпляра структурного типа `coord`:

```
struct coord {
    int x;
    int y;
};
/* Здесь могут стоять другие операторы */
struct coord first, second;
```

В этом примере определение структурного типа `coord` помещено отдельно от объявления переменных. В отдельно стоящем объявлении первым должно идти ключевое слово `struct`, затем метка структуры, затем имена объявляемых переменных.

Обращение к полям структуры

Отдельные элементы структуры могут использоваться точно так же, как простые переменные тех же типов. Чтобы извлечь их значения из структуры, применяется знак *операции обращения к элементу структуры* (`.`), представляющий собой точку между именем структуры и именем элемента-поля. Таким образом, чтобы структура `first` содержала экранные координаты `x = 50`, `y = 100`, следует записать:

```
first.x = 50;
first.y = 100;
```

Для вывода экранных координат, хранящихся в структуре second, запишем следующее:

```
printf("%d,%d", second.x, second.y);
```

Может возникнуть вопрос, в чем же преимущество структур перед отдельными переменными. Одно большое преимущество заключается в том, что можно копировать информацию из одной структуры в другую простым присваиванием. Продолжая предыдущий пример, запишем следующее:

```
first = second;
```

Этот оператор эквивалентен следующим двум операторам присваивания:

```
first.x = second.x;
first.y = second.y;
```

Если в программе используются сложные структуры с большим количеством элементов, такая запись может сэкономить много времени и места. Остальные преимущества структур станут понятны по мере изучения методов работы с ними. Вообще говоря, структуры полезно применять в тех случаях, когда совокупности разнородных переменных должны восприниматься как единое целое. Например, в базе данных, содержащей адреса, каждая отдельная запись может быть представлена структурой, а отдельные информативные поля такой записи (имя, фамилия, адрес и т.д.) — элементами структуры.

В листинге 11.1 представлен пример для иллюстрации всего сказанного. Эта программа не имеет особой практической ценности, однако иллюстрирует работу с простой структурой.

Листинг 11.1. simple.c — объявление и использование простой структуры

```
 1:  /* simple.c - Демонстрация использования простой структуры */
 2:
 3:  #include <stdio.h>
 4:
 5:  int length, width;
 6:  long area;
 7:
 8:  struct coord{
 9:      int x;
10:      int y;
11:  } myPoint;
12:
13:  int main( void )
14:  {
15:      /* помещение значений в поля координат */
16:      myPoint.x = 12;
17:      myPoint.y = 14;
18:
19:      printf("\nThe coordinates are: (%d, %d).",
20:             myPoint.x, myPoint.y);
21:
22:      return 0;
23:  }
```

Результат

```
The coordinates are: (12, 14).
```

Анализ В этой программе определяется простая структура для хранения координат точки. Примеры с этой структурой уже встречались вам ранее. В строке 8 находится ключевое слово struct и ее метка coord. Тело структуры находится в строках 9–11. Объявлено два поля структуры — целочисленные переменные x и y.

В строке 11 объявляется экземпляр структуры `coord` — переменная `myPoint`. Это же объявление можно было записать в отдельной строке таким образом:

```
struct coord myPoint;
```

В строках 16 и 17 полям структуры `myPoint` присваиваются значения. Как уже упоминалось, для этого используется имя структуры, затем точка и имя поля. В строках 19 и 20 эти же поля выводятся на экран с помощью функции `printf()`.

Синтаксис

Ключевое слово struct

```
struct метка {
элемент(ы)_структуры;
/* дополнительные операторы */
} экземпляр;
```

С помощью ключевого слова `struct` определяются структурные типы и объявляются структуры. Структура — это совокупность одной или нескольких переменных (*элемент(ы)_структуры*) под единым именем для удобства обработки. Переменные не обязаны иметь один и тот же тип, а также не обязательно должны быть простыми переменными. В структуры можно включать массивы, указатели и другие структуры.

Определение структуры начинается с ключевого слова `struct`. Следом идет метка структуры, представляющая собой имя создаваемого структурного типа. Затем в фигурных скобках перечисляются элементы структуры. Можно сразу же объявить и переменную (*экземпляр*) данного структурного типа. Если структура определяется без объявления переменных-экземпляров, то такое определение является всего лишь шаблоном, который можно использовать позже в программе для объявления структурных переменных. Такой шаблон имеет следующий формат:

```
struct метка {
элемент(ы)_структуры;
/* дополнительные операторы */
};
```

Объявление переменных с использованием этого шаблона структуры имеет следующий вид:

```
struct метка экземпляр;
```

Это объявление будет синтаксически правильным только в том случае, если ранее была определена структура с данной меткой.

Пример 1

```
/* Объявление шаблона структуры под именем SSN */
struct SSN {
    int first_three;
          char dash1;
          int second_two;
          char dash2;
          int last_four;
      }
      /* Использование шаблона для объявления */
      struct SSN customer_ssn;
```

Пример 2

```
      /* Совместное объявление структуры и экземпляра */
      struct date {
          char month[2];
```

```
    char day[2];
    char year[4];
} current_date;
```

Пример 3

```
/* Объявление и инициализация структуры */
struct time {
    int hours;
    int minutes;
    int seconds;
} time_of_birth = { 8, 45, 0 };
```

Сложные структуры

Вот вы и получили некоторое представление о простейших структурах в языке С. Настало время перейти к более сложным и интересным структурным типам данных, которые могут включать в себя другие структуры или массивы в качестве элементов.

Включение структур в структуры

Как уже упоминалось, структуры могут содержать элементы любых типов данных С, в том числе и других структурных типов. Проиллюстрируем это, доработав наш предыдущий пример.

Предположим, нашей графической программе необходимо работать с прямоугольниками. Прямоугольник можно задать координатами его противоположных (по диагонали) углов. Структура для хранения координат одной точки уже объявлялась ранее. Для задания прямоугольника необходимы две такие структуры, объединенные в единое целое. Это можно сделать следующим образом (при условии, что структурный тип `coord` уже определен):

```
struct rectangle {
    struct coord topleft;
    struct coord bottomrt;
};
```

В этом операторе определяется структурный тип `rectangle`, включающий две структуры типа `coord`: `topleft` и `bottomrt`. Это дает нам только новый тип; необходимо еще объявить экземпляр структуры:

```
struct rectangle mybox;
```

Как и ранее для структуры `coord`, определение типа и объявление его экземпляров можно совместить:

```
struct rectangle {
    struct coord topleft;
    struct coord bottomrt;
} mybox;
```

Для обращения к полям, где хранятся фактические числовые данные, необходимо два раза применить точку:

```
mybox.topleft.x
```

Это выражение представляет собой обращение к полю `x` поля `topleft` структуры типа `rectangle`, имеющей имя `mybox`. Для задания прямоугольника с координатами вершин `(0, 10)`, `(100, 200)` необходимо записать следующее:

```
mybox.topleft.x = 0;
mybox.topleft.y = 10;
mybox.bottomrt.x = 100;
mybox.bottomrt.y = 200;
```

Все это может показаться довольно запутанным. Для лучшего понимания взгляните на рис. 11.1, на котором показаны соотношения между структурой типа `rectangle`, двумя содержащимися в ней структурами типа `coord` и отдельными полями типа `int`, включенными в каждую из структур `coord`. Имена структур совпадают с именами из последнего приведенного примера.

