БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ РАДИОФИЗИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ
КАФЕДРА ИНФОРМАТИКИ И КОМПЬЮТЕРНЫХ СИСТЕМ

Н. В. Серикова

ПРАКТИЧЕСКОЕ РУКОВОДСТВО
к лабораторному практикуму

«СТРУКТУРЫ»

по дисциплине

«ПРОГРАММИРОВАНИЕ НА C++»

2024
МИНСК

Практическое руководство к лабораторному практикуму «СТРУКТУРЫ» по дисциплине «ПРОГРАММИРОВАНИЕ НА С++» предназначено для студентов, изучающих базовый курс программирования на языке С++, специальностей «Компьютерная безопасность», «Прикладная информатика», «Радиофизика».

Руководство содержит некоторый справочный материал, примеры решения типовых задач с комментариями.

Автор будет признателен всем, кто поделится своими соображениями по совершенствованию данного пособия.

Возможные предложения и замечания можно присылать по адресу:
E-mail: Serikova@bsu.by

ОГЛАВЛЕНИЕ

СТРУКТУРЫ

Структура – структурированный тип данных, состоящий из фиксированного числа компонентов одного или нескольких типов. Каждый компонент структуры должен иметь имя, чтобы можно было ссылаться на него. Определение структуры задает ее внутреннюю организацию.

Переменные, входящие в состав структуры, называются **полями** или **членами** структуры.

Компоненты структуры могут иметь любой тип, исключая тип void и тип этой же структуры (но может быть указателем на нее). Допускается вложенность структур.

Структуры в C++ обладают возможностью включать в себя не только переменные различных типов, но и функции. В языке C++ структура является видом класса и обладает некоторыми его свойствами.

Доступ к полю структуры осуществляется по имени переменной типа структуры и имени поля. Поля структуры могут иметь любой тип (кроме типа void и типа этой же структуры). Поля структуры располагаются в памяти последовательно друг за другом. Структура может содержать только такие поля, длина которых известна компилятору в момент определения структуры.

ОПРЕДЕЛЕНИЕ ТИПА СТРУКТУРЫ. ОБЪЯВЛЕНИЕ ПЕРЕМЕННЫХ СТРУКТУРНЫХ ТИПОВ

Для определения типа структуры необходимо указать имена всех полей структуры и их типы. При определении структуры возможно указание имени структуры, но допускается и определение структур, не имеющих имен (анонимная структура).

Синтаксис объявления структурного типа:

```
struct <Имя_структуры>
{
   Тип_поля_1  Имя_поля1;
   Тип_поля_2  Имя_поля2;
                …
};
```

Переменные типа структура объявляются также как переменные других типов. Возможны два способа.

1. При объявлении типа структуры можно сразу объявить одну или несколько переменных этого типа, не задавая имени для типа структуры.

2. Объявить (определить) переменную типа структура можно через имя типа структуры, определенного ранее.

Примеры объявления переменных структурных типов:

```
struct  emp
{
   int       empno;
   char      name [80];
   double    salary;
 };
emp engineer,  teacher,  professor;

struct         // Совмещение объявлений типа и переменных
{
   double re;
   double im;
} c1,  c2,  c3;
```

Примеры объявления переменных структурных типов с инициализацией

```
emp engineer  = {123, "Иванов",   650000},
    teacher   = {124, "Петров",   450000},
    professor = {127, "Сидоров",  790000};


struct         // Совмещение объявлений типа и переменных
{
     double re;
     double im;
} c1 = {0, 0},  c2 = {1, 0},  c3 = {1, 0};
```

ДОСТУП К ЭЛЕМЕНТАМ СТРУКТУРЫ

Оператор «точка» используется для доступа к полям структуры. В контексте структур оператор «точка» называется оператором доступа к полям (членам) структуры. Для доступа к членам вложенной структуры оператор «точка» используется столько раз, какова вложенность структуры.

