Лекции 5, 6. Практика 3. Структуры
Недели семестра – 5, 6

3 Структуры и объединения
- 3.1 Структуры
- 3.2 Объединения
- 3.3 Переменные структуры
- 3.4 Указатели и структуры

3 СТРУКТУРЫ И ОБЪЕДИНЕНИЯ

Цель изучения данного модуля – освоить работу с такими производными типами в Си, как структуры и объединения.

3.1 СТРУКТУРЫ

Структура – это совокупность переменных, объединенных под одним именем. Переменные, составляющие структуру, могут быть разных типов. Эти переменные называются элементами (или полями, членами, компонентами) структуры. Обычно элементы структуры связаны друг с другом по смыслу. Структура – это нестандартный, производный тип данных, который пользователь формирует из некоторых типов данных. Для объявления типа структуры в Си используется ключевое слово struct.

Структуры – одно из базовых понятий в современном программировании. Основные области применения структур: базы данных; объектно-ориентированное программирование (ООП) – понятие «класс» в ООП тесно связано с понятием «структура»; динамические структуры данных, например - односвязные списки. В терминологии языка Паскаль и в некоторых других случаях структуры называют записями.

Структуры позволяют размещать в смежных полях связанные между собой элементы информации. Чтобы понять смысл объединения некоторых компонент в одну структуру, рассмотрим следующий пример.

Пусть имеем семестровую экзаменационную ведомость (список), каждая строка которой содержит: 1) ФИО студента; 2) результаты сдачи экзаменов для этого студента. Например:

Арбузов Ф.В. 5 4 5 4

Батищев И.С. 3 3 2 3

Петров П.Р. 5 3 5 4

Необходимо организовать обработку списка, например, определить средний балл по всей группе, по отдельному студенту, по определенной дисциплине, составить отдельный список отличников и т.д.

Первый вопрос, который должен решить программист еще до написания программы, это то, как организовать хранение данных. Можно предложить 2 подхода.

1. Работать с двумя отдельными массивами: массив ФИО и двумерный массив оценок. В этом случае, чтобы найти, например, оценки определенного студента, надо вначале в первом массиве найти его ФИО и зафиксировать номер этого студента в списке ФИО. Затем во втором массиве надо найти оценки студента, соответствующие этому номеру. Такая работа будет не удобной и длительной по времени, особенно в случае, если отдельных полей информации не 2, как в нашем примере, а много, например, 200. Плохо, что связанные общим смыслом элементы информации по отдельному студенту оказываются «разбросанными» в памяти.

2. Было бы очень удобно сразу сгруппировать в одну совокупность (в одну переменную-структуру) всю информацию, относящуюся к конкретному студенту. Это можно сделать с помощью объявления структуры следующего вида:

```
struct  student  {          // student – метка (тег, имя) структуры
    char fio[20];   // ФИО студента
    int oz[4];        // Оценки студента
};
```

Обратите внимание, что объявление структуры завершается знаком «;», потому что объявление структуры является оператором.

В данном случае объявляется только тип данных «struct student», никакая переменная при этом не создается. Чтобы объявить переменную (то есть физический объект, который занимает место в памяти) типа «struct student», напишем, например:

```
struct  student  chel,grup[30];
```

Можно было сразу же при объявлении типа структуры объявить и переменные этого типа, то есть сразу записать:

```
struct  student  {
    char fio[20];
    int oz[4];
```

} chel, grup[30];

И в том, и в другом случае мы объявляем одиночную переменную chel типа «struct student» и массив grup из 30 элементов, каждый из которых имеет тип «struct student». Переменная chel состоит из 2 компонентов: fio и oz; каждый элемент массива grup также состоит из 2 компонентов: fio и oz.

При работе со структурами имена структур и отдельных полей должны быть читабельны и должны отражать смысл соответствующих объектов. В коротких лекционных примерах использовать в этих случаях имена типа «a», «в» допустимо, но в реальном программировании и в лабораторных работах имена должны быть осмысленные.

