Лекции по C/C++: составные типы данных (структуры)

Представим себе, что нам нужно программно обработать информацию о студентах. Информация о каждом студенте включает в себя величины нескольких различных между собой типов данных – строку с именем и инициалами, целое число, показывающее возраст студента в годах, средний балл, являющийся вещественным значением, и т.д. Простейший путь решения задачи – описать в программе набор соответствующих переменных:

```
char name[20]; //фамилия
int age;       //возраст в годах
float ball;    //средний балл
```

Когда студентов становится несколько, нам придётся перейти от переменных к массивам:

```
#define ST 5 /* количество студентов */
char name[ST][20];
int age[ST];
float ball[ST];
```

Чтобы извлечь информацию о последнем студенте, нам нужно обработать величины `name[4]`, `age[4]` и `ball[4]`. Таким образом, информация об одном объекте оказывается "раскидана" по нескольким контейнерам-массивам, что делает крайне неудобными операции со "студентом" как единым объектом. Преодолеть это неудобство может *структура*, позволяющая объединять в одном объекте разнотипные данные.

Итак, структура – это основной составной тип данных. В отличие от массива, она объединяет в одном объекте значения, которые могут иметь разные типы данных. Это делает структуры основным средством моделирования в программе объектов реального мира:

```
struct student {
 char name[20]; //фамилия
 int age;       //возраст в годах
 float ball;    //средний балл
};
```

Для совместимости с языком C перед ключевым словом `struct` часто записывают оператор определения типа `typedef`. На C++ этот оператор является излишним.

Опишем переменные "нового" типа данных `student`:

```
student s,s2;
```

Никакой разницы с описанием переменных простых типов `int`, `float` и т.д., нет.

Для доступа к отдельным данным о студенте, перечисленным в описании структурного типа (*полям* структуры) служит операция . ("точка"), применимая в записи вида `структура.поле`. Например, `s.name` будет означать поле "имя" студента, информация о котором сохранена в переменной с именем `s`.

При этом, с полем структуры разрешены все те же операции, что и с "обычной" переменной соответствующего типа данных:

```
strcpy (s.name, "Ivanov");
s.age = 19; s.ball = 3.5;
printf ("\n%s,%d,%.2f",s.name,s.age,s.ball);
```

Со структурами как с единым целым разрешены следующие операции:

- передача в качестве аргумента функции;
- возврат из функции в качестве её значения;
- получение адреса структуры в памяти и создание указателя на неё;
- присваивание структур одинакового типа.

Рассмотрим первые два действия на примере.

1. Функция `st_print` выводит информацию о структуре типа `student`, переданной в качестве параметра;

2. Функция `st_new` заполняет новую структуру `student` данными по умолчанию и возвращает структуру. За счёт того, что поддерживается присваивание структур, полученная из функция информация дублируется в структуре `s2` функции `main`.

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

struct student {
 char name[20]; int age;         float ball;
};

void st_print (student s) {
 cout << s.name << ", age=" << s.age << ", ball=" << s.ball << endl;
}

student st_new () {
 student s;
 strcpy (s.name,"Noname");
 s.age=17;
 s.ball=0.;
 return s;
}

int main() {
 student s1 = { "Petrov", 19, 4.2 };
 st_print (s1);
 student s2 = st_new();
 st_print (s2);
```

```
 cin.get(); cin.sync(); return 0;
}
```

Заметим, что передача в качестве параметров функции или возврат из функции структур большого размера могут быть чреваты переполнением стека. А определить объём структуры в оперативной памяти можно стандартными средствами:

```
sizeof(student)
 //количество байт, занимаемое структурой
```

Для сохранения информации о 20 студентах группы естественным выглядит описание массива структур:

```
student group[20];
```

Как и любой массив, массив структур тоже можно обрабатывать поэлементно, например, вычислим средний балл группы (предполагая, что данные уже введены или прочитаны из файла):

```
float average = 0.;
for (int i=0; i<20; i++) average += group[i].ball;
average /= 20;
```

Доступ к структуре можно осуществлять и *через указатель*:

```
student *ptr = &group[0];
```

После этого можно выполнять `ptr++` для сканирования списка или массива структур таким образом, как мы поступали с указателями на простые типы данных.

Доступ *к полю структуры через указатель* имеет следующий вид:

```
(*ptr).field
```

Круглые скобки здесь необходимы из-за приоритетов операций языка C++. Запись выглядит громоздкой, поэтому, вместо неё, как правило, применяют мнемоническое сокращение

```
ptr->field
```

Пример 1. Функция `show` печатает информацию о студенте `s` (без указателя):

```
void show (student s) {
 printf ("\n%s,%d,%.2f",s.name,s.age,s.ball);
}
```

Та же функция с указателем:

```
void show (student *s) {
 printf ("\n%s,%d,%.2f",s->name,s->age,s->ball);
}
```

В первом случае через стек передано `sizeof(student)=sizeof(char)*20+sizeof(int)+sizeof(float)` байт (от 28 байт, если символы `char` - однобайтовые), во втором случае - 4 или 8 байт (в зависимости от размера указателя в 32-разрядной или 62-разрядной архитектуре ЭВМ).

Для i-го элемента нашего массива структур вызов первой функции имел бы вид