Имя_переменной.Имя_поля

Примеры:

```
strcpy(engineer.name,"Петров");
teacher.salary  =  professor.salary / 2;
c1.re = 12.5;
c1.im = -24;
c2.re = c1.re * c1.re + c1.im * c1.im;
```

ВЛОЖЕННЫЕ СТРУКТУРЫ

Компоненты структуры могут иметь любой тип, исключая тип void и тип этой же структуры (но может быть указателем на нее). Допускается вложенность структур. Для доступа к членам вложенной структуры оператор «точка» используется столько раз, какова вложенность структуры.

Пример.

```
struct emp
{
  int empno;
  char name [80];
  double salary;
};

struct date
{
  unsigned short year;
  unsigned short month;
  unsigned short day;
};
```

```
struct  form_data
{
   date  birthday;
   emp employment;
};
```

Объявление и инициализация переменных:

```
form_data a, b =  {1933, 5, 19, 2345, "Petrova", 5678};

form_data a, b = { {1933, 5, 19}, {2345, "Petrova", 5678}};

a.birthday.year   = 1961;
a.birthday.month  =  4;
a.birthday.day    = 26;
a.employment       = b.employment;
```

МАССИВ СТРУКТУР

```
const int m = 10,  n = 30

struct student
{
    unsigned int key;
    char     name [m];
};

student group [n];
```

key	name	key	name	key	name	key	name	…	key	name
group[0]		group[1]		group[2]		group[3]			group[9]	

ОБЪЕДИНЕНИЯ

Объединение – частный случай структуры, все поля которой располагаются по одному и тому же адресу. Это означает, что изменение одного поля, автоматически приводит к изменению всех остальных полей объединения. Основное достоинство объединений – возможность разных трактовок одного и того же содержимого (кода) памяти.

Данные, входящие в объединения, называются **полями** или **членами** объединения. Доступ к полям объединения осуществляется аналогично доступу к полям структур с помощью оператора доступа к членам структуры или класса или оператора разыменования указателя на структуру или класс.

Объединение применяют для экономии памяти, когда известно, что больше одного поля одновременно не требуется. **В каждый момент времени в переменной типа объединение хранится только одно значение**, и ответственность за его правильное использование лежит на программисте. В отличие от структур, все поля объединения используют одну и ту же область памяти.

Объединения часто используют в качестве поля структуры, для разной интерпретации одного и того же битового представления данных.

По сравнению со структурой на объединения налагаются некоторые **ограничения**:

- может инициализироваться только значением первого элемента;
- не может содержать битовые поля;
- не может содержать виртуальные методы, конструкторы, деструкторы;
- не может входить в иерархию классов.

Определение объединения задает ее внутреннюю организацию. **Формат описания** такой же, как у структуры, только вместо ключевого слова struct используется union. Для определения объединения необходимо указать имена всех полей объединения и их типы. При определении объединения возможно указать имя для типа объединения, но допускается определение объединения без имени (анонимное объединение).

Длина объединения (размер памяти, выделяемый объединению) равна наибольшей из длин его полей. Тип поля может быть любым, в том числе и структурой.

Пример:

```
union
{
     float          f;
     unsigned long  k;
};
```

Переменные типа объединение объявляются также как переменные других типов. Переменную типа объединение можно объявить через имя типа объединения, определенного ранее. Также можно объявить переменную при определении типа объединения , не задавая имени для типа объединения.

Особенностью инициализации переменной типа объединение в отличие от инициализации структуры является то, что переменная может быть проинициализирована только значением, которое имеет тип первого члена объединения.

БИТОВЫЕ ПОЛЯ

Язык C++ предоставляет возможность задавать количество битов, в которых хранятся элементы целых типов. Это используется для плотной упаковки данных.

Битовые поля – особый вид полей структуры.

Битовое поле – это последовательность соседних двоичных разрядов (бит) внутри одного целого значения.

Битовые поля могут быть любого целого типа. Длина битового поля должна быть неотрицательным целым числом и не должна превышать длины базового типа данных битового поля.

Доступ к битовым полям осуществляется обычным способом – по имени. Адрес поля получить нельзя, в остальном битовые поля можно использовать как обычные поля структуры.

В отличие от обычного поля структуры при описании битового поля после имени переменной (через двоеточие) указывается длина поля в битах (целая положительная константа). Имя поля может отсутствовать, такие поля служат для выравнивания на аппаратную границу.