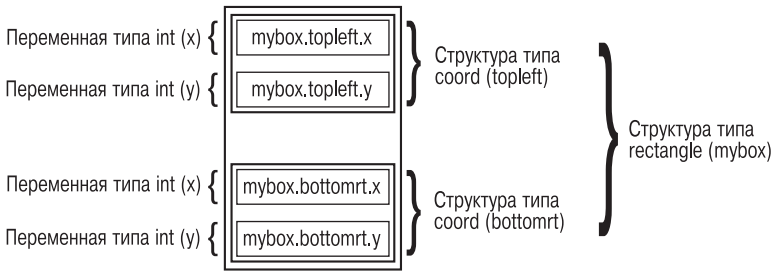

Рис. 11.1. Соотношения между структурой, ее полями структурных типов и простыми полями во вложенных структурах

В листинге 11.2 представлен пример работы со структурами, которые содержат другие структуры. Эта программа запрашивает у пользователя координаты прямоугольника, а затем вычисляет и выводит на экран его площадь. Обратите внимание на предположения, сделанные в программе. Они приведены в комментариях в начале исходного кода (строки 3–8).

Листинг 11.2. `struct.c` — демонстрация структур-членов других структур

```
 1:  /* Демонстрация структур, содержащих другие структуры. */
 2:
 3:  /* Принимает координаты вершин прямоугольника и вычисляет
 4:     его площадь. Предполагается, что координата y нижнего
 5:     правого угла больше, чем координата y верхнего левого
 6:     угла, и что координата x нижнего правого угла больше,
 7:     чем координата x верхнего левого угла. Все координаты
 8:     считаются положительными. */
 9:
10:  #include <stdio.h>
11:
12:  int length, width;
13:  long area;
14:
15:  struct coord{
16:      int x;
17:      int y;
18:  };
19:
20:  struct rectangle{
21:      struct coord topleft;
22:      struct coord bottomrt;
23:  } mybox;
24:
25:  int main( void )
```

```
26:  {
27:      /* Ввод координат */
28:
29:      printf("\nEnter the top left x coordinate: ");
30:      scanf("%d", &mybox.topleft.x);
31:
32:      printf("\nEnter the top left y coordinate: ");
33:      scanf("%d", &mybox.topleft.y);
34:
35:      printf("\nEnter the bottom right x coordinate: ");
36:      scanf("%d", &mybox.bottomrt.x);
37:
38:      printf("\nEnter the bottom right y coordinate: ");
39:      scanf("%d", &mybox.bottomrt.y);
40:
41:      /* Вычисление длины и ширины прямоугольника */
42:
43:      width = mybox.bottomrt.x - mybox.topleft.x;
44:      length = mybox.bottomrt.y - mybox.topleft.y;
45:
46:      /* Вычисление и вывод площади */
47:
48:      area = width * length;
49:      printf("\nThe area is %ld units.\n", area);
50:
51:      return 0;
52:  }
```

Результат

```
Enter the top left x coordinate: 1

Enter the top left y coordinate: 1

Enter the bottom right x coordinate: 10

Enter the bottom right y coordinate: 10

The area is 81 units.
```

Анализ Структурный тип `coord` определяется в строках 15–18. Его полями являются две целочисленные переменные `x` и `y`. В строках 20–23 определяется структура `rectangle`, и объявляется ее экземпляр `mybox`. Структура `rectangle` включает два поля, `topleft` и `bottomrt`, являющиеся структурами типа `coord`.

В строках 29–39 структура `mybox` заполняется значениями. На первый взгляд может показаться, что значений всего два, поскольку в `mybox` два элемента. Но это не так, потому что каждый из элементов `mybox` сам является структурой и содержит свои собственные поля: в `topleft` и `bottomrt` по два элемента, `x` и `y`, из структуры `coord`. Всего получается четыре элемента, которые нужно заполнить значениями. После заполнения вычисляется площадь. Для этого используются имена структуры и ее полей. Чтобы воспользоваться значениями `x` и `y`, необходимо знать имена экземпляров структур. Поскольку `x` и `y` находятся в структуре внутри другой структуры, в вычислениях используются имена экземпляров на двух уровнях структурной вложенности: `mybox.bottomrt.x`, `mybox.bottomrt.y`, `mybox.topleft.x` и `mybox.topleft.y`.

В языке C нет ограничений на глубину вложенности структур, хотя стандарт ANSI гарантирует поддержку не более 63 уровней. Пока позволяет память, можно определять структуры, содержащие структуры со структурами структур и т.д. — идея понятная, однако по до-

стижении определенного предела совершенно непродуктивная. На практике в программах на C почти никогда не используется больше трех уровней вложенности структур.

Структуры, содержащие массивы

Можно определять структуры, содержащие массивы в качестве своих элементов (полей) . Массивы могут быть любых типов (`int`, `char` и т.д.). Например, в следующем фрагменте кода определяется структурный тип `data`, содержащий целочисленный массив `x` из четырех элементов и символьный массив `y` из десяти элементов:

```
struct data
{
    int x[4];
    char y[10];
};
```

Затем можно объявить переменную `record` этого структурного типа:

```
struct data record;
```

Организация этой структуры показана на рис. 11.2. Заметьте, что на рисунке элементы массива `x` занимают вдвое больше места в памяти, чем элементы массива `y`. Это происходит потому, что данные типа `int` обычно занимают два байта памяти, тогда как данные типа `char` — только один (что обсуждалось на занятии 3, посвященном хранению данных в переменных и константах).

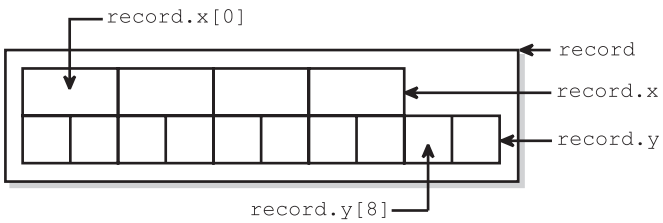

Рис. 11.2. Организация структуры с массивами в качестве элементов

Чтобы обратиться к отдельному элементу массива, являющегося полем структуры, применяется комбинация точки и индекса массива:

```
record.x[2] = 100;
record.y[1] = 'x';
```

Вы наверняка помните, что массивы символов чаще всего применяются для хранения строк. Следует также помнить (из материала занятия 9), что имя массива без квадратных скобок представляет собой указатель на этот массив. То же самое справедливо и для массивов — полей структур. Рассмотрим следующее выражение:

```
record.y
```

В силу сказанного, это указатель на первый элемент массива `y[]` в структуре `record`. Поэтому содержимое массива `y[]` можно вывести на экран следующим оператором:

```
puts(record.y);
```

Рассмотрим еще один пример. В листинге 11.3 используется структура, состоящая из переменной типа `float` и двух символьных массивов.

