НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»
Кафедра информатики и процессов управления (№17)

Дисциплина «Информатика» (углубленный уровень), 1-й курс, 2-й семестр.

Методические указания

Тематическое занятие 16

Структуры.

Содержание

Основы работы со структурами

Описание структуры

Для работы с *комбинациями данных разных типов* предназначены ***структуры***. Структура состоит из фиксированного числа компонентов одного или нескольких типов, называемых ***полями (элементами, членами)***. Каждое поле структуры имеет свое *имя*.

В общем виде объявление структуры выглядит так:

```
struct МеткаСтруктуры {
    ТипПоля1 ИмяПоля1;
    ТипПоля2 ИмяПоля2;
    ...
    ТипПоляN ИмяПоляN;
} СписокПеременных;
```

Данное объявление, во-первых, описывает форму (шаблон) структуры, т.е. фактически вводит новый тип данных, который в дальнейшем может быть повторно использован с помощью идентификатора `МеткаСтруктуры`, который называется ***метка*** (или ***тег***) структуры. А во-вторых, при этом происходит выделение памяти для всех переменных из `СпискаПеременных`.

При объявлении структуры `МеткаСтруктуры` или `СписокПеременных` могут быть пропущены, но только не оба сразу.

Например, объявление структуры *без списка переменных* выглядит так:

```
struct МеткаСтруктуры {
    ТипПоля1 ИмяПоля1;
    ТипПоля2 ИмяПоля2;
    ...
    ТипПоляN ИмяПоляN;
};
```

Данное объявление только описывает шаблон структуры, однако при этом не происходит выделения памяти для переменных. Память выделяется при объявлении переменной с помощью данного шаблона, для этого используется идентификатор `МеткаСтруктуры`:

```
struct МеткаСтруктуры ИмяПеременной;
```

Другой пример объявления структуры *без метки* выглядит так:

```
struct {
    ТипПоля1 ИмяПоля1;
    ТипПоля2 ИмяПоля2;
    ...
    ТипПоляN ИмяПоляN;
} СписокПеременных;
```

Такое объявление позволяет использовать описанный шаблон структуры только для переменных, содержащихся в `СпискеПеременных`.

Пример объявления структуры

Рассмотрим простой пример объявления структуры. Пусть нам необходимо описать комплексное число ***z = (Re(z), Im(z))***:

```
struct complex {
  double Re;
  double Im;
};
```

Здесь с помощью ключевого слова `struct` создается структура с меткой (тегом) `complex`, состоящая из двух полей `Re` и `Im`, которые в данном случае имеют одинаковый тип `double`. (В общем случае ту же структуру можно описать с разными типами полей `{int Re; double Im;}` или `{float Re; long int Im;}`, но для данного примера это нецелесообразно.)

Поскольку оба поля структуры имеют одинаковый тип, то их имена можно указать в списке:

```
struct complex {
  double Re,Im;
};
```

Однако такого описания следует избегать, а объявлять каждое поле в отдельной строке, для повышения наглядности текста программы.

Структура объявлена, но память для хранения комплексных чисел еще не выделена. Чтобы в программе создать комплексные числа, нужно объявить переменные:

```
struct complex z;
struct complex c1,c2,c3;
```

Теперь созданы четыре переменные – комплексные числа `z`,`c1`,`c2`,`c3`, и для них выделена память.

Те же самые объявления можно сделать и без предварительного описания структуры:

```
struct {double Re; double Im;} z;
struct {double Re; double Im;} c1,c2,c3;
```

Обратите внимание, что при таком непосредственном описании переменных `МеткаСтруктуры` отсутствует, т.к. она является необязательным идентификатором.

В предыдущем примере видно дублирование полей структуры в двух объявлениях. Поэтому структуру можно сначала описать с меткой при объявлении переменных, а затем использовать повторно с помощью метки для объявления других переменных:

```
struct complex {
  double Re;
  double Im;
} z,z1,z2;
struct complex c1,c2,c3,w1,w2;
```

Обращение к полям структуры

Работа с полями структуры происходит с помощью *операции обращения к полю структуры*, которая обозначается точкой:

```
ИмяПеремСтрукт.ИмяПоля
```

Например, задать поля объявленных выше переменных можно с помощью оператора присваивания:

```
z.Re = 12.345;
z.Im = 678;
```

Модуль этого комплексного числа (расстояние между точкой комплексной плоскости, соответствующей этому числу, и началом координат) можно рассчитать с помощью стандартной функции `sqrt()` описанной в `<math.h>`:

```
double modulus;
modulus = sqrt(z.Re*z.Re + z.Im*z.Im);
```

Вывод структуры с помощью стандартной функции `printf()` выполняется поэлементно, т.е. каждое поле выводится отдельно:

```
printf("Re(z)=%lf ; Im(z)=%lf\n", z.Re, z.Im);
```

(Аналогично выполняется ввод с помощью `scanf()`.)

