Bài 6: Tạo ra những biến kiểu riêng của bạn

Ngôn ngữ C cho phép chúng ta làm một số điều thật sự đặc biệt: tạo ra những biến kiểu riêng của bạn, ta có thể gọi là "những biến tự tạo". Hôm nay chúng ta sẽ học về **struct (cấu trúc)** và **kiểu liệt kê (enum)**.

Việc tạo kiểu biến của riêng bạn sẽ trở nên hữu ích khi các chương trình bắt đầu phức tạp hơn.

Hãy tập trung vì chúng ta sẽ tiếp tục sử dụng lại những kiến thức này trong những bài học tiếp theo.

Lưu ý rằng các thư viện thường định nghĩa theo kiểu riêng của chúng. Bạn sẽ không phải mất quá nhiều thời gian trước khi xử lý một kiểu biến, hoặc có thể sau này là những kiểu chương trình Window, Audio, Keyboard …

Định nghĩa một cấu trúc (struct):

Cấu trúc (struct) là một tập hợp gồm những phần tử có nhiều kiểu khác nhau. Không giống như khi làm việc với mảng (array), chúng ta được yêu cầu sử dụng cùng một kiểu định dạng đối với các phần tử trong toàn bộ mảng. Với cấu trúc (struct), bạn có thể tạo ra một tập hợp gồm các biến kiểu int, long, char hoặc là double…

Những cấu trúc thường được định nghĩa trong file.h, cũng giống như khi khai báo những nguyên mẫu (prototypes) hay những định nghĩa (defines).

Chúng ta cùng xem một VD:

C code:

```
struct TenCauTruc
{
   int bien1;
   int bien2;
   int bienKhac;
   double soThapphan;
};
```

Để khai báo một cấu trúc, chúng ta sẽ bắt đầu bằng từ khóa "**struct**", tiếp sau đó là tên đại diện cho tập hợp các phần tử (VD: sinhvien, taptin).

 Riêng cá nhân tôi thường hay áp dụng nguyên tắc đặt tên của biến để đặt tên cho cấu trúc, tôi thường viết hoa chữ cái đầu cho dễ nhận biết. Giả sử, nếu tôi thấy cụm "**tenSinhvien**" thì có nghĩa đó là tên của một biến bình thường vì chữ cái đầu của nó không được viết hoa. Tương tự khi thấy cụm "**TenSinhVien**", tôi biết đây là tên của một "biến tự tạo" trong cấu trúc.

Sau tên cấu trúc các bạn nhớ đóng mở ngoặc nhọn **{ }** giống khi thao tác với hàm.

Lưu ý một điều đặc biệt sau: Bạn phải đặt một dấu chấm phẩy (**;**) sau dấu đóng ngoặc nhọn. Điều này là bắt buộc vì nếu thiếu nó thì chương trình của bạn sẽ không thể biên dịch được.

Và những thứ nằm giữa 2 ngoặc nhọn đó có gì lạ ?

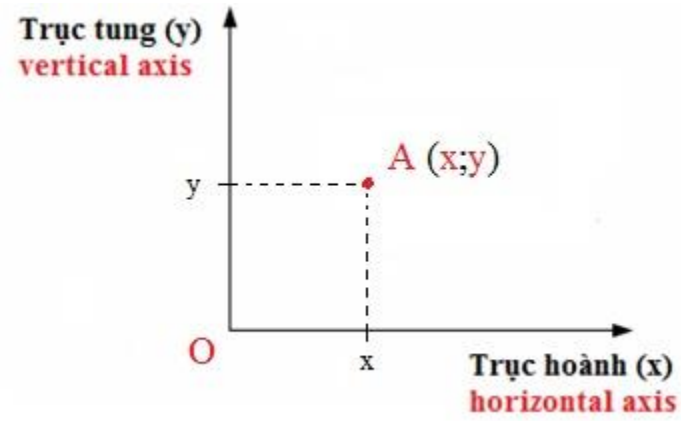
Đơn giản thôi, đó là những "biến thành phần" được tạo ra để xây dựng nên cấu trúc của bạn. Một cấu trúc thường có từ 2 biến trở lên, nhưng có lẽ bạn cũng không cần quan tâm quá nhiều về vấn đề này.