Листинг 11.3. array.c — структура с массивами в качестве элементов

```
 1:  /* Демонстрация структуры с массивами в качестве элементов. */
 2:
 3:  #include <stdio.h>
 4:
 5:  /* Определение и объявление структуры данных. */
 6:  /* Содержит одно вещественное поле и два символьных массива. */
 7:
 8:  struct data{
 9:      float amount;
10:      char fname[30];
11:      char lname[30];
12:  } rec;
13:
14:  int main( void )
15:  {
16:      /* Ввод данных с клавиатуры. */
17:
18:      printf("Enter the donor's first and last names,\n");
19:      printf("separated by a space: ");
20:      scanf("%s %s", rec.fname, rec.lname);
21:
22:      printf("\nEnter the donation amount: ");
23:      scanf("%f", &rec.amount);
24:
25:      /* Вывод данных. */
26:      /* Примечание: спецификация %.2f задает вывод */
27:      /* вещественного числа с двумя цифрами после */
28:      /* десятичной точки. */
29:
30:      /* Вывод данных на экран. */
31:
32:      printf("\nDonor %s %s gave $%.2f.\n", rec.fname, rec.lname,
33:              rec.amount);
34:
35:      return 0;
36:  }
```

Результат

```
Enter the donor's first and last names,
separated by a space: Bradley Jones

Enter the donation amount: 1000.00
Donor Bradley Jones gave $1000.00.
```

Анализ Эта программа использует структуру с двумя массивами под именами `fname[30]` и `lname[30]`. Оба эти массива — символьные и предназначены соответственно для хранения имени и фамилии человека. Структурный тип, объявленный в строках 8–12, имеет метку `data`. В качестве полей он содержит символьные массивы `fname` и `lname`, а также вещественную переменную `amount` типа `float`. Эта структура прекрасно подходит для хранения таких данных, как имя лица (точнее, имя и фамилия) в сочетании с некоторым относящимся к нему числом, например, суммой взноса в благотворительную организацию.

В строке 12 также объявляется экземпляр структуры под названием `rec`. Остальная часть программы использует структуру `rec` для получения данных от пользователя (строки 18–23) и вывода их на экран (строки 32 и 33).

Массивы структур

Если в структурах могут содержаться массивы, то нельзя ли хранить структуры в массивах? Оказывается, можно. Фактически, массивы структур являются мощнейшим средством программирования. Далее мы рассмотрим, как это средство работает.

Вы уже знаете, как с помощью структурных типов можно организовать данные в точном соответствии с потребностями вашей программы. Как правило, программе недостаточно всего одного экземпляра данных структурного типа. Пусть, например, программа предназначена для ведения телефонной книги. Можно определить структуру для хранения номера телефона каждого абонента:

```
struct entry
{
    char fname[10];
    char lname[12];
    char phone[8];
};
```

Но в телефонной книге обычно хранится много номеров, так что от одного экземпляра такой структуры немного толку. Поэтому необходим массив структур типа `entry`. После определения структуры массив объявляется следующим образом:

```
struct entry list[1000];
```

Этот оператор объявляет массив с именем `list` из тысячи элементов. Каждый элемент представляет собой структуру типа `entry` и распознается по индексу, как и в любом другом массиве. В каждой из структур содержится три поля, причем эти поля — массивы типа `char`. Вся получившаяся сложная конструкция данных показана схематически на рис. 11.3.

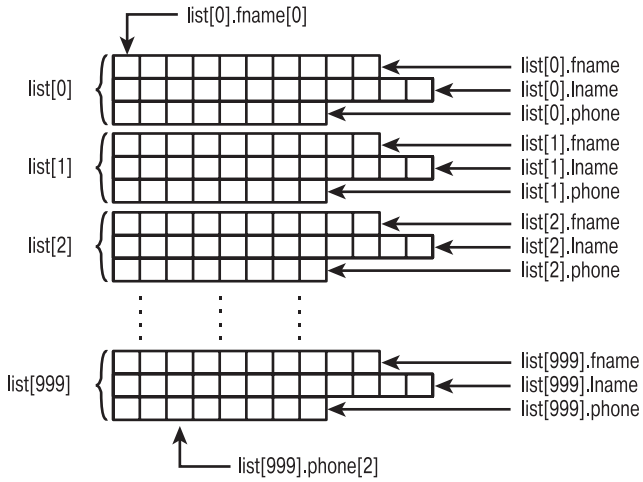

Рис. 11.3. Организация массива структур

После объявления массива структур можно использовать его для разнообразных задач обработки данных. Например, для присвоения данных из одного элемента массива другому можно записать следующее:

```
list[1] = list[5];
```

Этот оператор присваивает всем полям структуры `list[1]` значения соответствующих полей структуры `list[5]`. Можно также перемещать данные между отдельными полями структур. Например, следующий оператор копирует строку `list[5].phone` в строку `list[1].phone`:

```
strcpy(list[1].phone, list[5].phone);
```

(Здесь для копирования строк используется библиотечная функция `strcpy()`. Она будет подробно рассмотрена на занятии 17.) При желании можно даже обмениваться данными между отдельными элементами массивов, являющихся полями структур:

```
list[5].phone[1] = list[2].phone[3];
```

Этот оператор копирует четвертый символ номера телефона из записи `list[2]` во вторую позицию номера в записи `list[5]`. (Не забывайте, что индексы массивов начинаются с нуля, а не с единицы.)

В листинге 11.4 приведен пример использования массивов структур. Более того, структуры в этих массивах сами содержат массивы в качестве своих полей.

Листинг 11.4. `strucarr.c` — массивы структур

```
 1:  /* Демонстрация работы с массивами структур. */
 2:
 3:  #include <stdio.h>
 4:
 5:  /* Определение структуры отдельных элементов массива. */
 6:
 7:  struct entry {
 8:      char fname[20];
 9:      char lname[20];
10:      char phone[10];
11:  };
12:
13:  /* Объявление массива структур. */
14:
15:  struct entry list[4];
16:
17:  int i;
18:
19:  int main( void )
20:  {
21:
22:      /* Цикл ввода данных о четырех лицах. */
23:
24:      for (i = 0; i < 4; i++)
25:      {
26:          printf("\nEnter first name: ");
27:          scanf("%s", list[i].fname);
28:          printf("Enter last name: ");
29:          scanf("%s", list[i].lname);
30:          printf("Enter phone in 123-4567 format: ");
31:          scanf("%s", list[i].phone);
32:      }
33:
34:      /* Вывод двух пустых строк. */
35:
36:      printf("\n\n");
```

```
37:
38:      /* Цикл для вывода данных. */
39:
40:      for (i = 0; i < 4; i++)
41:      {
42:           printf("Name: %s %s", list[i].fname, list[i].lname);
43:           printf("\t\tPhone: %s\n", list[i].phone);
44:      }
45:
46:      return 0;
47:  }
```

Результат

```
Enter first name: Bradley
Enter last name: Jones
Enter phone in 123-4567 format: 555-1212

Enter first name: Peter
Enter last name: Aitken
Enter phone in 123-4567 format: 555-3434

Enter first name: Melissa
Enter last name: Jones
Enter phone in 123-4567 format: 555-1212

Enter first name: Kyle
Enter last name: Rinni
Enter phone in 123-4567 format: 555-1234

Name: Bradley Jones             Phone: 555-1212
Name: Peter Aitken              Phone: 555-3434
Name: Melissa Jones             Phone: 555-1212
Name: Kyle Rinni                Phone: 555-1234
```

Анализ Эта программа следует хорошо знакомой вам форме наших примеров. Она начинается с комментария в строке 1 и включения заголовочного файла средств ввода-вывода `stdio.h` в строке 3. В строках 7–11 определяется шаблон структуры `entry`, состоящей из трех символьных массивов: `fname`, `lname` и `phone`. В строке 15 с помощью этого шаблона объявляется массив из четырех структур типа `entry` под именем `list`. В строке 17 объявляется переменная типа `int`, которая используется как счетчик на протяжении всей программы. Функция `main()` начинается в строке 19. Первое, что она делает — это выполняет цикл `for` из четырех повторений для ввода данных в массив структур в строках 24–32. Заметьте, что обращение к массиву выполняется по индексу — точно так же, как и в случае простых массивов, рассмотренных на занятии 8.

В строке 36 на экран стандартным способом выводятся две пустые строки, чтобы отделить введенные данные от выводимых результатов. Строки 40–44 выполняют вывод на экран данных, введенных пользователем ранее. Чтобы вывести значения отдельных полей структур, используются индексы и точка.

