Составные типы данных (часть 4)

Структуры. Вводная

- В некоторых задачах удобно оперировать совокупностью переменных как одним программным объектом;
  - например программа, которая содержит информацию о группе людей;
    - о каждом человеке требуется хранить следующие данные: имя, пол, год рождения, рост, вес и т. д;

```
//Для формирования информации о каждом человеке требуется определить
  набор переменных
//Для Васи:
      char name1[30] = "Вася";
```

```
      int age1 = 30;
```

Структуры. Вводная

Возникают проблемы:

- А если таких наборов данных много?
  - должны все время помнить, какие данные относятся к одному и тому же конкретному человеку;
- поэтому логично отдельные характеристики, относящиеся к одному объекту, объединить в составном элементе данных:
  - хранить совокупность характеристик как единое целое;
  - манипулировать этой совокупностью как единым целым;
  - иметь возможность обращаться к характеристикам по отдельности.

Структуры языка Си как раз и предоставляют программисту возможность формирования таких **новых составных типов данных**, которые строятся на базе уже существующих типов.

Структуры. Вводная

- Структура может включать в свой состав произвольное количество типов данных;
  - которые в дальнейшем будем называть ***полями*** структуры;
- В качестве поля можно использовать любой из ранее определенных типов данных, как базовых, так и более сложных: указатели, строки, массивы, другие структуры;
- Разные типы структур могут различаться:
  - количеством полей;
  - их типами;
  - порядком расположения полей в структуре.

Структуры. Объявление

- Компилятор сам знает без дополнительных указаний программиста, как обращаться с переменными базового типа;
  - Программисту нужно лишь объявить переменную требуемого (базового) типа;
- Однако только программист знает, **сколько и каких полей** должен содержать пользовательский тип;
- Программист должен описать компилятору свойства своего **нового пользовательского типа данных**, т. е. **объявить** структуру

Объявление структуры — это описание внутреннего устройства нового типа данных.

- количества, типа и порядка расположения полей;

Структуры. Объявление

Синтаксис объявления структуры:

```
struct имя_пользовательского_типа{//ключевое слово struct означает, что
                                  вводится новый пользовательский
                                  агрегатный тип данных с указанным
                                  именем
        список_полей_структуры (типы и имена переменных, которые будут
                                присутствовать в каждом создаваемом
                                экземпляре такой структуры)
}; //после закрывающей фигурной скобки точка с запятой обязательна!
```

Структуры. Объявление. Пример

Пример. Спроектируем структуру human для описания свойств любого человека

```
struct human{//ввели свой пользовательский агрегатный тип human,
             в котором сгруппировали все нужные поля данных

        int age;
        char name[30];
        …
};
```

Структуры.Создание экземпляров.Присваивание

Создание объекта пользовательского типа выглядит так же, как создание переменной базового типа:

```
int main()
{
        //Создание переменной пользовательского типа:
//В языке ANSI Си
        struct human man1; //тип переменной - struct human,
                           имя переменной - man1. Ключевое
                           слово struct обязательно в языке Си,
                           но не обязательно в С++, где достаточно
                           написать так:
//В С++
        human woman1; //тип переменной - human, имя переменной - woman1,
                      а о том, что human - это структура, компилятор
                      помнит сам
}
```

Структуры.Создание экземпляров.Присваивание

- Если в распоряжении программиста есть переменная структурного типа, то обратиться к полю структуры можно посредством оператора `.` ;

Синтаксически обращение выглядит:

```
имя_переменной.имя_поля
```

Структуры.Создание экземпляров.Присваивание

Пример.

Очень большой

Смотрим в код...

Структуры.Совмещение объявления и определения

Иногда для локального использования совмещают объявление структуры и создание переменных структурного типа:

```
struct human{ //ввели свой пользовательский агрегатный тип human
       SEX sex;
       int age;
       char name[30];
       …
       } man1, woman1, *phuman, people[10]; //создали две переменных типа
                                            human - man1, woman1,
                                            указатель типа human* - phuman
                                            и массив из 10 элементов
                                            типа human - people
```

Структуры.Совмещение объявления и определения

Синтаксис позволяет **объявить** структуру, не указывая имени типа. При этом очевидно, что в дальнейшем использовать такую структуру невозможно

```
int main()
{
        //Создание переменных структурного типа:
        struct { //имя пользовательского типа опущено
                SEX sex;
                int age;
                char name[30];
                …
                } man1, woman1, *phuman, people[10];
        //Использование переменных man1, woman1, phuman, people
}
```

Структуры. Инициализация

- При определении структуры, ее поля можно проинициализировать явно;
  - как и при определении переменной базового типа;
- Инициализация структур похожа на инициализацию массивов;
  - Инициализаторы в фигурных скобках указываются в том же порядке, в котором в структуре объявлены соответствующие поля:

```
struct human man1 = {MALE, 30, "Вася"}; //определена структурная
                                        переменная с именем man1 и ее
                                        поля проинициализированы при
                                        создании указанными значениями
```

- Инициализация массивов структур похожа на инициализацию многомерных массивов:

```
struct human people[100] = {
                           {MALE, 30, "Вася"},
                           {FEMALE, 20, "Маша"},
                                    …
                                       };
```

Структуры. Инициализация

- Для структур справедливы правила неполной инициализации (так же, как и для массивов):

```
struct human man2 = {MALE}; //все остальные поля компилятор
                            проинициализирует нулевыми значениями
struct human man3 = {0};    //компилятор обнулит все поля структуры
```

- Так же, как и для массивов, проинициализировать поля структуры можно только при создании;
  - поэтому попытка использовать список инициализаторов позже вызовет ошибку;

```
man2 = {MALE, 30, "Вася"}; //ошибка: список инициализации можно
                           использовать только при определении
```

Действия над структурами

Компилятор умеет:

- **создавать** копии существующих структурных переменных;
- **копировать** поля одного (уже существующего экземпляра структуры);
- **поля** другого (тоже уже существующего) экземпляра структуры того же типа;

```
struct human man1 = {"Вася", MALE, 30};
struct human man2 = man1; //создание нового объекта и инициализация.
                          Компилятор выделяет память для man2 и
                          копирует значения всех полей экземпляра
                          man1 в соответствующие поля экземпляра
                          man2. Таким образом, man2 становится
                          копией man1
man1 = man2; //присваивание: значения полей одного существующего
               экземпляра заменяются значениями полей другого
               существующего
```

Структуры. Поля структуры

Поле структуры может быть:

- базового типа — `int age`;
- указателем на базовый тип — `int* p`;
- массивом элементов базового типа — `char name[30]`;
- пользовательского типа (другая структура).

Структура. sizeof()

- При выделении памяти под структурную переменную (независимо от того, в какой конкретно области выделяется память:
  - в стеке, в статической области, в куче);
- компилятор гарантированно делает следующее:
  - выделяет количество байтов, большее или равное сумме всех полей структуры;
  - выделяет память для каждого поля в том порядке, в котором поля объяв- лены в структуре;
- Количество выделяемой для структурной переменной памяти зависят от:
  - оптимизирующих возможностей конкретного компилятора;
  - опций командной строки компилятору.