ШАГ
компьютерная
АКАДЕМИЯ

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ C++

Урок № 16

Структуры

Содержание

1. Структура. Определение структур 3
   Особенности структур 5
2. Действия над структурами 9
3. Оператор sizeof 16
   Размеры структуры 16
4. Домашнее задание 18

Материалы урока прикреплены к данному PDF-файлу. Для доступа к материалам, урок необходимо открыть в программе Adobe Acrobat Reader.

1. Структура. Определение структур

В самом начале изучения языка C++ мы с Вами познакомились с понятием типа данных. Сегодня мы попробуем расширить это понятие. Дело в том, что помимо стандартных типов программист может создавать свои собственные составные типы данных под названием структуры. Именно они и будут являться темой нашего занятия.

Структура — это множество, состоящее из одной и более переменных, возможно имеющих различные типы, объединенных под одним именем.

Тип данных struct — один из основных строительных блоков данных в языке. Он предоставляет удобный способ объединения различных элементов, связанных между собой логической связью. Например, с помощью структуры можно объединить данные, относящиеся к студенту (ФИО. Название группы, название факультета и т.д.) или можно создать структуру, описывающую Город (название города, название страны, количество жителей и т.д.). Структура — отличный механизм для создания новых типов данных в ваших программах.

Рассмотрим особенности работы со структурами.

```
#include <iostream>
using namespace std;

/*
   Создание пользовательского типа данных
```

```
   (структуры) для хранения даты.
   Все данные, касающиеся одного объекта собраны
   вместе.
*/

struct date
{
    int day; // день
    int month; // месяц
    int year; // год
    int weekday; // день недели
    char mon_name[15]; // название месяца
};

int main(){
    // создание объекта с типом date и инициализация
    // его при создании
    date myBirthday={20,7,1981,1,"July"};
    // показ содержимого объекта на экран
    // обращение к отдельному члену структуры
    // производится через оператор точка (.)
    cout<<"________MY BIRTHDAY __________\n\n";
    cout<<"DAY "<<myBirthday.day<<"\n\n";
    cout<<"MONTH "<<myBirthday.month<<"\n\n";
    cout<<"YEAR "<<myBirthday.year<<"\n\n";
    cout<<"WEEK DAY "<<myBirthday.weekday<<"\n\n";
    cout<<"MONTH NAME ";
    cout<<myBirthday.mon_name<<"\n\n";
    // Создание пустого объекта и заполнение его
    // с клавиатуры date your_birthday;
    cout<<"DAY ? ";
    cin>>your_birthday.day;
    cout<<"MONTH ?";
    cin>>your_birthday.month;
    cout<<"YEAR ?";
    cin>>your_birthday.year;
    cout<<"WEEK DAY ?";
```

```
    cin>>your_birthday.weekday;
    cout<<"MONTH NAME ?";
    cin>>your_birthday.mon_name;
    cout<<"_____YOUR BIRTHDAY_____\n\n";
    cout<<"DAY "<<your_birthday.day<<"\n\n";
    cout<<"MONTH "<<your_birthday.month<<"\n\n";
    cout<<"YEAR "<<your_birthday.year<<"\n\n";
    cout<<"WEEK DAY "<<your_birthday.weekday<<"\n\n";
    cout<<"MONTH NAME ";
    cout<<your_birthday.mon_name<<"\n\n";
    return 0;
}
```

Особенности структур

1. Описание структуры начинается со служебного слова struct, за которым может следовать необязательное имя, называемое именем типа структуры (в примере выше date). Это имя типа структуры используется в дальнейшем для создания экземпляра структуры. Конкретный экземпляр типа структура называют объектом. В наше примере было два объекта myBirthday и your_birthday.
2. За именем типа структуры идет заключенный в фигурные скобки список элементов структуры, с описанием типа каждого элемента (элементом структуры может быть переменная, массив или объект другой структуры). Элементы структуры отделяются друг от друга точкой с запятой. Например:

```
struct date
{
    int day;
```

```
    int month;
    int year;
    int yearday;
    char mon_name[5];
};
```

Обратите внимание, что после закрывающей фигурной скобки, вы должны обязательно поставить;
Иначе будет ошибка на этапе компиляции.

3. За фигурной скобкой, закрывающей список элементов, может следовать список объектов. Например, оператор struct date {...} x, y, z; определяет переменные x, y, z в качестве объектов структуры описанного типа. Под x, y, z выделяется память, как под любые переменные. После последнего объявленного объекта должна быть указана;
   Изобразим распределение памяти для объекта x типа date:

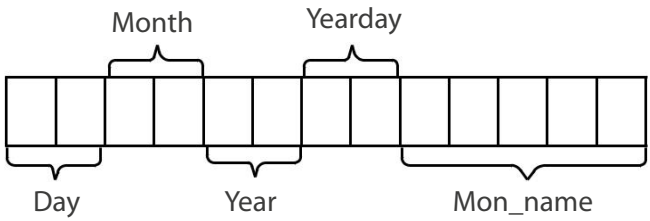

Рисунок 1

4. Описание структуры, за которым не следует список объектов, не приводит к выделению памяти (как в программе выше); оно только определяет шаблон (форму) структуры. Однако, если такое описание снабжено именем типа (например, date), то это имя типа может

быть использовано позднее при определении объектов структур (определение your_birthday и myBirthday в программе).