Операции с отдельными битами реализуются менее эффективно, чем с байтами и словами, и экономия памяти под переменные оборачивается увеличением объема кода программы. Размещение битовых полей в памяти зависит от компилятора и аппаратуры.

Синтаксис описания битового поля:

тип [имя]**:** ширина;

Члены битовых полей могут иметь значения типа signed или unsigned, char или int.

Пример:

```
struct
{
    unsigned  b1:1;
    unsigned  b2:2;
    unsigned  b4:4;
};
```

Массивы битовых полей недопустимы. К битовым полям не может быть применен оператор взятия адреса “&”, так как битовое поле может находиться внутри байта.

ПРИМЕР 1. Структура «комплексное число»

Создать тип для представления комплексного числа. Реализовать ввод-вывод комплексного числа, определить сумму двух комплексных чисел.

```
#include <iostream>  // for cin cout
using namespace std;

struct complex   // объявление структуры комплексное число
{
     float re,  im;
};

complex read ();            // ввод комплексного числа
void print (complex c);     // вывод комплексного числа
complex add (complex, complex); // сумма комплексных чисел

void main()
{
   complex c1, c2, c3; // определение трех комплексных чисел

     c1 = read ();                 // ввод 1 числа
     c2 = read ();                 // ввод 2 числа
     c3 = add (c1, c2);           //сложение двух компл. чисел
     print(c3);                   // вывод результата на экран
}

complex read()                    // ввод комплексного числа
{    complex c;
     cout<<" re = ";        cin>> c.re;
     cout<<" im = ";        cin>> c.im;
     return c;
}
void print (complex c) // вывод на экран комплексного числа
{
     cout<<c.re<< " +i " << c.im <<endl;;
}
                        // сложение двух комплексных чисел
complex add (complex c1, complex c2)
{
     complex c3;
     c3.re  = c1.re  + c2.re;
     c3.im = c1.im + c2.im;
     return c3;
}
```

ПРИМЕР 2. Массив структур

Заполнить массив записей, содержащих поля: ФИО, ключ. Вывести значения записей массива на экран.

```
#include <iostream>  // for cin cout
using namespace std;

int main()
{
     const int m = 40, n = 3;
     int i;
     struct
     {
          unsigned int key;
          char name [m];
     } group [n];              // объявление массива записей

     for (i = 0; i < n; i++)
                    // заполнение полей массива записей
     {
          cout<<" FIO = ";
                 // ввод ФИО для i-го элемента массива
          cin.getline(group[i].name, m);
          cout<<" key = ";
            // ввод значения key для i-го элемента массива
          cin >> group[i].key;
          cin.get();
     }

     for (i = 0; i < n; i++)
          // вывод значений элементов массива на экран
     {
          cout<<" FIO = "<<group[i].name;
          cout<<" key = "<<group[i].key<<endl;
     }
     return 0;
}
```

ПРИМЕР 3. Передача структуры в качестве аргумента. Структура как возвращаемое значение

Заполнить массив записей, содержащих поля: ФИО, ключ. Вывести значения записей массива на экран.

Пример аналогичен предыдущему, но оформлен с использованием функций.

```
#include <iostream>  // for cin cout
using namespace std;

const int m = 40;
struct student           // объявление структуры student
{    unsigned int key;
     char name [m];
};

student read();            //  ввод значений полей структуры
void print(student);      //  вывод значений полей структуры

void main()
{    const int n = 3;
     int i;
     student group[n];      // объявление массива записей
                          // заполнение полей массива записей
     for (i = 0; i < n; i++)
          group[i] = read();

               // вывод значений элементов массива на экран
     for (i = 0; i < n; i++)
          print(group[i]);
}

student read()       //  ввод значений полей структуры
{    student stud;
     cout<<" FIO = ";
     cin.getline(stud.name,m);        // ввод ФИО
     cout<<" key = ";
     cin>>stud.key;                      // ввод значения key
     cin.get();
     return stud;
}
void print(student stud) //  вывод значений полей структуры
{    cout<<" FIO = "<<stud.name;
     cout<<" key = "<<stud.key<<endl;
}
```