Внимательно ознакомьтесь с приемами, использованными в листинге 11.4. Массивы структур, каждая из которых в свою очередь содержит массивы, — это важный инструмент программирования практически важных задач.

Рекомендуется	Не рекомендуется
Используйте ключевое слово `struct` при объявлении экземпляра структуры, определенной ранее. **Объявляйте** экземпляры структур с той же областью действия, что и обычные переменные. (См. занятие 12.)	**Не забывайте** о точке и имени экземпляра при обращении к полям структуры. **Не путайте** метку структурного типа с именем экземпляра структуры! Метка используется только для определения шаблона, или формата, структуры. Фактические данные хранятся в экземпляре, объявленном с помощью метки.

Инициализация структур

Как и переменные других типов, структуры можно инициализировать при объявлении. Это делается аналогично инициализации массивов: после объявления структуры ставится знак равенства и список начальных значений, разделенных запятыми и заключенных в фигурные скобки. Рассмотрим следующий пример:

```
1: struct sale {
2:     char customer[20];
3:     char item[20];
4:     float amount;
5: } mysale = {
6:              "Acme Industries",
7:              "Left-handed widget",
8:              1000.00
9:            };
```

При выполнении этих операторов происходит следующее:

1. Определяется структура с меткой `sale` (строки 1–5).
2. Объявляется экземпляр структурного типа `sale` с именем `mysale` (строка 5).
3. Поле структуры `mysale.customer` инициализируется литералом `"Acme Industries"` (строки 5 и 6).
4. Поле структуры `mysale.item` инициализируется литералом `"Left-handed widget"` (строка 7).
5. Поле структуры `mysale.amount` инициализируется значением `1000.00` (строка 8).

Если структура в свою очередь содержит другие структуры в качестве полей, то начальные значения должны строго следовать порядку как структур, так и полей в их определениях. Вот пример, являющийся обобщением предыдущего:

```
1:  struct customer {
2:      char firm[20];
3:      char contact[25];
4:  }
5:
6:  struct sale {
7:     struct customer buyer;
8:     char item[20];
9:     float amount;
10: } mysale = { { "Acme Industries", "George Adams" },
11:                "Left-handed widget",
12:                 1000.00
13:            };
```

В этом фрагменте выполняются следующие инициализации:

1. Поле `mysale.buyer.firm` инициализируется литералом `"Acme Industries"` (строка 10).
2. Поле `mysale.buyer.contact` инициализируется литералом `"George Adams"` (строка 10).
3. Поле `mysale.item` инициализируется литералом `"Left-handed widget"` (строка 11).
4. Поле `mysale.amount` инициализируется значением `1000.00` (строка 12).

Можно инициализировать и целые массивы структур. Заданные начальные значения помещаются в структуры массива в строгом порядке. Например, в следующем фрагменте кода объявляется массив структур типа `sale` и инициализируются два его первых элемента (т.е. две первые структуры):

```
1:  struct customer {
2:      char firm[20];
3:      char contact[25];
4:  };
5:
6:  struct sale {
7:      struct customer buyer;
8:      char item[20];
9:      float amount;
10: };
11:
12:
13: struct sale y1990[100] = {
14:      { { "Acme Industries", "George Adams" },
15:          "Left-handed widget",
16:           1000.00
17:      }
18:      { { "Wilson & Co." , "Ed Wilson" },
19:          "Type 12 gizmo",
20:           290.00
21:      }
22: };
```

Эти операторы работают следующим образом:

1. Поле `y1990[0].buyer.firm` инициализируется литералом `"Acme Industries"` (строка 14).
2. Поле `y1990[0].buyer.contact` инициализируется литералом `"George Adams"` (строка 14).
3. Поле `y1990[0].item` инициализируется литералом `"Left-handed widget"` (строка 15).
4. Поле `y1990[0].amount` инициализируется значением `1000.00` (строка 16).
5. Поле `y1990[1].buyer.firm` инициализируется литералом `"Wilson & Co."` (строка 18).
6. Поле `y1990[1].buyer.contact` инициализируется литералом `"Ed Wilson"` (строка 18).
7. Поле `y1990[1].item` инициализируется литералом `"Type 12 gizmo"` (строка 19).
8. Поле `y1990[1].amount` инициализируется значением `290.00` (строка 20).

Структуры и указатели

Учитывая важность указателей в языке C, совсем не удивительно, что их можно эффективно использовать для работы со структурами. Во-первых, указатели могут быть элементами структур, а во-вторых, можно объявлять и указатели на структуры. Давайте рассмотрим эти вопросы поподробнее в следующих разделах.

Указатели как поля структур

Программист имеет полную свободу в использовании указателей как элементов структур. Указатели-поля структур объявляются точно так же, как и любые указатели, не являющиеся членами структур, т.е. с помощью звездочки:

```
struct data
{
    int *value;
    int *rate;
} first;
```

В этом примере определяется и создается структура, в которой оба элемента являются указателями на `int`. Как вы уже знаете, объявить указатели недостаточно — их необходимо еще инициализировать. Это можно сделать, присвоив им подходящие адреса переменных. Если есть две переменные типа `int` с именами `cost` и `interest`, то можно записать:

```
first.value = &cost;
first.rate = &interest;
```

Теперь указатели получили конкретные значения, и для обращения к ним можно использовать операцию ссылки (*), изученную на занятии 9. Выражение `*first.value` равно значению переменной `cost`, а выражение `*first.rate` — значению переменной `interest`.

Наверное, самым распространенным указателем, используемым в структурах, можно считать указатель на `char`. Вспомните материал занятия 10: последовательность символов, однозначно определенная указателем на ее начало и нулевым символом на конце, называется *строкой*. Для повторения пройденного объявим указатель на строку символов и инициализируем его:

```
char *p_message;
p_message = "Teach Yourself C In 21 Days";
```

То же самое можно проделать и с указателями на `char`, являющимися полями структур:

```
struct msg {
    char *p1;
    char *p2;
} myptrs;
myptrs.p1 = "Teach Yourself C In 21 Days";
myptrs.p2 = "By SAMS Publishing";
```

На рис. 11.4 показан результат выполнения этих операторов. Каждый из указателей в структурах указывает на первый байт строки, помещенной в памяти где-то в другом месте. Сравните этот способ с рис. 11.3, где показано, как данные хранятся в структурах, содержащих массивы символов.