Như bạn đã biết, việc khởi tạo một biến cơ bản không quá phức tạp. Tất cả những cấu trúc mà bạn thấy thật ra chỉ là "tập hợp" của những biến kiểu cơ bản như *int, long,...* Chả có điều gì mới mẻ ở đây cả.

Ví dụ về cấu trúc (struct):

Giả sử bạn muốn tạo ra một biến để lưu tọa độ một điểm trên màn hình. Chắc chắn bạn sẽ cần đến một cấu trúc như thế này khi tạo ra các game 2D trong những phần tiếp theo, đây là cơ hội để nâng cao trình độ một chút.

Đối với những bạn chưa có nhiều kiến thức về những khái niệm "hình học không gian 2 chiều" thì sau đây là một số giải thích cơ bản cho hình học 2 chiều (2D).



Hệ trục tọa độ Oxy chứa điểm A

Khi làm việc với không gian 2 chiều (2D), chúng ta có 1 hệ trục tọa độ gồm:

- Gốc tọa độ O. Tại đây giá trị gốc bằng 0 và tăng dần theo chiều mũi tên của 2 trục tọa độ.
- Trục hoành (trục x) chạy từ trái sang phải, chứa các giá trị gọi là hoành độ.
- Trục tung (trục y) chạy từ dưới lên trên, chứa các giá trị gọi là tung độ.

Một điểm bất kỳ trên mặt phẳng tọa độ 2 chiều thường có 2 thông số để xác định vị trí của nó trên mặt phẳng. Hai thông số đó gồm hoành độ (giá trị trên trục x) và tung độ (giá trị trên trục y).

Chúng ta thường dùng một biến tên "x" để biểu diễn giá trị của hoành độ, tương tự như vậy ta sẽ dùng biến "y" để biểu diễn giá trị của tung độ.

Nếu viết điểm A (x;y) có nghĩa là điểm này có tên là A, hoành độ của nó là x và tung độ là y.

VD: ta có một điểm diembatky (**20**;**10**), có nghĩa là điểm này có tên là diembatky, hoành độ của nó là x=20 và tung độ là y=10.

Bạn có thể viết một cấu trúc Toadodiem để lưu trữ các giá trị hoành độ trên trục x và tung độ trên trục y của một điểm.

Nào nào, nó thật sự không khó đâu:

C code:

```
struct Toadodiem
{
   int x; // hoanh do cua diem
   int y; // tung do cua diem
};
```

Cấu trúc (srtuct) của chúng ta có tên là "Toadodiem" gồm có 2 biến "x" và "y" để lần lượt biểu diễn hoành độ trên trục x và tung độ trên trục y.

Nếu muốn bạn hoàn toàn có thể tạo ra một cấu trúc (struct) mới cho không gian 3 chiều (3D), chỉ việc thêm vào một biến "z" để biểu diễn tọa độ trên trục đó (thường gọi là cao độ). Với những kiến thức này, chúng ta có thể tạo ra một cấu trúc để quản lý các điểm trong không gian 3D.

Mảng cấu trúc (mảng struct):

Những cấu trúc có chứa mảng. Thật may mắn, chúng ta hoàn toàn có thể tạo các mảng cơ bản và mảng ký tự (string) trong cấu trúc.

Nào bây giờ giả sử chúng ta có một cấu trúc *Taikhoan* để lưu trữ thông tin của một người dùng:

C code:

```
struct Taikhoan
{
   char ten[100];
   char ho[100];
   char diachi[1000];

   int tuoi;
   int gioitinh; // Boolean : 1 = nam, 0 = nu
};
```

Cấu trúc Taikhoan chứa 5 biến thành phần, trong đó:

- 3 biến đầu tiên kiểu *char* lưu trữ các thông tin lần lượt là: tên, họ, địa chỉ.
- 2 biến còn lại kiểu *int* lưu trữ các thông tin: tuổi và giới tính. Riêng giới tính là một biến dạng boolean (bạn đã học ở các bài trước về boolean), biến này sẽ trả về 1 = đúng = giới tính là nam, trả về 0 = sai = giới tính là nữ, chúng ta tạm chia ra 2 giới tính thôi nhé ^^!

Bạn có thể ứng dụng cấu trúc này để tạo một chương trình lưu trữ danh sách người dùng. Dĩ nhiên là bạn có thêm một số biến khác để bổ sung những thông tin mà bạn muốn. Không có giới hạn số lượng biến trong một cấu trúc nên bạn đừng lo.