ПРИМЕР 4. Побайтный вывод значения вещественного числа (float)

Написать функцию для вывода на экран побайтного представления в ЭВМ вещественного числа в формате float.

```
#include <iostream>  // for cin cout
using namespace std;

union  Uflc             // объявление объединения
{
     float f;           // веществ. число - 4 байта
     unsigned long l;   // длинное целое без знака - 4 байта
     unsigned char c[4];        // массив из 4 байт
};

void print(Uflc);       // вывод полей объединения

void main()
{
     Uflc FLC = {1.0};              // инициализация

     print(FLC);                 // вывод полей

     cout<<" input float number =  ";
     cin>>FLC.f;                 // ввод вещ. числа
     print(FLC);  // вывод по байтам представления вещ. числа

     FLC.l = 1;                   // ввод длинного целого
     print(FLC);                // вывод полей
}

                          //  вывод полей объединения
void print(Uflc FLC)
{
     cout<<" float= "<<FLC.f<<endl;
     cout<<" long = "<<dec<<FLC.l<<endl;
     cout<<" char = ";

     for(int i = 0; i < 4; i++)
          cout<<hex<<(unsigned(FLC.c[i])&0xff)<<" ";
     cout<<endl;
}
```

ПРИМЕР 5. Побайтный вывод значения вещественного числа (double) в двоичном представлении

Написать функцию для вывода на экран побайтного представления в ЭВМ вещественного числа в формате double.

```
#include <iostream>  // for cin cout
using namespace std;

union  bits                          // объявление объединения
{
     double  d;                                  // веществ. число
     unsigned char c[sizeof (double)];     // массив байт
     bits (double n);          // инициализация числа
     void show_bits();         // побайтный вывод на экран
};

bits::bits(double n)
{
  d = n;
}

void bits::show_bits()       // побайтный вывод на экран
{
     for(int j = sizeof (double)- 1; j >= 0; j--)
     {
          cout<<" byte  "<< j << " = ";
          for(int i = 128; i ; i >>= 1)
               if (i & c[j])cout<<"1";
               else cout<<"0";
          cout<<endl;
     }
}

void main()
{
     double x;
     cout<<" input double number =  ";
     cin>>x;                  // ввод вещ. числа

     bits ob(x);              // инициализация
     ob.show_bits();          // вывод
}
```

ПРИМЕР 6. Код символа через объединение

Формирование кода символа с помощью битовых полей объединения:

```
#include <iostream>  // for cin cout
using namespace std;

void main( void )
{

     union
     {
          char simb;
          struct
          {
               int x : 5;
               int y : 3;
          };
     } cod;

     cod.x = 4;                          //  x = 001002
     cod.y = 2;                          //  y = 0102
     cout<<cod.simb;       // 'D' (код =  44 16 = 010001002)
}
```

ПРИМЕР 7. Битовое представление целого числа

Написать функцию для вывода на экран битового представления в ЭВМ числа в формате unsigned char.

```
#include <iostream>  // for cin cout
using namespace std;
    //вывод на экран двоичного представления байта-параметра
void binary (unsigned char ch)
{
    union
    {
        unsigned char ss;
        struct
        {            unsigned char a0:1;
            unsigned char a1:1;
            unsigned char a2:1;
            unsigned char a3:1;
            unsigned char a4:1;
            unsigned char a5:1;
            unsigned char a6:1;
            unsigned char a7:1;
        };
    } cod;

    cod.ss = ch;//занесение в объединение значения параметра
    cout <<"  bity =  ";

                                          // вывод бит
    cout <<" "<<cod.a7<<" "<<cod.a6
        <<" "<<cod.a5<<" "<<cod.a4
        <<" "<<cod.a3<<" "<<cod.a2
        <<" "<<cod.a1<<" "<<cod.a0<<endl;
}


void main()
{
    unsigned c;
    cin>>c;                    // вводим число 0..255
    if (c < 256)
        binary(c);             // битовое представление числа
}
```

СЛОВАРЬ ПОНЯТИЙ, ИСПОЛЬЗУЕМЫХ В ЗАДАНИЯХ