3.1.1 Определение структуры

Существует 3 основных способа (варианта) определения структур в программе. Если провести небольшую модификацию 3-го способа, то можно получить еще и 4-ый способ.

1 способ. Краткий способ без определения имени для типа структуры.

```
struct {
   список описаний
} описатели;
```

где *список описаний* - это описание компонентов структуры (должен быть указан хотя бы один компонент);

описатели - это обычно имена переменных, массивов, указателей и функций.

Пример 1. Переменные a и b определяются как структуры, каждая из которых состоит из двух компонентов x и y. Переменная c определяется как массив из 9 таких структур.

```
struct {
   double x,y;
} a,b,c[9];
```

Пример 2. Каждая из двух переменных-структур date1 и date2 состоит из 3 компонент year, month, day.

```
struct {
   int year;
   short month,day;
} date1,date2;
```

2 способ. Полный способ с использованием ключевого слова typedef: определяется новое имя для типа структуры.

В этом случае с помощью ключевого слова typedef явно определяется имя типа структуры, а затем это имя используется для определения переменных.

Общий вид определения типа структуры:

```
typedef struct {
   список описаний
} имя типа;
```

Пример 3. Вначале объявлен тип структуры с именем empl, состоящей из 2 компонентов name и d. Затем объявлены переменные e1, e2 как структуры типа empl.

```
typedef struct {
   char name[30];
   int d;
} empl;          // Объявлен тип структуры с именем empl
empl e1,e2;    // Объявлены переменные e1,e2 как структуры типа empl
```

3 способ. Полный способ с определением метки (тега) для типа структуры.

Способ основан на применении меток (или шаблонов, тегов) структуры (аналогично меткам перечисляемого типа).

Метка структуры объявляется следующим образом:

struct *метка* {

список описаний

};

где *метка* - это идентификатор.

Переменные-структуры определяются с помощью меток структур следующим образом:

struct *метка список идентификаторов*;

В данном случае конструкция «struct *метка*» является, по сути, именем типа.

Пример 4. Вначале объявлена метка структуры student, состоящей из 4 компонентов name, id, age, pol. Затем объявлены переменные s1, s2 как структуры, тип которых определен меткой student. Затем объявлен массив studgrup из 30 элементов, каждый из которых имеет тип структуры с меткой student.

```
struct  student  {       // student - метка структуры
    char name[23];
    int id,age;
    char pol;
};
struct student s1,s2;  // s1,s2 - это структуры, тип которых задан меткой student
struct student studgrup[30];  // studgrup - это массив структур шаблона student
```

4 способ. Полный способ с определением метки для типа структуры и с одновременным объявлением переменных этого типа. Это «усовершенствованный» способ 3.

В большинстве современных системах Си задание шаблона структуры и объявление переменных можно производить одновременно, например:

```
struct  student  {       // student - метка структуры
    char name[23];
```

```
    int id,age;
    char pol;
} s1,s2,studgrup[30];  // переменные и массив типа структуры с меткой student.
```

Рассмотрим рекомендации по использованию различных способов определения структур. Для небольших программ, в которых нужна только переменная-структура, а имя для типа структуры не используется, можно использовать самый краткий способ 1. Но в большинстве реальных случаев необходимо объявлять специальное имя для типа структуры, поэтому надо использовать способы 2 или 3. Это необходимо, например, в случае, если структура является параметром функции, или в случае описания рекурсивных структур. Использование способов 2 и 3 в современных системах Си практически ничем не отличается друг от друга (хотя раньше различия были), так как и в том, и в другом случае формируется имя для типа структуры. В современной практике очень широко используется способ 4 как самый быстрый в написании, но надо иметь в виду, что некоторые Си-компиляторы этот способ не поддерживают.