Использование структур

Вложенные структуры

Поля структуры могут также являться структурами. Например, чтобы описать вектор на комплексной плоскости, можно создать структуру, полями которой будут являться две точки – начало и конец вектора:

```
struct complex {
  double Re;
  double Im;
};
struct cVector {
  struct complex p1; /* начало вектора */
  struct complex p2; /* конец вектора */
};
```

Доступ к элементам такой структуры осуществляется с помощью нескольких операций обращения к полю (точка). Например, зададим координаты вектора:

```
struct cVector v;
v.p1.Re = 1.25;
v.p1.Im = 2.5;
v.p2.Re = 4.0;
v.p2.Im = 3.75;
```

В таких *составных именах*, которые еще называются *квалифицируемыми* или *уточненными* идентификаторами, указывается вся цепочка имен от имени переменной-структуры до имени требуемого поля.

Инициализация структуры

Значения полей структуры можно задать при инициализации, используя список констант. Например:

```
struct complex c = { 1.23, 4.567 };
```

Значения присваиваются полям в том, порядке, в котором поля описаны при объявлении структуры (т.е. в данном примере `w.Re=1.23` и `w.Im=4.567`).

Пример инициализации описанного выше вектора `v` , содержащего вложенные структуры:

```
struct cVector v = {
  {1.25, 2.5},
  {4.0, 3.75}
};
```

Здесь компоненты вектора принимает те же значения, что и при присваивании каждого поля по отдельности через составные идентификаторы.

При такой инициализации внутренние фигурные скобки можно опускать:

```
struct cVector v = {1.25, 2.5, 4.0, 3.75};
```

Присваивание структур

Значения всех полей одной структуры (включая вложенные) могут быть присвоены другой структуре такого же типа с помощью одного оператора присваивания. Продолжая приведенный выше пример:

```
struct cVector v1 = {0.0, 0.0, 0.0, 0.0};
v1 = v;
```

Теперь вектор `v1` имеет те же координаты, что и `v`.

Присваивание структур – *исключение* из общего правила, поскольку здесь копирование значений сразу всех полей выполняется целиком, а не поэлементно.

Массивы как поля структур

Полями структур могут являться массивы. Например, структура для хранения данных измерений с помощью некоторого прибора может состоять из нескольких полей: 1) выбранная шкала прибора (`scale`), 2) множитель (`mult`) и 3) значения серии измерений некоторой величины `x`, хранящихся в массиве (не более 10-и измерений в серии):

```
struct tData {
  char scale; /* шкала прибора */
  int mult;  /* множитель */
  double x[10]; /* массив вершин многоугольника */
} data1, data2;
```

При обращении к каждому измерению нужно указывать его индекс в массиве (значение индексов нумеруются как обычно с нуля):

```
data1.scale = 'A';
data1.mult = 4;
data1.x[0] = 123.07;
data1.x[1] = 122.95;
...
data1.x[9] = 123.03;
```

Другой пример. Многоугольник на комплексной плоскости можно описать последовательностью его вершин, координаты которых хранятся в массиве (не более 8-и точек):

```
struct cPolygon {
  struct complex p[8]; /* массив вершин многоугольника */
} poly1, poly2, poly3;
```

Здесь при обращении к каждой вершине нужно указывать ее индекс внутри составного идентификатора справа от имени массива:

```
poly1.p[0].Re = 1.23;
poly2.p[7].Im = 98.7;
```

Типичный пример использования структур

Опишем личную карточку успеваемости студента:

```
struct tStudentCard {
  char   SurName[20]; /* фамилия */
  char   Name[20]; /* имя */
  int    BirthYear; /* год рождения */
  char   HomeAddress[150]; /* домашний адрес */
  int    MathAn;  /* оценка по Мат.анализу */
  int    LinAlg;  /* оценка по Лин.алгебре */
  int    Phys;    /* оценка по Физике */
  int    Inform;  /* оценка по Информатике */
};
```