Sử dụng cấu trúc:

Bây giờ những cấu trúc của chúng ta đã được định nghĩa trong các file.h và chúng ta có thể sử dụng các function của chúng trong file.c

Vậy hãy cùng xem làm thế nào để tạo một biến mang kiểu Toadodiem (cấu trúc được tạo ở trên):

C code:

```
#include "main.h" // File.h chua cac prototypes va structs

int main (int argc, char *argv[ ])
{
   struct Toadodiem diembatky; // Khoi tao bien diembatky co kieu Toadodiem

   return 0;
}
```

Vừa rồi chúng ta đã tạo ra một biến "diembatky" mang kiểu biến "Toadodiem". Biến này sẽ tự động bao gồm luôn 2 biến thành phần x và y (hoành độ và tung độ) mà ta đã khai báo trước đó.

 Vậy chúng ta có bắt buộc phải thêm từ khóa "struct" mỗi lần khai báo biến không ?

Câu trả lời là CÓ: Điều này sẽ giúp máy tính phân biệt các biến tự tạo (VD như biến kiểu Toadodiem) với những biến cơ bản (VD như biến kiểu int).

Tuy nhiên các lập trình viên thường cảm thấy lười khi phải luôn thêm từ khóa "struct" mỗi khi khai báo các biến tự tạo. Để giải quyết vấn đề này, họ đã phát minh ra một lệnh đặc biệt, họ gọi nó là typedef.

Typedef:

Trở lại với những file.h có chứa những định nghĩa cấu trúc Toadodiem của chúng ta.

Chúng ta sẽ thêm vào một câu lệnh gọi là typedef để tạo ra một tên cấu trúc thay thế cho toàn bộ cấu trúc đó.

Bây giờ chúng ta sẽ thêm vào một dòng trước khi khai báo cấu trúc ở đầu đoạn code lúc nãy:

C code:

```
typedef struct Toadodiem Toadodiem;
struct Toadodiem
{
   int x; // hoanh do cua diem
   int y; // tung do cua diem
};
```

Tôi sẽ giải thích cho bạn về dòng mới được thêm vào này, nó sẽ được chia làm 3 phần chính (nói thêm với bạn là tôi không hề mắc lỗi khi lặp lại cụm Toadodiem 2 lần).

1. typedef: sẽ chỉ ra cho máy tính biết rằng chúng ta đang đặt một tên thay thế cho cấu trúc.
2. struct Toadodiem: đây là tên của cấu trúc mà bạn sẽ đặt tên thay thế với typedef.
3. Toadodiem: đây chính là tên mà bạn đặt để thay thế cho cấu trúc struct Toadodiem. Bạn có thể đặt một tên bất kỳ mà bạn thích, tôi đặt là Toadodiem để cho các bạn thấy rằng khi dùng typedef, bạn sẽ tạo ra một cụm từ thay thế cho cấu trúc với chức năng tương đương.

Rõ ràng điều này có nghĩa là khi bạn viết cụm từ Toadodiem thì nó sẽ thay thế cho toàn bộ cấu trúc struct Toadodiem. Bằng cách này, bạn sẽ không phải đặt cụm struct Toadodiem mỗi khi khai báo biến tự tạo của mình nữa.

Vậy bây giờ chúng ta sẽ viết lại đoạn code trong main.c sau khi đã dùng lệnh typedef nhé:

C code:

```
int main (int argc, char *argv[ ])
{
   Toadodiem diembatky; /* May tinh se hieu bien nay mang kieu cau truc Toadodiem sau khi da duoc dat ten thay the boi typedef */

   return 0;
}
```

Tôi khuyến khích các bạn nên tập thói quen dùng typedef với các cấu trúc giống như cách tôi đã làm với cấu trúc *Toadodiem* trong bài học này. Hầu hết các lập trình viên đều làm như vậy. Việc này sẽ giúp các bạn tiết kiệm thời gian khi không phải viết lại nhiều lần từ "struct" trong cả đoạn code. Có một điều lạ là hình như một lập trình viên giỏi thì thường khá lười.