Замечания по вопросам терминологии. Термин «имя структуры» используется в том случае, если тип структуры объявлен с помощью ключевого слова typedef (способ 2). Для способов 3 и 4 следует использовать термины «метка», «тег», «шаблон». Последние три термина в данном случае – это слова-синонимы, и их наличие связано с тем, что в русскоязычных переводах еще не сформировался один устойчивый термин. Ближе всего к англоязычному исходному термину «tag» транслитерация «тег», но на практике чаще всего используется, пожалуй, слово «метка».

В языке С термины «имя структуры» и «тег» (и соответствующие синонимы) путать нельзя, а в языке С++ они обозначают практически одно и то же, а именно имя типа структуры. Чтобы понять разницу, проанализируем следующий пример. Пусть имеем объявление структуры по способу 3, например:

```
struct  student  {       // student - метка структуры
    char name[23];
    int id; };
```

В языке С для объявления переменных типа «struct student»

необходимо записать, например:

```
struct student s1,s2;
```

А в языке C++ достаточно использовать более короткое объявление:

```
student s1,s2;  // Нормально для C++, но неправильно для C
```

Все вышесказанное можно обобщить для объединений и перечислений. Таким образом, в C при объявлении объектов непосредственно перед тегом имени должно находиться одно из ключевых слов: struct, union или enum (в зависимости от конкретного случая). А в C++ ключевое слово не требуется. Так как для C++ подходят объявления в стиле C, то при переносе программ из C в C++ по этому поводу беспокоится нечего. Но при переносе из C++ в C соответствующие изменения сделать придется. Поэтому для лучшей совместимости рекомендуется, особенно в случае небольших отличий в написании, придерживаться универсального стиля C.

3.1.2 Доступ к компонентам структуры

Все основные действия со структурой проводятся по отдельным полям. Так, вводить и выводить структуру «целиком» нельзя. Ввод и вывод надо организовать по отдельным полям.

Доступ к компонентам структуры осуществляется с помощью операции точка «**.**» и имеет вид:

*имя_структуры-переменной**.**имя_компонента*

Пример. Рассмотрим следующие выражения (они относятся к переменным из примера 4 выше):

s1.id, s2.pol - дают доступ к полю id переменной s1 и к полю pol переменной s2, соответственно;

studgrup[20].age - дает доступ к полю age элемента массива studgrup с индексом 20;

s1.name[5] - дает доступ к элементу с индексом 5 массива name, который является компонентом переменной-структуры s1;

studgrup[20].name[0] - дает доступ к элементу с нулевым индексом поля name для элемента с 20 индексом массива studgrup.

Если объявлены две переменные типа структуры с одним и тем же именем типа или шаблона, то их можно присвоить одна другой, например:

```
s1=s2;
```

Переменные типа структуры нельзя сравнивать с помощью операций сравнения типа "==", "!=" и т.д.

К структуре, как к любой переменной, можно применить операцию & для вычисления ее адреса. К отдельным полям структуры также можно применить операцию &.

Надо иметь в виду, что в большинстве реальных задач используется не просто одиночная структура, а массив структур.