5. Объект структуры можно инициализировать, поместив вслед за его объявлением список инициализаторов для ее полей, заключенный в фигурные скобки. В программе таким образом инициализирован объект myBirthday.

```
date myBirthday={20,7,1981,1,"July"};
```

6. Обращение к определенному члену (полю) структуры производится с помощью конструкции вида:

```
<имя структуры>.<имя элемента>
```

7. Объекты структур можно вкладывать в создание новых структур.

Например, учетная карточка служащего может выглядеть так:

```
struct date
{
    int day; // День.
    char month[10]; // Месяц.
    int year; // Год.
};

struct person
{
    char name[namesize]; // Имя, фамилия, отчество.
    char address[adrsize]; // Домашний адрес.
    int zipcode[2]; // Почтовый индекс.
    int s_number [2]; // Код соц.обеспечения.
```

```
    int salary[4]; // Зарплата.
    date birthdate; // Дата рождения.
    date hiredate; // Дата поступления на работу.
};
```

Структура person содержит две объекта типа date. Структура date определена выше в этом же примере.

Если определить переменную Nick следующим образом:

```
person Nick;
```

то Nick.birthdate.month будет обозначать месяц рождения. Операция доступа к элементу структуры «.» вычисляется слева направо.

А, теперь, рассмотрев основы, перейдем к тому, что же можно делать со структурами. Для этого открываем следующий раздел урока.

2. Действия над структурами

Итак, разберем операции, которые допустимо производить над структурами.

1. Доступ к члену структуры с помощью операции «.» (точка).
2. Доступ к члену структуры по указателю с помощью операции «->» в виде: <указатель на структуру>«->»<-элемент_структуры>. Так, обращения pd->year и (*pd).year эквивалентны. Круглые скобки (*pd) необходимы, поскольку операция «.» доступа к элементу структуры старше, чем «*».
3. Определение адреса объекта структуры с помощью операции «&».

Покажем использование этого аппарата на примере:

```
#include <iostream>
using namespace std;

/*
  Описание структуры с именем date.
*/
struct date
{
    int day;
    int month;
    int year;
    char mon_name[12];
};
```

```
int main()
{
    /* Создание и инициализация объекта структуры.
        d - переменная типа date.
        2 попадет в day
        5 попадет в month
        1976 в year
        July в mon_name
    */

    date d = { 2,5,1976,"July" };

    // Указатель p указывает на структуру типа date
    date* p = nullptr;

    // Показ содержимого структуры на экран
    // (обращение через объект)
    cout << d.day << " ";
    cout << d.month << " ";
    cout << d.year << " ";
    cout << d.mon_name << "\n\n";

    // запись адреса объекта структуры в указатель
    p = &d;

    // Показ содержимого структуры на экран
    // (обращение через указатель)
    // -> состоит из минуса - и знака больше >
    cout << p->day << " ";
    cout << p->month << " ";
    cout << p->year << " ";
    cout << p->mon_name << "\n\n";
    return 0;
}
```

4. Присваивание объектов структуры друг другу. При присваивании значения полей одного объекта структуры копируются в другой объект.

```
#include <iostream>
using namespace std;

struct date{
    int day;
    int month;
    int year;
    char mon_name[12];
};

int main(){

    // создаём объект и инициализируем поля
    date a = { 14,7,1954,"July" };

    // создали объект без инициализации
    date b;

    // показ содержимого объекта a
    cout << a.day << " ";
    cout << a.month << " ";
    cout << a.mon_name << " ";
    cout << a.year << "\n\n";

    /*
        Инициализация объекта b объектом a
        Все значения полей из a копируются в b
    */
    b = a;

    // показ содержимого объекта b
    cout << b.day << " ";
    cout << b.month << " ";
    cout << b.mon_name << " ";
    cout << b.year << "\n\n";
    return 0;
}
```