В приведенном примере поля, несущие одинаковую смысловую нагрузку (`MathAn`, `LinAlg`, `Phys`, `Inform`), целесообразно объединить в отдельную структуру оценок:

```
struct tStudentCard {
  char   SurName[20]; /* фамилия */
  char   Name[20]; /* имя */
  int    BirthYear; /* год рождения */
  char   HomeAddress[150]; /* домашний адрес */
  struct {
    int  MathAn;  /* оценка по Мат.анализу */
    int  LinAlg;  /* оценка по Лин.алгебре */
    int  Phys;    /* оценка по Физике */
    int  Inform;  /* оценка по Информатике */
  } Marks;
};
```

Вложенную структуру оценок лучше описать отдельно с меткой, чтобы ее можно было использовать повторно и не только внутри структуры `tStudentCard`:

```
struct tMarks { /* структура оценки */
  int    MathAn;  /* оценка по Мат.анализу */
  int    LinAlg;  /* оценка по Лин.алгебре */
  int    Phys;    /* оценка по Физике */
  int    Inform;  /* оценка по Информатике */
};
struct tStudentCard {
  char   SurName[20]; /* фамилия */
  char   Name[20]; /* имя */
  int    BirthYear; /* год рождения */
  char   HomeAddress[150]; /* домашний адрес */
  struct tMarks Marks; /* оценки */
};
```

Пусть для последнего варианта описания карточки студента будут объявлены следующие переменные:

```
struct tStudentCard stud1, stud2;
```

Тогда будут корректными следующие присваивания и обращения к полям записей:

```
stud1.BirthYear = 1990;
stud1.Marks.Inform = 5;
stud1.Marks.Phys = 4;
stud2.Marks = stud1.Marks;
stud2 = stud1;
...
```

Поле `Name` структуры `tStudentCard` является массивом символов, поэтому для него допустимо лишь поэлементное заполнение:

```
stud1.Name[0] = 'S';
```

Для заполнения поля целиком можно использовать цикл наподобие следующего:

```
char s[20] = "Sergy";
for (int i=0; i<20; ++i)
  stud1.Name[i]=s[i];
```

Для ввода и вывода значений строковых полей удобно использовать стандартные функции `gets()` и `puts()`, соответственно:

```
puts(stud1.Name);
```

Массивы структур

Работа с массивами структур

Остановимся на последнем варианте описания личной карточки студента. Объявим два массива, соответствующие студенческим группам, элементами которых будут 22 и 25 таких карточек:

```
struct tStudentCard group1[22];
struct tStudentCard group2[25];
```

Тогда для элементов созданных массивов необходимо указывать их индексы внутри составного идентификатора справа от имени массива:

```
group1[0].BirthYear = 1988;
group1[0].Marks.MathAn = 5;
group1[0].Marks.Inform = 4;
```

Так же можно применять следующие операции и обращения к полям структур:

```
group1[2].Marks = group1[1].Marks; /* все оценки совпадают */
group2[5] = group1[3]; /* перевод студента в другую группу */
```

Теперь доступ к элементам поля-массива `Name` структуры `tStudentCard` выполнятся для каждого элемента одного из созданных массивов (`group1` или `group2`) с помощью составного идентификатора с двумя индексами:

```
group1[7].Name[0]:='A';
group1[7].Name[1]:='l';
group1[7].Name[2]:='e';
group1[7].Name[3]:='x';
group1[7].Name[4]:='y';
...
char s[20] = "Ivanov";
for (int i=0; i<20; ++i)
  group2[15].SurName[i]=s[i];
```

Инициализация массива структур

Инициализация массива структур происходит аналогично инициализациям массива и структуры.