3.1.3 Пример работы со структурой

```
/*  Дан список автомобилей, каждая строка которого содержит: марка автомобиля, год выпуска, цена. Распечатать список тех автомобилей, год выпуска которых не ранее некоторого заданного года, а цена не превышает некоторой заданной цены */

#include <stdio.h>

 int main()

 {

  struct{

   char marka[10]; /* Марка авто  */

   int year;            /* Год выпуска*/

   float money;      /* Цена */

        } avto[20];   /* Массив структур - список данных по автомобилям */

   int   n,       /* Количество элементов в массиве avto */

         i,       /* Индекс массива avto */
```

```
        y;      /* Контрольный год */
  float m;    /* Контрольная цена */
/* Ввод списка */
do {printf("\n Введите количество автомобилей n (n<20): ");
   scanf("%d",&n); while(getchar()!='\n');} while (n<1||n>20);
printf("Введите список из %d автомобилей:\n",n);
for (i=0;i<n;i++) {
        printf("Марка: "); gets(avto[i].marka);
        printf("Год:   ");  scanf("%d",&avto[i].year); while(getchar()!='\n');
        printf("Цена:  ");  scanf("%f",&avto[i].money); while(getchar()!='\n');
}
/* Вывод основного списка */
printf("Основной список (Марка -  Год - Цена)\n");
for (i=0;i<n;i++)
        printf("%12s  %4d  %7.3f\n", avto[i].marka, avto[i].year, avto[i].money);
/* Обработка списка и вывод результата */
int f;          /* Флаг для вывода списка или сообщения «ничего нет»*/
f=1;
printf("\nВведите контрольные год и цену: ");
scanf("%d%f",&y,&m);
printf("Список автомобилей по запросу:\n");
for (i=0;i<n;i++)
        if (avto[i].year>=y&&avto[i].money<=m)
          {printf("%12s  %4d  %7.3f\n", avto[i].marka, avto[i].year, avto[i].money);
           f=0;}
```

```
if(f) printf("\nПо данному запросу ничего нет\n ");
}
```

3.2 ОБЪЕДИНЕНИЯ

Объединение подобно структуре. Объединение – это набор компонентов, возможно разнотипных, объединенных под одним именем, однако в каждый момент времени может использоваться (или являться активным) только один из компонентов.

Как и структуру, объединение можно определить 3 основными способами (смотрите выше пункт 3.1.1), только вместо ключевого слова «struct» надо написать слово «union». Доступ к компонентам объединения осуществляется так же, как и к компонентам структуры, то есть используется операция точка «.».

Так, согласно 1-му способу, объявление объединения имеет вид:

union {

описание компонента 1

описание компонента 2

... ...

описание компонента n

} *описатели*;

Для каждого из этих компонентов 1, 2, ...n выделяется одна и та же область памяти, то есть они перекрываются. Причем места в памяти выделяется ровно столько, сколько надо тому элементу объединения, который имеет наибольший размер в байтах.

Итак, основное отличие структур и объединений состоит в выделении памяти. Объединения иногда (если это возможно по смыслу задачи) позволяют более экономно использовать память, нежели структуры. Дело в том, что при объявлении структуры память выделяется для хранения каждого компонента, а при объявлении объединения память выделяется для хранения только одного компонента. Поэтому для объединения в каждый конкретный

момент времени является активным (реально существующим в памяти) только один компонент, который последним получил значение.

Пример 1. Рассмотрим объявление и выделение памяти для переменной-структуры с именем «a» и для переменной-объединения с именем «b», имеющих одинаковый состав компонентов.

struct {
  int x;
  char s[10];
} a;
	union{
	  int x;
	  char s[10];
	} b;

Для хранения переменной «a» выделяется 14 байтов: 4 байта для компонента «x» и 10 байтов для компонента «s». Значения компонентов «x» и «s» хранятся в памяти одновременно.	Для хранения переменной «b» выделяется 10 байтов, и в этой области в каждый момент времени хранится или значение компонента «x», или значение компонента «s». Выделено 10 байтов, так как память выделяется того объема, который необходим для хранения максимального по размеру компонента.

Пример 2. Переменная geom_fig объявлена как объединение. Эта переменная позволяет работать с различными геометрическими фигурами, но в каждый момент времени можно хранить информацию только об одной фигуре; в данном примере – это окружность.

```
union {
    float radius;   /* Окружность */
    float a[2];     /* Прямоугольник */
    int b[3];       /* Треугольник */
    position p;     /* Точка.  Тип position – это тип, описанный пользователем */
} geom_fig;       /* Геометрическая фигура */
geom_fig.radius=2.5; /* Компоненту, соответствующему окружности, присвоено значение – это активный компонент */
```

Имеет смысл обрабатывать лишь активный компонент, который

последним получил свое значение. Так, после присваивания значения компоненту radius не имеет смысла обращаться к массиву b[3].