Chỉnh sửa các thành phần của cấu trúc:

Bây giờ thì biến diembatky đã được khởi tạo, và nếu chúng ta muốn thay đổi những thành phần trong nó thì sao. Làm thế nào để tác động vào biến x và y, nào cùng xem thử nhé:

C code:

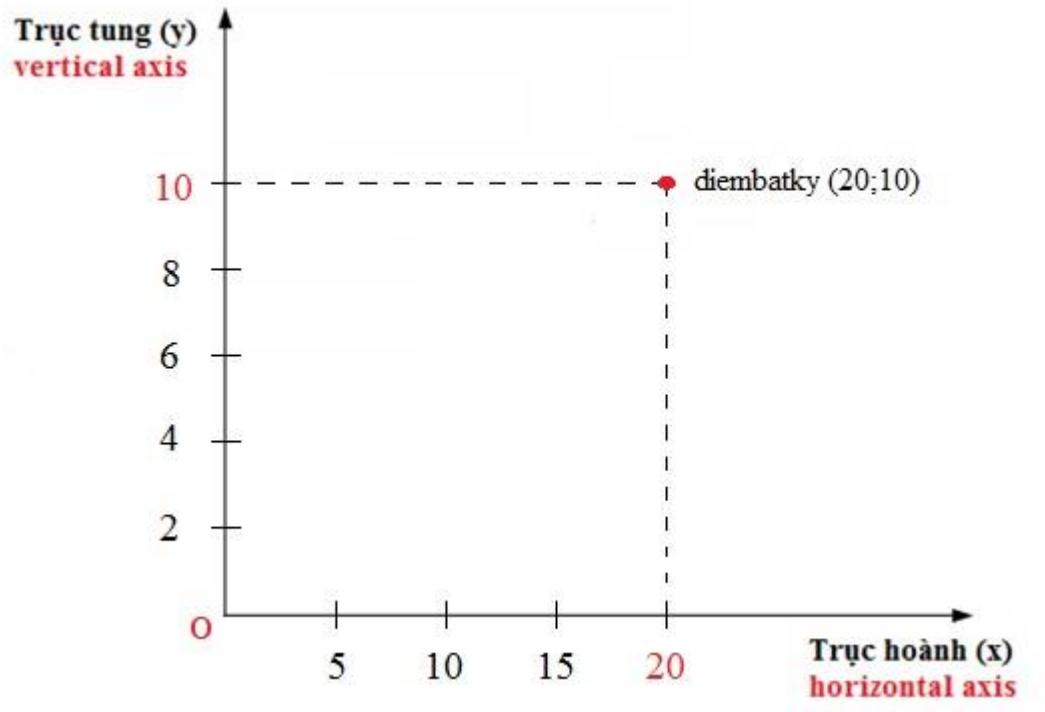
```
int main (int argc, char *argv[ ])
{
   Toadodiem diembatky;

   diembatky.x = 10;
   diembatky.y = 20;

   return 0;
}
```

Bây giờ thì giá trị của biến diembatky đã được thay đổi, chúng ta đã cho nó một giá trị hoành độ x=20 và tung độ y=10. Bây giờ diembatky của chúng ta đang ở tọa độ (20;10).

Tôi sẽ minh họa một chút bằng mặt phẳng tọa độ Oxy trong hình học 2 chiều để các bạn có cái nhìn trực quan hơn:



Vậy tóm lại, để truy cập vào các biến thành phần của cấu trúc chúng ta sẽ viết theo cách sau:

C code:

```
tenbientutao.tenBienThanhPhanTrongCauTruc
```

Biến tự tạo **diembatky** tách biệt với biến thành phần **x** và **y** trong cấu trúc **Toadodiem.**

Sử dụng cấu trúc Taikhoan mà chúng ta đã tạo ở đầu bài và viết chương trình yêu cầu người dùng nhập tên, họ sau đó in ra màn hình.

Đoạn code sẽ như sau:

C code:

```
int main (int argc, char *argv[ ])
{
   Taikhoan nguoidung;

   printf ("Ten ban la gi ? ");
   scanf ("%s", nguoidung.ten);
   printf ("Ho cua ban la gi ? ");
   scanf ("%s", nguoidung.ho);

   printf ("Ho ten day du cua ban la %s %s", nguoidung.ho, nguoidung.ten);

   return 0;
}
```

Console:

```
Ten ban la gi ? Nhan
Ho cua ban la gi ? Sieu
Ho ten day du cua ban la Sieu Nhan
```

Chúng ta dùng hàm scanf để nhập giá trị cho biến nguoidung.ten, có nghĩa là lúc này giá trị đó sẽ được truyền thẳng vào biến ten của nguoidung. Bạn chỉ việc thực hiện tương tự như vậy đối với họ, tuổi, giới tính.

