Kiểu cấu trúc trong C++

Cấu trúc dữ liệu

Một cấu trúc dữ liệu là một tập hợp của những kiểu dữ liệu khác nhau được gộp lại với một cái tên duy nhất. Dạng thức của nó như sau:

```
struct model_name {
type1 element1;
    type2 element2;
    type3 element3;
    . .
} object_name;
```

trong đó `model_name` là tên của mẫu kiểu dữ liệu và tham số tùy chọn *object_name* một tên hợp lệ cho đối tượng. Bên trong cặp ngoặc nhọn là tên các phần tử của cấu trúc và kiểu của chúng.

Nếu định nghĩa của cấu trúc bao gồm tham số *model_name* (tuỳ chọn), tham số này trở thành một tên kiểu hợp lệ tương đương với cấu trúc.

Ví dụ:

```
struct products {
char name [30];
    float price;
};
products apple;
products orange, melon;
```

Chúng ta đã định nghĩa cấu trúc **products** với hai trường: **name** và **price**, mỗi trường có một kiểu khác nhau. Chúng ta cũng đã sử dụng tên của kiểu cấu trúc (**products**) để khai báo ba đối tượng có kiểu đó : **apple**, **orange** và **melon**.

Sau khi được khai báo, **products** trở thành một tên kiểu hợp lệ giống các kiểu cơ bản như *int*, *char* hay *short*.

Trường tuỳ chọn *object_name* có thể nằm ở cuối của phần khai báo cấu trúc dùng để khai báo trực tiếp đối tượng có kiểu cấu trúc.

Ví dụ, để khai báo các đối tượng **apple**, **orange** và **melon** như đã làm ở phần trước chúng ta cũng có thể làm theo cách sau:

```
struct products {
  char name [30];
  float price;
} apple, orange, melon;
```

Hơn nữa, trong trường hợp này tham số *model_name* trở thành tuỳ chọn. Mặc dù nếu *model_name* không được sử dụng thì chúng ta sẽ không thể khai báo thêm các đối tượng có kiểu mẫu này.

Một điều quan trọng là cần phân biệt rõ ràng đâu là **kiểu mẫu** cấu trúc, đâu là **đối tượng** cấu trúc. Nếu dùng các thuật ngữ chúng ta đã sử dụng với các biến, kiểu mẫu là tên kiểu dữ liệu còn đối tượng là các biến.

Sau khi đã khai báo ba đối tượng có kiểu là một mẫu cấu trúc xác định (**apple**, **orange** and **melon**) chúng ta có thể thao tác với các trường tạo nên chúng. Để làm việc này chúng ta sử dụng một dấu chấm (.) chèn ở giữa tên đối tượng và tên trường.

Ví dụ, chúng ta có thể thao tác với bất kì phần tử nào của cấu trúc như là đối với các biến chuẩn:

```
apple.name
apple.price
orange.name
orange.price
melon.name
melon.price
```

mỗi trường có kiểu dữ liệu tương ứng: **apple.name**, **orange.name** và **melon.name** có kiểu **char[30]**, và **apple.price**, **orange.price** và **melon.price** có kiểu **float**.

Chúng ta tạm biệt apples, oranges và melons để đến với một ví dụ về các bộ phim:

```
// example about structures
#include <iostream.h>
#include <string.h>
#include <stdlib.h>

struct movies_t {
  char title [50];
  int year;
} mine, yours;
```

```
void printmovie (movies_t movie);

int main ()
{
  char buffer [50];

  strcpy (mine.title, "2001 A Space Odyssey");
  mine.year = 1968;

  cout << "Enter title: ";
  cin.getline (yours.title,50);
  cout << "Enter year: ";
  cin.getline (buffer,50);
  yours.year = atoi (buffer);

  cout << "My favourite movie is:\n ";
  printmovie (mine);
  cout << "And yours:\n ";
  printmovie (yours);
  return 0;
}

void printmovie (movies_t movie)
{
  cout << movie.title;
  cout << " (" << movie.year << ")\n";
}
```

Kết quả:

```
Enter title: Alien
Enter year: 1979

My favourite movie is:
 2001 A Space Odyssey (1968)
And yours:
 Alien (1979)
```

Ví dụ này cho chúng ta thấy cách sử dụng các phần tử của một cấu trúc và bản thân cấu trúc như là các biến thông thường. Ví dụ, **`yours.year`** là một biến hợp lệ có kiểu **`int`** cũng như **`mine.title`** là một mảng hợp lệ với 50 phần tử kiểu *chars*.

Chú ý rằng cả **mine** and **yours** đều được coi là các biến hợp lệ kiểu **movie_t** khi được truyền cho hàm **printmovie()**.Hơn nữa một lợi thế quan trọng của cấu trúc là chúng ta có thể xét các phần tử của chúng một cách riêng biệt hoặc toàn bộ cấu trúc như là một khối.