Указатели, входящие в структуры, можно использовать везде, где вообще допускается применение указателей. Например, чтобы вывести на экран содержимое строки, пишем следующее:

```
printf("%s %s", myptrs.p1, myptrs.p2);
```

В чем разница между использованием массива типа `char` и указателя того же типа в качестве элементов структуры? В сущности, оба эти способа предназначены для “помещения” строк в структуры, как показано ниже. Здесь в структуре `msg` используются оба способа:

```
struct msg
{
    char p1[30];
    char *p2; /* осторожно: не инициализирован! */
} myptrs;
```

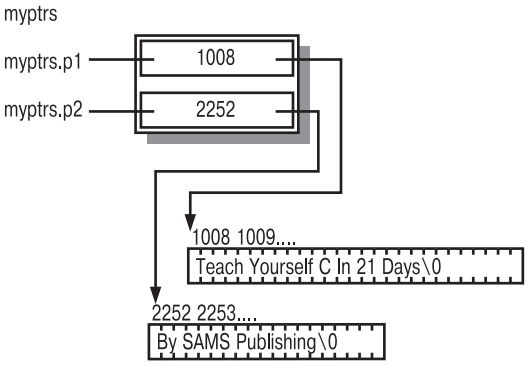

Рис. 11.4. Структура, содержащая указатели типа `char`

Вспомните: имя массива без квадратных скобок само по себе является указателем на первый элемент массива. Поэтому к двум приведенным элементам структуры можно обращаться одинаковым образом (не забудьте инициализировать `p2`, прежде чем помещать что-либо по хранящемуся в нем адресу):

```
strcpy(myptrs.p1, "Teach Yourself C In 21 Days");
strcpy(myptrs.p2, "By SAMS Publishing");
/* здесь идет дополнительный код */
puts(myptrs.p1);
puts(myptrs.p2);
```

В чем же разница между этими двумя способами? А вот в чем: если определить структуру с массивом типа `char`, то каждый экземпляр такой структуры будет непосредственно содержать участок памяти нужной длины для хранения этого массива. Кроме того, этот размер никак нельзя превысить, поместив в массив строку большего размера. Вот пример:

```
struct msg
{
    char p1[10];
    char p2[10];
} myptrs;
...
strcpy(p1, "Minneapolis"); /* Ошибка! Строка длиннее массива. */
strcpy(p2, "MN");          /* Нет ошибки, но напрасная трата памяти, */
                           /* поскольку строка короче массива. */
```

Если же определить структуру, имеющую в своем составе указатель на строку, то эти ограничения отпадут. В каждом экземпляре структуры выделяется место лишь для указателя, а сами строки находятся в других местах (где именно — это не ваша забота). Ни напрасной траты памяти, ни ограничения на длину строки не будет, поскольку сами строки не хранятся непосредственно в структуре. Каждый указатель может указывать на строку любой длины, и строка фактически становится частью структуры, хотя и не помещена в память в едином блоке с ней.

Если не инициализировать указатель, то можно случайно затереть информацию в памяти, предназначенную для чего-то важного. Поэтому, используя указатели вместо массивов, выполняйте их корректную инициализацию. Это можно проделать с помощью динамического выделения памяти или ассоциируя указатели с адресами реальных переменных.

Создание указателей на структуры

В языке C можно объявлять и использовать указатели на структуры — точно так же, как указатели любых других типов. Как будет показано далее, указатели на структуры часто используются при передаче структур в функции через аргументы. Еще указатели на структуры применяются в чрезвычайно мощном методе хранения и обработки данных под названием *связанные списки*. Этот метод подробно разбирается на занятии 15, посвященном дополнительным возможностям указателей.

Сейчас мы обсудим, как же объявить и использовать указатель на структуру в программе на C. Вначале определим структуру:

```
struct part
{
    short number;
    char name[10];
};
```

Теперь объявим указатель на переменную типа `part`:

```
struct part *p_part;
```

Напомним, что знак ссылки (`*`) в объявлении показывает, что `p_part` является указателем на значение типа `part`, а не экземпляром этого структурного типа.

Можно ли теперь инициализировать этот указатель? Увы, нет. Хотя структурный тип `part` уже определен, пока что не создано ни одного экземпляра этого типа. Помните, что не определение структурного типа, а объявление переменной выделяет место в памяти для объекта данных. Указатель должен содержать вполне определенный адрес объекта. Поэтому необходимо вначале объявить хотя бы один экземпляр типа `part`, чтобы указатель смог указывать на что-то реально существующее. Вот это объявление:

```
struct part gizmo;
```

Вот сейчас уже можно инициализировать указатель:

```
p_part = &gizmo;
```

Этот оператор присваивает адрес структуры `gizmo` указателю `p_part`. (Вспомните операцию взятия адреса, рассмотренную на занятии 9.) На рис. 11.5 показано соотношение между структурой и указателем на структуру.

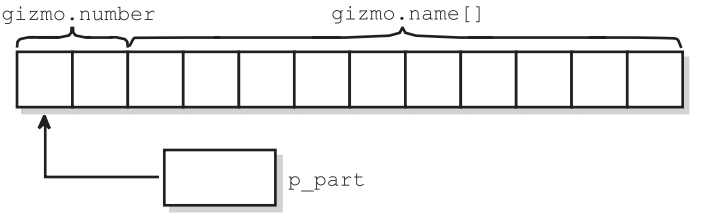

Рис. 11.5. Указатель на структуру: указывает на первый байт хранимых данных

Итак, у нас есть указатель на структуру `gizmo`. Как же обращаться к структуре, используя этот указатель? Один из способов — воспользоваться операцией ссылки по указателю (`*`). Напомним материал занятия 9: если `ptr` — указатель на элемент данных, то выражение `*ptr` обозначает само значение этого элемента.

Применим эти знания к данному примеру. Если `p_ptr` — указатель на структуру `gizmo`, то `*p_ptr` — это сама структура. К отдельным полям структуры можно обращаться с помощью точки (`.`). Например, чтобы присвоить значение `100` полю `gizmo.number`, запишем следующее:

```
(*p_part).number = 100;
```

Выражение *p_part необходимо заключить в скобки, поскольку операция обращения к элементу структуры (.) имеет более высокий приоритет, чем операция ссылки по указателю (*).

Есть еще один способ работы с полями структуры через указатели. Он предполагает использование *косвенного обращения к элементам структуры*. Знак этой операции состоит из двух символов -> (дефиса и знака "больше"). Когда эти символы стоят рядом без пробела между ними, компилятор воспринимает их как один знак операции, а не два. Знак косвенного обращения ставится между именем указателя и именем поля. Например, для обращения к полю number структуры gizmo через указатель p_part можно записать следующее:

```
p_part->number
```

Соответственно, если str — структура, p_str — указатель на эту структуру, а memb — поле структуры str, то запись str.memb имеет то же значение, что и следующая:

```
p_str->memb
```

Итак, есть три способа обращения к полю структуры.

- Через имя структуры
- Через указатель на структуру и ссылку по указателю (*)
- Через указатель на структуру и косвенное обращение к элементам структуры (->)

Пусть p_str указывает на структуру str. Тогда следующие три выражения эквивалентны:

```
str.memb
(*p_str).memb
p_str->memb
```

Косвенное обращение к элементу структуры называют еще *обращением к структуре по указателю*.

Указатели и массивы структур

Указатели на структуры и массивы структур являются весьма мощными средствами программирования. Оба этих средства можно комбинировать, используя указатели для обращения к структурам — элементам массивов.

Для примера возьмем определение структуры из ранее приведенных фрагментов кода:

```
struct part
{
    short number;
    char name[10];
};
```

Определив структурный тип part, можно объявить массив структур этого типа:

```
struct part data[100];
```

Далее объявим указатель на структуру типа part и присвоим ему адрес первого элемента в массиве data:

```
struct part *p_part;
p_part = &data[0];
```

Напоминаем, что имя массива без квадратных скобок является указателем на первый элемент массива, поэтому вторую строку можно записать еще и так:

```
p_part = data;
```

Теперь у нас есть массив структур типа part и указатель на первый элемент массива (т.е. первую структуру в массиве). Можно, например, вывести на экран содержимое этого первого элемента:

```
printf("%d %s", p_part->number, p_part->name);
```

Что если необходимо вывести все элементы массива? Вероятно, понадобится цикл `for`, в котором за один проход будет выводиться один элемент. Для обращения к элементам через указатель необходимо изменять `p_part` таким образом, чтобы он указывал на следующую структуру в массиве после каждого прохода цикла. Как же это сделать?