```
show (group[i]);
```

а второй - вид

```
show (&group[i]);
```

Пример 2. Показанная выше функция `st_new` с использованием указателя приняла бы вид

```
student *st_new () {
 student *s = new student;
 strcpy (s->name,"Noname");
 s->age=17;
 s->ball=0.;
 return s;
}
```

...и её вызов:

```
student *s2 = st_new();
st_print (*s2);
```

Поскольку оператор `new` выделяет память в "куче", а не в стеке, её потом можно освободить оператором `delete`.

Остановимся на *особенностях присваивания структур*. Следует понимать, что если структурный тип содержит указатели или динамические поля (под которые мы выделяли оперативную память), операция присваивания таких структур будет работать некорректно. Например, если после операции `a=b` объект `b` удалён, а скопированный указатель в объекте `a` продолжает показывать на несуществующий адрес.

Поясним на примере 2 программ.

В первой программе все поля структуры `student` – статические, под них не выделялась динамическая память, поэтому присваивание таких структур корректно:

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

struct student {
 char name[20]; int age;        float ball;
};

void st_print (student s) {
 cout << s.name << ", age=" << s.age << ", ball=" << s.ball << endl;
}

int main() {
 student s1 = { "Petrov", 19, 4.2 };
 student s2 = s1; //Так делать можно
 strcpy (s1.name,"Ivanov");
 st_print (s2);
 cin.get(); cin.sync(); return 0;
}
```

Во второй программе статическое поле `name` заменено динамическим, под поле "имя" каждого студента теперь нужно выделять динамическую память. Присваивание таких структур оператором вида `s2=s1` будет означать установку указателя `s2.name` на тот же адрес оперативной памяти, который содержится в `s1.name`.

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

struct student {
 char *name; int age; float ball;
};

void st_print (student s) {
 cout << s.name << ", age=" << s.age << ", ball=" << s.ball << endl;
}

int main() {
 student s1;
 s1.name = new char [20]; //выделяем память!
 strcpy (s1.name,"Petrov"); s1.age=19; s1.ball=4.2;
 student s2 = s1; //А так делать нельзя!
 delete s1.name;
 st_print (s2); //крах программы
 cin.get(); cin.sync(); return 0;
}
```

Как вариант, поставим вместо строки

```
delete s1.name;
```

строку

```
strcpy (s1.name,"Ivanov");
```

Так как указатели `s1.name` и `s2.name` показывают на один и тот же адрес памяти, после `strcpy` изменится и имя второго студента!

Элемент структуры может быть структурой другого структурного типа или указателем на структуру своего типа. Последнее важное свойство структур позволяет организовывать динамические структуры данных, такие как стеки, очереди или деревья. Динамическим структурам будет посвящена отдельная тема.

Наконец, заметим, что наиболее естественным выглядит сохранение статических структур в бинарных файлах данных (так как размер такой структуры в байтах – фиксированный) и обмен данными с этими файлами с помощью методом `fread/fwrite` из `stdio.h` или `read/write` из `fstream`:

```
//Чтение 20 записей типа student из бинарного файла
for (int i=0; i<20; i++)
 fread (&group[i],sizeof(student),1,f);
//Перейти к записи номер k (нумерация с нуля):
```

```
long int pos = sizeof(student)*k;
fsetpos (f,&pos);
```

К структурам с указателями в общем случае сказанное неприменимо. В качестве альтернативы напишем программу, позволяющую прочитать массив структур `student` с динамическим полем `name` из текстового файла `data.txt` следующего вида:

```
Ivanov 19 4.2
Petrova 19 3.9
Popov 20 4.1
```

и т.д. Предполагается, что файл находится в текущей для исполняемого файла приложения папке. Максимальное число записей пока что ограничим константой `ST`.

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;
#define ST 10

struct student {
 char *name; int age;   float ball;
};

int main() {
 student st[ST];
 FILE *f = fopen ("data.txt","rt");
 if (!f) {
  cout << "Can't open data.txt in current folder!";
  cin.get(); cin.sync(); return 1;
 }
 char buf[80],name[80];
 int i,k=0;
 while (1) {
  if (k>ST-1) break;
  fgets (buf, 79, f);
  int i=sscanf (buf,"%s %d %f",name,&st[k].age,&st[k].ball);
  if (i!=3) {
   cout << "Bad line " << (k+1) << "in file, skipped" << endl;
   break;
  }
  st[k].name = new char [strlen(name)+1];
  strcpy (st[k].name, name);
  k++;
  if (feof(f)) break;
  buf[0]='\0';
 }
 cout << "All data:" << endl;
 for (i=0; i<k; i++)
  cout << st[i].name << ", " << st[i].age << ", " << st[i].ball << endl;
```

```
 cin.get(); cin.sync(); return 0;
}
```

Программа также содержит простейшую защиту от неверных данных в исходном файле.

Оглавление серии

теги: c++ учебное

Здесь можно оставить комментарий, обязательны только выделенные цветом поля. Не пишите лишнего, и всё будет хорошо

Ваше имя:

Пароль (если желаете запомнить имя):

Любимый URL (если указываете, то вставьте полностью):

Текст сообщения (до 1024 символов):

Введите код сообщения: **3**3$9_1$

Добавить Сброс

12.11.2015, 17:39; рейтинг: 11775