Проведем инициализацию личных карточек группы студентов (пусть для краткости в группе имеются только три студента):

```
struct tStudentCard group1[3] = {
  {"Ivanov","Evgeny",1989,"Moscow,str.Lenin,12-67",{4,5,3,5}},
  {"Petrov","Alexy",1990,"Kaluga,str.Marks,56-123",{5,4,5,3}},
  {"Sidorov","Sergy",1989,"Tver,str.Engels,34-45",{5,5,4,5}}
};
```

Здесь при инициализации все внутренние фигурные скобки можно также опустить:

```
struct tStudentCard group1[3] = {
  "Ivanov","Evgeny",1989,"Moscow,str.Lenin,12-67",4,5,3,5,
  "Petrov","Alexy",1990,"Kaluga,str.Marks,56-123",5,4,5,3,
  "Sidorov","Sergy",1989,"Tver,str.Engels,34-45",5,5,4,5
};
```

Указатели на структуры

Объявление указателя на структуру

Объявления указателя на структуру для рассмотренных выше примеров:

```
struct tStudentCard *pstud;
```

Такой указатель может ссылаться на любую существующую структуру `tStudentCard`:

```
pstud = &stud1;
```

В отличие от массивов ***имя структуры не является ее адресом***, поэтому здесь необходимо использовать операцию `&`.

Указатель на структуру, которая является элементом массива:

```
pstud = &group1[0];
```

Тогда выражение `pstud+1` будет указывать на элемент `group1[1]` массива `group1`.

Обращение к полям структуры через указатели

Пусть для объявленного ранее массива `group1` объявлен указатель на структуру, который ссылается на первый элемент массива (с индексом `0`):

```
struct tStudentCard *pstud;
...
pstud = &group1[0];
```

Существуют два способа обращения к полям структуры через указатель на нее:

1) через операцию «**->**» :

```
pstud->SurName  /* значение выражения равно Ivanov */
```

2) через операцию разыменования «*****» :

```
(*pstud).SurName  /* значение выражения тоже равно Ivanov */
```

В первом способе применение операции «->» к указателю на структуру

```
pstud->SurName  /* эквивалентно group1[0].SurName */
```

дает тот же результат, что и имя структуры, за которым следует операция точки

```
group1[0].SurName  /* эквивалентно pstud->SurName */
```

Однако выражение

```
pstud.SurName  /* НЕВЕРНО */
```

использовать нельзя, поскольку `pstud` не является именем структуры.

Следует отметить, что хотя `pstud` – указатель, тем не менее `pstud->SurName` – поле структуры, на которую он указывает, т.е. `pstud->SurName` является массивом из 20-и элементов типа `char`.

Взаимодействие структур и функций

Передача элемента структуры в функцию

Составим функцию, которая для объявленной ранее структуры с меткой `tStudentCard` будет подсчитывать средний балл студента по всем дисциплинам. Передавать в нее данные можно тремя способами.

Первый способ – передача частей структуры:

```
double AverageMark(int MMathAn, int MLinAlg, int MPhys, int MInform) {
  return (MMathAn + MLinAlg + MPhys + MInform) / 4.0;
};
```

Вызов такой функции для первого студента в группе `group1` :

```
printf("Средний балл =%lf\n", AverageMark(group1[0].Marks.MathAn,
                                          group1[0].Marks.LinAlg,
                                          group1[0].Marks.Phys,
                                          group1[0].Marks.Inform));
```

Передача адреса структуры в функцию

Второй способ – передача адреса структуры в качестве аргумента функции:

```
double AverageMark(struct tStudentCard *pstud) {
  return (pstud->Marks.MathAn + pstud->Marks.LinAlg +
          pstud->Marks.Phys   + pstud->Marks.Inform) / 4.0;
};
```

Тогда вызов этой функции для первого студента в группы `group1` :

```
printf("Средний балл =%lf\n", AverageMark(&group1[0]) );
```

Можно составить функцию так, чтобы в нее передавался адрес не всей структуры `group1[0]`, а только той ее части, где указаны оценки по дисциплинам, т.е. адрес поля `group1[0].Marks`, которое само является структурой с меткой `tMarks`:

```
double AverageMark(struct tMarks *pmarks) {
  return (pmarks->MathAn + pmarks->LinAlg +
          pmarks->Phys   + pmarks->Inform) / 4.0;
};
```

Тогда вызов такой функции для первого студента в группы `group1` :

```
printf("Средний балл =%lf\n", AverageMark(&group1[0].Marks) );
```

Передача структуры в функцию

Третий способ – передача самой структуры в качестве аргумента функции:

```
double AverageMark(struct tStudentCard stud) {
  return (stud.Marks.MathAn + stud.Marks.LinAlg +
          stud.Marks.Phys   + stud.Marks.Inform) / 4.0;
};
```

Вызов этой функции для первого студента в группы `group1` :

```
printf("Средний балл =%lf\n", AverageMark(group1[0]) );
```