Основные области применения объединений:

1) для минимизации объема памяти, если в каждый момент времени только один объект из многих является активным;

2) для интерпретации основного представления объекта одного типа, как если бы этому объекту был присвоен другой тип.

В целях минимизации объема памяти объединения сами по себе используются редко. При решении реальных задач добиться экономии памяти обычно удается в случае, когда объединение являются частью (компонентом) структуры. Такие структуры называются переменными структурами и рассматриваются в следующем подразделе 3.3.

Объединения часто используются тогда, когда нужно выполнить специфическое преобразование типов, потому что хранящиеся в объединении данные можно обозначать совершенно разными способами. То есть объединение обеспечивает доступ к одному и тому же участку памяти с помощью переменных разного типа. Например, необходимо выделить из внутреннего представления целого числа (4-х байтного) его отдельные байты. Для решения этой задачи можно ввести следующее объединение:

```
union{
  int ii;
  char hh[4];
} cc;
```

3.3 ПЕРЕМЕННЫЕ СТРУКТУРЫ

Структуры, примеры которых представлены в подразделе 3.1 - это постоянные структуры с фиксированными компонентами. Они имеют в различных ситуациях строго определенную форму, все компоненты постоянно существуют в памяти, так как для каждого компонента выделена своя область в памяти.

Иногда необходимо, чтобы некоторые компоненты структуры в различных ситуациях были бы различны. Для реализации такой возможности предназначены переменные структуры, которые представляют собой комбинацию структуры и объединения.

Переменная структура содержит набор общих компонентов плюс набор меняющихся компонентов.

В общем случае переменные структуры состоят из трех частей: набора общих компонентов, метки активного компонента и части с меняющимися компонентами. Порядок следования этих трех частей может быть любой, но обычно именно такой, какой приведен здесь.

Общая форма переменной структуры:

```
struct {
  общие компоненты;
  метка активного компонента;
  union {
    описание компонента 1;
    описание компонента 2;
     ... ... ... ... ... ...
    описание компонента n;
   } идентификатор для union;
} описатели;
```

Текущее значение метки активного компонента (например, 1, 2,... n) указывает, какой из переменных компонентов объединения является активным в данный момент. В принципе, метка активного компонента не обязательна, но организовать работу с объединением без нее проблематично. При работе с переменной структурой для проверки метки активного компонента рекомендуется использовать оператор switch.

Пример 1. Рассмотрим представление фигур: окружности, прямоугольника, треугольника. Пусть каждая фигура характеризуется тремя

параметрами: площадь, периметр, размер. Площадь и периметр - это общая информация для этих фигур; информация о размере различна в зависимости от формы фигуры: для круга это - размер радиуса, для прямоугольника - размер двух сторон, для треугольника - размер трех сторон. Рассмотрим соответствующую структуру fig и переменную f типа fig:

```
typedef struct {
  float area, perimetr;  /* Общие компоненты: площадь и периметр */
  int type;                /* Метка активного компонента */
  union {                 /* Переменный компонент */
        float r;              /* радиус окружности */
        float a[2];           /* две стороны прямоугольника */
        float b[3];           /* три стороны треугольника */
         } geomfig;    /* Имя объединения - переменного компонента */
} fig;    /* Имя типа структуры */
fig f;     /* Определение переменной  f  типа  fig  */
```

Каждая переменная типа fig состоит из трех постоянных компонентов: area, perimetr, type. Компонент type (метка активного компонента) используется для указания, какой из компонентов объединения geomfig является активным в данный момент. Например:

type=1 - считаем, что активна первая компонента r;

type=2 - считаем, что активна вторая компонента a[2];

type=3 - считаем, что активна третья компонента b[3].