На помощь нам приходит адресная арифметика языка C. Одноместная операция инкрементирования (приращения) в применении к указателям имеет особый смысл. Она означает "увеличить указатель на размер объекта, на который он указывает". Другими словами, пусть у нас есть указатель `ptr` на объект данных типа `obj`. Тогда следующие два оператора эквивалентны:

```
ptr++;
ptr += sizeof(obj);
```

Этот аспект адресной арифметики особенно важен для работы с массивами, поскольку элементы массивов располагаются в памяти последовательно один за другим. Если указатель указывает на элемент массива `n`, то после его инкрементирования (операция `++`) он будет указывать на элемент `n+1`. Иллюстрацией этому служит рис. 11.6, на котором показан массив `x[]`, состоящий из 4-байтных элементов (например, структур, содержащих по два двухбайтных элемента типа `short`). Указателю `ptr` присвоено значение адреса `x[0]`. Всякий раз после инкрементирования `ptr` он указывает на следующий элемент массива.

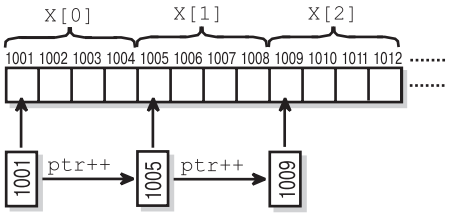

Рис. 11.6. Движение указателя вдоль массива структур

Все это означает, что программа может перебирать массив структур (или если уж на то пошло, массив вообще любых данных) путем инкрементирования указателя. Данный способ работы с массивом обычно дает более удобную и краткую запись, чем обращение по индексу. В листинге 11.5 приведен пример такой записи.

Листинг 11.5. `access.c` — обращение к последовательным элементам массива с помощью указателя

```
 1:  /* Демонстрация перебора массива структур */
 2:  /* с помощью указателей. */
 3:
 4:  #include <stdio.h>
 5:
 6:  #define MAX 4
 7:
 8:  /* Объявление структуры, затем объявление и инициализация */
 9:  /* массива из четырех структур. */
10:
11:  struct part {
12:      short number;
13:      char name[10];
14:  } data[MAX] = {1, "Smith",
15:                 2, "Jones",
```

```
16:                 3, "Adams",
17:                 4, "Wilson"
18:                 };
19:
20:  /* Объявление указателя на тип part и переменной-счетчика. */
21:
22:  struct part *p_part;
23:  int count;
24:
25:  int main( void )
26:  {
27:      /* Установка указателя на первый элемент массива. */
28:
29:      p_part = data;
30:
31:      /* Перебор массива с приращением указателя */
32:      /* на каждом проходе цикла. */
33:
34:      for (count = 0; count < MAX; count++)
35:      {
36:          printf("At address %d: %d %s\n", p_part, p_part->number,
37:                  p_part->name);
38:          p_part++;
39:      }
40:
41:      return 0;
42:  }
```

Результат

```
At address 4202504: 1 Smith
At address 4202516: 2 Jones
At address 4202528: 3 Adams
At address 4202540: 4 Wilson
```

Анализ Вначале в строках 11–18 объявляется и инициализируется массив с именем `data` структур типа `part`. В строке 22 объявляется указатель `p_part` для указания на массив структур `data`. Первое, что выполняет функция `main()`, это устанавливает указатель `p_part` на начало объявленного массива структур. После этого в строках 34–39 в цикле `for` выводятся все элементы массива. Для перебора элементов используется указатель, получающий приращение на каждом проходе цикла. Программа также отображает адрес каждого элемента.

Посмотрите внимательно на выведенные программой адреса. В вашей конкретной системе их значения могут отличаться, но расстояние между ними будет одинаковым: в точности размер структуры-элемента массива. Это демонстрирует со всей наглядностью, что инкрементирование указателя дает ему приращение, равное размеру объекта данных в массиве.

Передача структур в функции

Как и данные других типов, структуры можно передавать в функции в качестве аргументов. Листинг 11.6 демонстрирует, как это делается. Приведенная в нем программа является модификацией программы из листинга 11.3. В ней используется функция, которая выводит на экран данные из переданной в нее структуры, в то время как в программе листинга 11.3 эта же задача выполняется непосредственно в функции `main()`.

Листинг 11.6. `func.c` — передача структуры в функцию

```
1:  /* Демонстрация передачи структуры в функцию. */
2:
```

```
 3:  #include <stdio.h>
 4:
 5:  /* Определение и объявление структуры data. */
 6:
 7:  struct data {
 8:      float amount;
 9:      char fname[30];
10:      char lname[30];
11:  } rec;
12:
13:  /* Прототип функции. Функция не возвращает никакого значения. */
14:  /* Единственный аргумент - структура типа data. */
15:
16:  void print_rec(struct data displayRec);
17:
18:  int main( void )
19:  {
20:      /* Ввод данных с клавиатуры. */
21:
22:      printf("Enter the donor's first and last names,\n");
23:      printf("separated by a space: ");
24:      scanf("%s %s", rec.fname, rec.lname);
25:
26:      printf("\nEnter the donation amount: ");
27:      scanf("%f", &rec.amount);
28:
29:      /* Call the display function. */
30:      print_rec( rec );
31:
32:      return 0;
33:  }
34:  void print_rec(struct data displayRec)
35:  {
36:      printf("\nDonor %s %s gave $%.2f.\n", displayRec.fname,
37:              displayRec.lname, displayRec.amount);
38:  }
```

Результат

```
Enter the donor 's first and last names,
separated by a space: Bradley Jones

Enter the donation amount: 1000.00

Donor Bradley Jones gave $1000.00.
```

Анализ В строке 16 находится прототип функции, принимающей структуру в качестве аргумента. Как полагается, в прототипе объявлен список ее параметров — в данном случае это структура типа `data`. Прототип совпадает с заголовком функции в строке 34. При вызове функции достаточно передать в нее имя экземпляра структуры, т.е. в данном случае `rec` (строка 30). Вот, собственно, и все. Передача структуры в функцию ничем не отличается от передачи простой переменной.

В функцию можно передавать и адрес структуры (т.е. указатель на нее) вместо самой структуры. Фактически, в старых версиях C это был единственный способ передачи структур в функции. Сейчас в этом нет необходимости, но вам могут встретиться старые программы, которые все еще применяют этот способ. Если в функцию передается указатель на структуру, то для обращения к отдельным полям структуры в теле функции необходимо использовать косвенное обращение (`->`).

Рекомендуется	Не рекомендуется
Пользуйтесь преимуществами, которые предоставляют указатели на структуры — особенно при работе с массивами структур. **Используйте** косвенное обращение к элементам (`->`) при работе с указателями на структуры.	**Не путайте** массивы со структурами. **Не забывайте**, что при инкрементировании указателя он сдвигается на размер элемента данных. В случае указателя на структуру это размер структуры.

Объединения

Объединения (*unions*) внешне похожи на структуры. Объединение объявляется и используется аналогичным образом. В отличие от структуры, всякий раз только один из элементов объединения доступен для обращения. Причина этого проста — все элементы объединения занимают одну и ту же область памяти, как бы располагаясь один поверх другого.

Определение, создание и инициализация объединений

Объединения создаются и объявляются аналогично структурам. Единственным различием при их объявлении является использование ключевого слова `union` вместо `struct`. Для примера объявим простейшее объединение, состоящее из одной целой и одной символьной переменной:

```
union shared
{
    char c;
    int i;
};
```

Определение этого объединения, `shared`, можно дальше использовать для создания его экземпляров, которые могут содержать или целочисленное значение `i`, или символьное `c`. Тут очень важно понять условие “или – или”. В отличие от структуры, в которой сразу хранились бы оба значения, объединение может содержать только одно из них. На рис. 11.7 показано, как это объединение размещается в памяти.

