Оглавление

Структура. Определение структур.

В самом начале изучения языка С мы с Вами познакомились с понятием *типы 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 одного или более объектов, 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.

Рассмотрим особенности работы со структурами на примере.

```
#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
};

void main(){

        //создание объекта с типом date и инициализация его при создании
        date my_birthday={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 "<<my_birthday.day<<"\n\n";
        cout<<"MONTH "<<my_birthday.month<<"\n\n";
        cout<<"YEAR "<<my_birthday.year<<"\n\n";
        cout<<"WEEK DAY "<<my_birthday.weekday<<"\n\n";
        cout<<"MONTH NAME "<<my_birthday.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 "<<your_birthday.mon_name<<"\n\n";

}
```

Особенности структур.

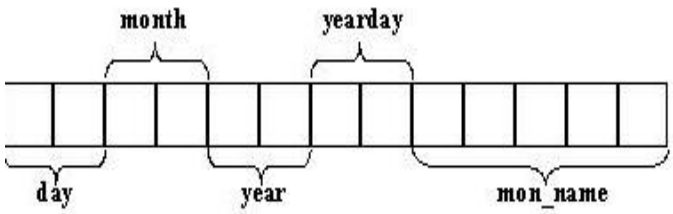
1. Описание структуры начинается со служебного слова struct, за которым может следовать необязательное имя, называемое именем типа структуры (здесь это date). Это имя типа структуры используется в дальнейшем для создания конкретного объекта.

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 структура). Элементы структуры отделяются друг от друга точкой с запятой. Например:

```
struct date
{
  int day;
  int month;
  int year;
  int yearday;
  char mon_name[5];
};
```

3. За правой фигурной скобкой, закрывающей список элементов, может следовать список объектов. Например, оператор struct date {...} x,y,z; определяет переменные x,y,z в качестве структур описанного типа и приводит к выделению памяти.

Изобразим распределение памяти для структуры x типа date:



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 структур your_birthday и my_birthday в программе).

5. Структуру можно инициализировать, поместив вслед за ее определением список инициализаторов для ее компонент, заключенный в фигурные скобки. В программе таким образом инициализирована структура my_birthday.

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

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

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

7. Структуры могут вкладываться одна в другую, но **самовложение структур запрещено!**

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 структуры типа date и наша программа должна содержать шаблон для структуры date.

Если определить переменную Nick следующим образом:

```
struct 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 данного урока.

Действия над структурами

Итак, разберем операции, которые допустимо производить над структурами.

1. Доступ к элементу структуры с помощью операции "." (точка).

2. Доступ к элементу 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 структуры с помощью операции "&".

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

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

//Создание и инициализация объекта структуры.
date d = { 2,5,1776,"July" }; //d - переменная типа date.

void main ()
{
        // Указатель p указывает на структуру типа date.
        struct date *p;

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

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

        // Показ содержимого структуры на экран
        //(обращение через указатель)
        cout << p->day << " ";
        cout << p->month << " ";
        cout << p->year << " ";
        cout << p->mon_name << "\n\n";
}
```

4. Присваивание структуры как единого целого.

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

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

```
date a = { 14,7,1954,"July" };
date b;

void main ()
{
        // показ содержимого объекта a
        cout<< a.day << " ";
        cout<< a.year << " ";
        cout<< a.month << " ";
        cout<< a.mon_name << "\n\n";

        // инициализация объекта b объектом a
        b = a;

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

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

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

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

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;
}

date a = { 14,7,1954,"July" };
date b;

void main ()
{
```

```
        // передача объекта в функцию
        Show(a);

        // получение объекта в качестве возвращаемого значения
        b=Put();

        // показ содержимого объекта b
        Show(b);

}
```

6. Передача отдельных компонент структуры в качестве аргументов функции. Например, для функции day_of_year1:

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

// вспомогательный массив, отвечающий за месяцы в году
int day_tab[2][13]=
{0,31,28,31,30,31,30,31,31,30,31,30,31,
0,31,29,31,30,31,30,31,31,30,31,30,31};

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

void Show(date a){
        // показ содержимого объекта a
        cout<< a.day << " ";
        cout<< a.year << " ";
        cout<< a.month << " ";
        cout<< a.dayyear << " ";
        cout<< a.mon_name << "\n\n";
}

int day_of_year1 (int day,int month,int year)
{
        //Вычисление дня в году с помощью месяца и года.
        int i, leap;
        leap = year%4==0 && year%100!=0 || year%400==0;
        for (i=1; i <; month; i++)
                day += day_tab[leap][i];
        return (day);
}

date a = {20,7,1981,0,"July"};

void main ()
{
        // передача отдельных членов  функцию
        a.dayyear=day_of_year1(a.day,a.month,a.year);

        // показ содержимого объекта a
        Show(a);
}
```

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.

Оператор sizeof

В языке С 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.

Получение размера объекта.

```
#include <iostream>
using namespace std;
void main ()
{
      int a;
      char b;
      unsigned 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";
}
Результат работы программы:
sizeof(a)=4
sizeof(b)=1
sizeof(c)=4
sizeof(p)=4
sizeof(int)=4
sizeof(int *)=4
```

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

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

Домашнее задание.

1. 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