По соглашению, обычно перед присваиванием значения одному из компонентов структуры соответствующее значение присваивается также переменной f.type для указания активного компонента. Например:

```
f.type=1;   f.geomfig.r=5.0;
```

Аналогично перед обращением к компоненту объединения необходимо проверить, является ли этот компонент активным. Традиционно проверку значения активного компонента организуют с помощью оператора switch. Например:

```
switch (f.type) {
  case 1: <обработка окружности>; break;
  case 2: <обработка прямоугольника>; break;
  case 3: <обработка треугольника>; break;
  default:<ошибка>;
}
```

Метку активного компонента type можно для наглядности описать типа fclass, где fclass - это перечисляемый тип:

typedef enum {circle, rect, triangle} fclass;

В этом случае для работы с треугольником необходимо написать:

f.type=triangle;

Использование перечисляемого типа позволит компилятору предупредить программиста о потенциально ошибочных присваиваниях вида:

f.type=44;

Пример 2. Имеется список граждан. Необходимо сформировать структуру данных для хранения этого списка в памяти.

Структура списка следующая. Для каждого человека указаны следующие компоненты (в скобках дан соответствующий тип данных):

1) ФИО (строка);

2) социальный статус (int):

1 – учащийся,

2 – работающий,

3 – пенсионер;

3) -> 3.1) для учащегося: название учебного заведения (строка);

-> 3.2) для работающего:

а) год рождения (int),

б) код подразделения (int);

-> 3.3) для пенсионера: размер пенсии (float).

В данном случае компоненты 1 и 2 – это общие компоненты, они должны всегда присутствовать для каждого человека. А компонента 3 – это переменная компонента. Для каждого отдельного человека компоненты 3.1, 3.2, 3.3 альтернативны, и хранить в памяти для каждого человека надо только один из этих трех компонентов.

В результате сформируем следующую структуру данных для хранения рассматриваемого списка в памяти.

```
struct anceta{
  char fio[20];            /* Фамилия */
  int status;              /* Статус - метка активного компонента */
  union {                  /* Переменный компонент */
        char name[20];     /* Название учебного заведения для учащегося */
        struct {
           int g;                /* Год рождения работающего*/
           int kod;              /* Код подразделения работающего */
         } rabot;              /* Имя структуры для работающего*/
        float razmer;          /* Размер пенсии  для пенсионера */
  }un;                     /* Имя объединения - переменного компонента */
} spisok[30];    /* Список – массив из 30 структур типа anceta */
```

Внимание! Для данного примера объединение параметров g и kod в одну структуру rabot обязательно. Если эти параметры будут объявлены как два отдельных компонента объединения un, то они будут альтернативны, и одновременно сохранить в памяти всю информацию для работающего – год рождения g и код подразделения kod – будет невозможно.

3.4 УКАЗАТЕЛИ И СТРУКТУРЫ

Указатели на структуры описываются точно так же, как и на другие типы данных.

Указатели на структуры часто используются для создания связных списков и других динамических структур данных, элементами которых, в свою очередь, являются структуры данных.

Пример 1. Предположим, имеются следующие объявления:

```
struct  student {          /*student - метка структуры */
        char name[25];
        int id, age;
        char sex;
};
struct  student  *st;      /*st - это указатель на структуру student */
```

Пусть память уже выделена так, что st указывает на конкретную структуру student. Тогда на компоненты этой структуры можно ссылаться следующим образом:

```
(*st).name
(*st).id
(*st).sex
```

Указатели на структуры так часто используются в Си, что существует специальная операция для ссылки на элементы структуры, адресованной указателем. Эта операция обозначается стрелкой вправо "->". У этой операции - высший приоритет (такой же, как у круглых и квадратных скобок).

Пример 2. Ссылки на компоненты структуры, описанной выше в примере 1, можно записать так:

```
st->name
st->id
st->sex
```

Пример 3. Предположим, имеются следующие объявления:

```
struct  empl {
    char name[80];
    int age;
}em;
struct  empl *p=&em;  // в указателе p - адрес переменной-структуры em
```

Чтобы присвоить элементу age значение 20, можно записать любой из приведенных ниже операторов, но при работе с указателем p профессиональнее использовать последний оператор.

```
em.age=20;
(*p).age=20;
p->age=20;
```