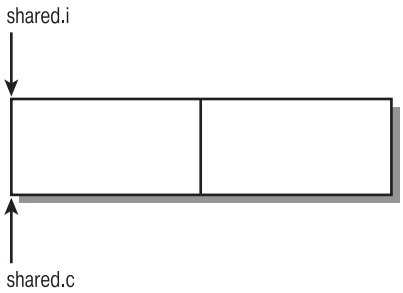

Рис. 11.7. Способ размещения объединения в памяти

При объявлении объединения его можно инициализировать. Раз уж только один элемент объединения может использоваться при каждом обращении к нему, то и инициализировать нужно только один элемент. Во избежание путаницы инициализируется первый элемент. Вот пример объявления и инициализации объединения типа `shared`:

```
union shared generic_variable = { '@' };
```

Обратите внимание, что объединение инициализируется точно так же, как первый элемент структуры.

Обращение к элементам объединения

Обращение к отдельным полям объединения выполняется так же, как к полям структуры: с помощью точки (`.`). Однако есть и существенное различие. Обращаться всякий раз можно только к одному элементу объединения, поскольку элементы хранятся один поверх другого. В листинге 11.7 представлен пример.

Листинг 11.7. `union.c` — пример некорректного использования объединения

```
 1:  /* Пример одновременного обращения к нескольким полям объединения */
 2:  #include <stdio.h>
 3:
 4:  int main( void )
 5:  {
 6:      union shared_tag {
 7:          char   c;
 8:          int    i;
 9:          long   l;
10:          float  f;
11:          double d;
12:      } shared;
13:
14:      shared.c = '$';
15:
16:      printf("\nchar c   = %c",  shared.c);
17:      printf("\nint i    = %d",  shared.i);
18:      printf("\nlong l   = %ld", shared.l);
19:      printf("\nfloat f  = %f",  shared.f);
20:      printf("\ndouble d = %f",  shared.d);
21:
22:      shared.d = 123456789.8765;
23:
24:      printf("\n\nchar c   = %c",  shared.c);
25:      printf("\nint i    = %d",  shared.i);
26:      printf("\nlong l   = %ld", shared.l);
27:      printf("\nfloat f  = %f",  shared.f);
28:      printf("\ndouble d = %f\n",  shared.d);
29:
30:      return 0;
31:  }
```

Результат

```
char c   = $
int i    = 65572
long l   = 65572
float f  = 0.000000
double d = 0.000000
```

```
char c   = 7
int i    = 1468107063
long l   = 1468107063
float f  = 284852666499072.000000
double d = 123456789.876500
```

Анализ В строках 6–12 этой программы объявляется и создается объединение `shared`. Объединение содержит пять элементов, все различных типов. В строках 14 и 22 выполняется инициализация отдельных элементов объединения `shared`. Затем в строках 16–20 и 24–28 выполняется вывод всех элементов с помощью функции `printf()`.

Обратите внимание, что за исключением значений `char c = '$'` и `double d = 123456789.876500`, результаты на экране вашего компьютера могут отличаться от приведенных. Раз в строке 14 было инициализировано символьное поле объединения `c`, то и использоваться должно только оно — до тех пор, пока не будет инициализировано следующее. Результаты вывода остальных элементов (`i`, `l`, `f` и `d`) в строках 16–20 непредсказуемы. В строке 22 значение помещается уже в вещественное поле `d`. Теперь результат вывода всех остальных полей, кроме `d`, становится непредсказуемым. Помещенное в `c` значение (строка 14) теряется, поскольку его затирает значение `d`, присвоенное в строке 22. Все это показывает, что поля объединения действительно занимают одну и ту же область памяти.

Синтаксис

Ключевое слово union

```
union метка {
    член(ы)_объединения;
    /* здесь могут стоять дополнительные операторы */
} экземпляр;
```

Ключевое слово `union` используется для объявления объединений. Объединение представляет собой совокупность одной или нескольких переменных (*член(ы)_объединения*) под одним общим именем. Кроме того, все члены объединения занимают один и тот же участок в памяти.

Ключевое слово `union` указывает начало определения объединения. За ним следом идет метка типа объединения. Затем в фигурных скобках стоят члены объединения. Можно также объявить *экземпляр*, т.е. фактический объект этого типа. Если определить объединение, не создавая экземпляра, это определение будет всего лишь шаблоном, который можно использовать далее в программе для объявления экземпляров объединений. Такой шаблон имеет следующий синтаксис:

```
union метка {
    член(ы)_объединения;
    /* здесь могут стоять дополнительные операторы */
};
```

Объявление объединения с помощью шаблона выглядит следующим образом:

```
union метка экземпляр;
```

Конечно, объединение с этой меткой должно было быть определено заранее.

Пример 1

```
/* Объявление шаблона объединения tag */
union tag {
    int nbr;
    char character;
}
```

```
/* Использование шаблона в объявлении экземпляра */
union tag mixed_variable;
```

Пример 2

```
/* Совместное объявление типа и экземпляра */
union generic_type_tag {
    char c;
    int i;
    float f;
    double d;
} generic;
```

Пример 3

```
/* Инициализация объединения. */
union date_tag {
    char full_date[9];
    struct part_date_tag {
        char month[2];
        char break_value1;
        char day[2];
        char break_value2;
        char year[2];
    } part_date;
} date = { "01/01/97" };
```

В листинге 11.8 демонстрируется часто встречающееся на практике, хотя и несколько упрощенное, применение объединений.

Листинг 11.8. `union2.c` **— практическое применение объединений**

```
 1:  /* Типичный пример применения объединений */
 2:
 3:  #include <stdio.h>
 4:
 5:  #define CHARACTER   'C'
 6:  #define INTEGER     'I'
 7:  #define FLOAT       'F'
 8:
 9:  struct generic_tag{
10:      char type;
11:      union shared_tag {
12:          char   c;
13:          int    i;
14:          float  f;
15:      } shared;
16:  };
17:
18:  void print_function( struct generic_tag generic );
19:
20:  int main( void )
21:  {
22:      struct generic_tag var;
23:
24:      var.type = CHARACTER;
25:      var.shared.c = '$';
26:      print_function( var );
27:
```

```
28:      var.type = FLOAT;
29:      var.shared.f = (float) 12345.67890;
30:      print_function( var );
31:
32:      var.type = 'x';
33:      var.shared.i = 111;
34:      print_function( var );
35:      return 0;
36:  }
37:  void print_function( struct generic_tag generic )
38:  {
39:      printf("\n\nThe generic value is...");
40:      switch( generic.type )
41:      {
42:          case CHARACTER: printf("%c",  generic.shared.c);
43:                          break;
44:          case INTEGER:   printf("%d",  generic.shared.i);
45:                          break;
46:          case FLOAT:     printf("%f",  generic.shared.f);
47:                          break;
48:          default:        printf("an unknown type: %c\n",
49:                                  generic.type);
50:                          break;
51:      }
52:  }
```

Результат

```
The generic value is...$
The generic value is...12345.678711
The generic value is...an unknown type: x
```

Анализ Эта программа представляет весьма упрощенную версию того, что можно делать с помощью объединений, а именно способ поочередного хранения данных разных типов в одном участке памяти. Структура `generic_tag` позволяет помещать в одну область памяти либо символ, либо целое число, либо вещественное число. Эта область памяти представлена объединением `shared`, устроенным точно так же, как в листинге 11.7. В структуре `generic_tag` имеется дополнительное поле под именем `type`. Это поле используется для хранения информации о типе переменной, помещенной в объединение `shared`. Поле типа препятствует некорректному обращению к объединению `shared`, таким образом помогая избежать ошибочного вывода данных, показанного в листинге 11.7.