5. Передача объектов структуры в качестве параметра функции и возвращение объектов структуры в результате работы функции.

```
#include <iostream>
using namespace std;

struct date{
    int day;
    int month;
    int year;
    char mon_name[12];
};

/*
   Объект структуры передаётся в функцию по значению
   Возможна также передача по ссылке и по указателю
*/
void Show(date a) {
    // показ содержимого объекта a
    cout << a.day << " ";
    cout << a.year << " ";
    cout << a.month << " ";
    cout << a.mon_name << "\n\n";
}

date Put() {
    // формирование объекта
    date temp;
    cout << "DAY ? ";
    cin >> temp.day;
    cout << "MONTH ? ";
    cin >> temp.month;
    cout << "YEAR ? ";
    cin >> temp.year;
    cout << "MONTH NAME ? ";
    cin >> temp.mon_name;
```

```
    return temp;
}

int main(){
    date a = { 14,7,1954,"July" };
    date b;
    // передача объекта в функцию
    Show(a);
    // получение объекта в качестве возвращаемого
    // значения
    b = Put();
    // показ содержимого объекта b
    Show(b);
    return 0;
}
```

6. Передача отдельных компонент структуры в качестве аргументов функции. Например, для функции IsLeapYear. Функция проверяет год на високосность:

```
#include<iostream>
using namespace std;

struct date {
    int day;
    int month;
    int year;
    char mon_name[12];
};

// проверка на високосный год
bool IsLeapYear(int checkYear) {
    // год не високосный, если условие истина
    if(checkYear%4!=0 || ((checkYear%100==0)&&
                          (checkYear%400!=0))) {
```

```
        return false;
    }
    // если условие ложно год високосный
    return true;
}

int main(){
    int tempYear = 2020;
    date curDate = { 12,5,tempYear,"May" };

    if (true == IsLeapYear(curDate.year)) {
        cout << tempYear << " is leap year\n";
    }
    else {
        cout << tempYear << " is NOT leap year\n";
    }

    return 0;
}
```

А теперь, рассмотрим порядок выполнения некоторых наиболее распространенных операций над элементами структуры на примере следующего описания:

```
struct
{
    int x;
    int *y;
} *p; // p - указатель на структуру.
```

- (++p)->x — операция увеличивает p до доступа к x;
- (p++)->x — операция увеличивает p после доступа к x (круглые скобки не обязательны, так как по старшинству раньше будет применена операция «->»);

- *p->y — выбирается содержимое объекта, на который указывает y;
- *p->y++ — увеличивается y после обработки того, на что он указывает (аналогично *s++);
- *p++->y — увеличивает p после выборки того, на что указывает y;
- (*(*p).y)++ — увеличивает то, на что указывает y.

Можно отметить одно очень важное использование структур: создание новых типов данных. Существуют типы данных, гораздо более эффективные при решении определенных задач, чем массивы и структуры. Это очереди, двоичные деревья, множества, таблицы и графы. Многие из этих типов создаются из «связанных» структур. Обычно каждая такая структура содержит один или два типа данных плюс один или два указателя на другие структуры такого же типа. Указатели служат для связи одной структуры с другой и для обеспечения пути, позволяющего вести поиск по всей структуре.

Об очередях, двоичных деревьях и другие сложных сущностях вы узнаете в ближайшем будущем.

3. Оператор sizeof

В языке C++ существует специальная унарная операция sizeof, которая возвращает размер своего операнда в байтах. Операндом операции sizeof может быть любое выражение:

```
sizeof(Выражение);
```

Результат операции sizeof имеет тип int. Пример использования:

```
#include <iostream>
using namespace std;

int main(){
    int a;
    char b;
    double c;
    int* p;
    cout << "sizeof(a)=" << sizeof(a) << "\n";
    cout << "sizeof(b)=" << sizeof(b) << "\n";
    cout << "sizeof(c)=" << sizeof(c) << "\n";
    cout << "sizeof(p)=" << sizeof(p) << "\n";
    cout << "sizeof(int)=" << sizeof(int) << "\n";
    cout << "sizeof(int *)=" << sizeof(int*) << "\n";
    return 0;
}
```

Размеры структуры

Наверняка, вы предполагаете, что размер структуры равен сумме размеров ее членов. Это не так. Вследствие выравнивания объектов разной длины в структуре могут

появляться безымянные «дыры». Так, например, если переменная типа char занимает один байт, а int — четыре байта, то для структуры:

```
#include <iostream>
using namespace std;

struct Test
{
    char c;
    int i;
};

int main ()
{
    Test d={'#',78};
    cout<<sizeof(Test)<<" "<<sizeof(d)<<"\n\n";
    return 0;
}
```

может потребоваться восемь байт, а не пять. Правильное значение возвращает операция sizeof.

4. Домашнее задание

Создать структуру ВИДЕОМАГАЗИН со следующими полями:

- Название фильма;
- Режиссер;
- Жанр;
- Рейтинг популярности;
- Цена диска.

Реализовать следующие возможности:

- Поиск по названию;
- Поиск по жанру;
- Поиск по режиссеру;
- Самый популярный фильм в жанре;
- Показ всех записей и добавление.

Для реализации задачи используйте функции. Объекты структуры можно передавать в функцию целиком или отдельными полями. Выберите верный механизм передачи в каждом конкретном случае.

Урок № 16

Структуры