Các cấu trúc được sử dụng rất nhiều để xây dựng cơ sở dữ liệu đặc biệt nếu chúng ta xét đến khả năng xây dựng các mảng của chúng.

```
// array of structures
#include <iostream.h>
#include <stdlib.h>

#define N_MOVIES 5

struct movies_t {
  char title[50];
  int year;
} films[N_MOVIES];

void printmovie (movies_t movie);

int main ()
{
  char buffer[50];
  int n;
  for (n=0; n<N_MOVIES; n++)
  {
    cout << "Enter title: ";
    cin.getline (films[n].title,50);
    cout << "Enter year: ";
    cin.getline (buffer,50);
    films[n].year = atoi (buffer);
  }
  cout << "\nYou have entered these movies:\n";
  for (n=0; n<N_MOVIES; n++)
    printmovie (films[n]);
  return 0;
}

void printmovie (movies_t movie)
{
  cout << movie.title;
```

```
  cout << " (" << movie.year << ")\n";
}
```

Kết quả:

```
Enter title: Alien
Enter year: 1979
Enter title: Blade Runner
Enter year: 1982
Enter title: Matrix
Enter year: 1999
Enter title: Rear Window
Enter year: 1954
Enter title: Taxi Driver
Enter year: 1975

You have entered these movies:
Alien (1979)
Blade Runner (1982)
Matrix (1999)
Rear Window (1954)
Taxi Driver (1975)
```

Con trỏ trỏ đến cấu trúc

Như bất kì các kiểu dữ liệu nào khác, các cấu trúc có thể được trỏ đến bởi con trỏ. Quy tắc hoàn toàn giống như đối với bất kì kiểu dữ liệu cơ bản nào:

```
struct movies_t {
char title[50];
int year;
};
```

Ở đây `amovie` là một đối tượng có kiểu `movies_t` và `pmovie` là một con trỏ trỏ tới đối tượng `movies_t`. OK, bây giờ chúng ta sẽ đến với một ví dụ khác, nó sẽ giới thiệu một toán tử mới:

```
// pointers to structures
#include <iostream.h>
#include <stdlib.h>

struct movies_t {
```

```
  char title[50];
  int year;
};


int main ()
{
  char buffer[50];

  movies_t amovie;
  movies_t * pmovie;
  pmovie = &amovie;

  cout << "Enter title: ";
  cin.getline (pmovie->title,50);
  cout << "Enter year: ";
  cin.getline (buffer,50);
  pmovie->year = atoi (buffer);

  cout << "\nYou have entered:\n";
  cout << pmovie->title;
  cout << " (" << pmovie->year << ")\n";

  return 0;
}
```

Kết quả:

```
Enter title: Matrix
Enter year: 1999

You have entered:
Matrix (1999)
```

Đoạn mã trên giới thiệu một điều quan trọng: toán tử **->**. Đây là một toán tử tham chiếu chỉ dùng để trỏ tới các cấu trúc và các lớp (class). Nó cho phép chúng ta không phải dùng ngoặc mỗi khi tham chiếu đến một phần tử của cấu trúc. Trong ví dụ này chúng ta sử dụng:

```
movies->title
```

nó có thể được dịch thành:

```
(*movies).title
```

cả hai biểu thức **movies->title** và **(*movies).title** đều hợp lệ và chúng đều dùng để tham chiếu đến phần tử **title** của cấu trúc được trỏ bởi **movies**. Bạn cần phân biệt rõ ràng với:

```
*movies.title
```

nó tương đương với

```
*(movies.title)
```

lệnh này dùng để tính toán giá trị được trỏ bởi phần tử **title** của cấu trúc **movies**, trong trường hợp này (title không phải là một con trỏ) nó chẳng có ý nghĩa gì nhiều. Bản dưới đây tổng kết tất cả các kết hợp có thể được giữa con trỏ và cấu trúc:

Biểu thức	**Mô tả**	**Tương đương với**
movies.title	Phần tử **title** của cấu trúc **movies**	
movies->title	Phần tử **title** của cấu trúc được trỏ bởi **movies**	(*movies).title
***movies.title**	Giá trị được trỏ bởi phần tử **title** của cấu trúc **movies**	*(movies.title)

Các cấu trúc lồng nhau

Các cấu trúc có thể được đặt lồng nhau vì vậy một phần tử hợp lệ của một cấu trúc có thể là một cấu trúc khác.

```
struct movies_t {
    char title[50];
int year;
}

struct friends_t {
    char name[50];
    char email[50];
    movies_t favourite_movie;
} charlie, maria;

friends_t * pfriends = &charlie;
```

Vì vậy, sau phần khai báo trên chúng ta có thể sử dụng các biểu thức sau:

```
charlie.name
maria.favourite_movie.title
charlie.favourite_movie.year
pfriends->favourite_movie.year
```

(trong đó hai biểu thức cuối cùng là tương đương).

Các khái niệm cơ bản về cấu trúc được đề cập đến trong phần này là hoàn toàn giống với ngôn ngữ C, tuy nhiên trong C++, cấu trúc đã được mở rộng thêm các chức năng của một lớp với tính chất đặc trưng là tất cả các phần tử của nó đều là công cộng (public).

Revision #1
Created 5 October 2019 15:53:11 by Laptrinh.vn
Updated 5 October 2019 16:01:19 by Laptrinh.vn