Рассмотрим эту программу подробнее. В строках 5, 6 и 7 объявляются константы `CHARACTER`, `INTEGER` и `FLOAT`. Они используются далее в программе для удобства обозначения типов. В строках 9–16 определяется структурный тип `generic_tag`, который будет использоваться в объявлениях чуть позже. В строке 18 представлен прототип функции `print_function()`. В строке 22 объявлена структура `var`, которая затем инициализируется символьными значениями в строках 24–25. Вызов функции `print_function()` в строке 26 выводит значение на экран. В строках 28–30 и 32–34 эта процедура повторяется с другими значениями.

Центральной частью программы является функция `print_function()`. Здесь она используется для вывода значений из структуры типа `generic_tag`. Аналогичную функцию можно было бы применять и для присваивания структуре значений. Функция `print_function()` анализирует переменную `type` для вывода корректного значения из объединения с подобающим сопроводительным текстом. Благодаря этому не происходит ошибочного вывода данных, показанного в листинге 11.7.

Рекомендуется	Не рекомендуется
Помните, какой из членов объединения используется вами в данный момент. Если вы поместите значение в одно поле и попытаетесь воспользоваться другим, то получите непредсказуемый результат.	**Не инициализируйте** никакие поля объединений, кроме первого. **Не забывайте**, что размер объединения равен размеру его наибольшего элемента.

Создание структурных типов с помощью typedef

С помощью ключевого слова `typedef` создается синоним структурного типа или метки объединения. Например, в следующем объявлении создается синоним `coord` для некоторого структурного типа:

```
typedef struct {
    int x;
    int y;
} coord;
```

Далее можно объявить экземпляры этого типа, пользуясь его идентификатором:

```
coord topleft, bottomright;
```

Обратите внимание, что созданный таким образом тип отличается от метки структуры. В следующей записи идентификатор `coord` является меткой структуры:

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

С помощью этой метки можно объявлять экземпляры данного структурного типа, но в отличие от определенных через `typedef` типов, для этого придется писать ключевое слово `struct`:

```
struct coord topleft, bottomright;
```

Выбор между `typedef` и меткой структуры является делом вкуса. Большой практической разницы тут нет. Определение типа через `typedef` несколько короче, потому что не надо писать ключевого слова `struct`. С другой стороны, использование этого слова и метки структурного типа ясно показывает, что объявляется именно структура, и это удобно для программиста при чтении кода.

Резюме

На этом занятии мы рассмотрели использование структур — составных типов данных, проектируемых специально для нужд конкретной программы. Структура может содержать элементы любых типов данных C, в том числе другие структуры, массивы и указатели. Обращение к отдельному элементу данных внутри структуры, называемому также членом или полем, производится с помощью знака точки (`.`) между именем структуры и именем поля. Структуры могут использоваться по отдельности или объединяться в массивы.

Объединения имеют сходство со структурами. Основное различие между структурами и объединениями состоит в том, что в объединении все поля хранятся в одной и той же области памяти, накладываясь друг на друга. В результате элементами объединения можно пользоваться только по очереди, а не вместе, как в структуре.

Вопросы и ответы

Есть ли смысл определять структуру без объявления ее экземпляров?

Сегодня были рассмотрены два способа объявления структур. Один из них — это одновременное объявление метки, тела и экземпляров структурного типа. Второй способ — это объявление метки и тела без экземпляров. Экземпляры структурных переменных можно объявить потом, записав ключевое слово `struct`, метку типа и имя экземпляра. Наиболее распространенным на практике является именно второй способ. Многие программисты объявляют структурные типы отдельно, а экземпляры этих типов — позже по мере необходимости. На следующем занятии будет рассмотрена область действия переменных, и в этой связи важно знать, что область действия имеет значение для экземпляров структур, но не для их типов.

Какой способ объявления структурного типа более распространен: с помощью `typedef` или метки структуры?

Многие программисты используют `typedef` для удобочитаемости программ, но на самом деле большой практической разницы здесь нет. На рынке программного обеспечения предлагается много библиотек дополнительных функций. Обычно в них применяется много определений типов с помощью `typedef`, чтобы сделать их код оригинальным. Особенно это справедливо в отношении программных продуктов для работы с базами данных.

Можно ли попросту скопировать одну структуру в другую с помощью оператора присваивания?

И да, и нет. Новейшие версии компиляторов C позволяют это делать, а вот более старые — нет. Работая со старым компилятором, придется копировать каждый элемент структуры отдельно. Это относится и к объединениям.

Как вычислить размер объединения?

Все члены объединения располагаются в одной области памяти, накладываясь друг на друга. Поэтому объем памяти, необходимый для размещения объединения, равен размеру его самого длинного элемента.

Коллоквиум

В этом коллоквиуме вам предлагаются контрольные вопросы для закрепления пройденного материала, а также упражнения для выработки практических навыков программирования.

Контрольные вопросы

1. В чем разница между структурой и массивом?
2. Зачем нужна операция обращения к элементу структуры и каким знаком она обозначается?
3. С помощью какого ключевого слова объявляется структура?
4. В чем разница между меткой структурного типа и экземпляром структуры?
5. Что делает следующий фрагмент кода?

```
struct address
{
    char name[31];
    char add1[31];
    char add2[31];
    char city[11];
    char state[3];
    char zip[11];
} myaddress = { "Bradley Jones",
                "RTSoftware",
                "P.O.Box 1213",
                "Carmel", "IN", "46082-1213"};
```

6. Если вы создали тип под названием `word` с помощью ключевого слова `typedef`, то как вы объявите переменную `myWord` этого типа?
7. Предположим, вы объявили массив структур и указатель `ptr`, который указывает на его первый элемент (т.е. первую структуру в массиве). Как изменить `ptr` так, чтобы он указывал на второй элемент массива?

Упражнения

1. Напишите определение структуры `time`, содержащей три поля типа `int`.
2. Напишите фрагмент кода, выполняющий две задачи: во-первых, определение структуры `data`, состоящей из одной переменной типа `int` и двух переменных типа `float`, во-вторых, объявление экземпляра типа `data` с именем `info`.
3. Продолжая упражнение 2, присвойте значение `100` целочисленному полю структуры `info`.
4. Напишите операторы объявления и инициализации указателя на структуру `info`.
5. Продолжая упражнение 4, присвойте значение 5.5 первому из вещественных (`float`) полей структуры `info` с использованием указателя, причем двумя способами.
6. Напишите определение структурного типа с именем `data`, в котором может храниться строка длиной до 20 символов.
7. Объявите структуру, содержащую пять строк: `address1`, `address2`, `city`, `state`, `zip`. Определите тип `RECORD` с помощью ключевого слова `typedef`, чтобы можно было объявлять структуры этого типа.
8. Используя определение типа из упражнения 7, объявите и инициализируйте переменную `myaddress`.
9. **Поиск ошибок**. Найдите ошибки в следующем коде:

```
struct {
    char zodiac_sign[21];
    int month;
} sign = "Leo" , 8;
```

10. **Поиск ошибок**. Найдите ошибки в следующем коде:

```
/* объявление объединения */
union data {
    char a_word[4];
    long a_number;
} generic_variable = { "WOW", 1000 };
```