Dĩ nhiên là bạn cũng không cần phải học về cấu trúc (struct) thì mới có thể viết được chương trình trên. Mọi việc bạn cần làm chỉ đơn giản là tạo ra các biến lưu trữ tên, họ … giống như các bài học mà ta đã học trước đây.

Nhưng điều thú vị ở struct là bạn có thể tạo ra những kiểu biến riêng cho từng trường hợp.

Giả sử bạn có một game dành cho 2 người:

C code:

```
Taikhoan nguoichoi1, nguoichoi2;
```

… Bạn thấy không, nhờ nó mà người chơi sẽ có thể lưu trữ tên, họ … những thông tin riêng của từng người.

Tuy nhiên, chúng ta còn có thể làm tốt hơn, thậm chí chúng ta còn có thể tạo một mảng cho cấu trúc đó. Rất đơn giản thôi:

C code:

```
Taikhoan nguoichoi[2];
```

Để khai báo biến tên người chơi ở vị trí thứ 0 trong mảng trên thì bạn sẽ viết như sau:

C code:

```
nguoichoi[0].ten
```

Lợi ích của việc sử dụng mảng ở đây là bạn có thể tạo ra một vòng lặp để yêu cầu nhập thông tin người chơi thứ 1 và thứ 2 nhưng không cần phải viết đoạn code đó 2 lần. Chỉ cần tham chiếu vào từng thành phần của mảng để yêu cầu nhập từng tên, họ, địa chỉ …

Bài Tập: Hãy tạo một mảng và sử dụng vòng lặp để yêu cầu người chơi nhập vào những thông tin khác. Hãy bắt đầu với 2 người chơi, nhưng sau khi đã nắm vững bạn có thể mở rộng hơn theo ý thích của bạn. Sau cùng hiển thị ra màn hình tất cả những thông tin mà bạn đã thu thập được từ những người chơi.

Khởi tạo một cấu trúc:

Đối với những cấu trúc cũng giống như các biến, mảng hay con trỏ, tôi khuyến khích các bạn nên khởi tạo chúng "không chứa giá trị nào" ngay từ đầu. Tôi nói thật đấy, hãy để tôi nhắc lại một chút, khi một biến bình thường được tạo ra, nó sẽ mang giá trị bất kỳ của ô địa chỉ được máy tính cấp phát cho nó. Đôi khi biến mang giá trị 0, nhưng đôi khi nó lại mang một giá trị rác của một chương trình khác đã sử dụng trước đó, những giá trị rác này thường không có ý nghĩa (chẳng hạn như: -69,69).

Sẵn tiện tôi sẽ nhắc lại một chút về các cách khởi tạo:

- Biến: chúng ta thường cho nó mang giá trị 0 lúc đầu (trong những trường hợp đơn giản).
- Con trỏ: chúng ta thường đặt giá trị là **NULL**. **NULL** là một định nghĩa có sẵn trong thư viện *stdib.h* nói chung nó cũng có nghĩa là giá trị 0, nhưng chúng ta vẫn sử dụng NULL để biết được rằng đó là con trỏ chứ không phải biến bình thường.
- Mảng: chúng ta thường đặt giá trị 0 cho mỗi phần tử của mảng.

Đối với cấu trúc (struct), việc khởi tạo sẽ có một chút giống với mảng. Thật vậy, chúng ta cùng xem khi khởi tạo một biến cấu trúc thì như thế nào nhé:

C code:

```
Toadodiem diembatky = {0, 0};
```

Sau khi khởi tạo như trên máy tính sẽ tự động gửi giá trị lần lượt vào diembatky.x = 0 và diembatky.y = 0.

Quay trở lại với cấu trúc Taikhoan (có chứa chuỗi ký tự). Bạn có thể bắt đầu tạo ra một chuỗi trong cấu trúc bằng cách viết cặp dấu ngoặc kép " " (không có thành phần nào chứa bên trong cặp dấu này). Tôi chưa nói cho bạn biết về nó ở những bài học về chuỗi trước đây, nhưng bây giờ vẫn chưa muộn để biết về nó.

Nhờ đó, chúng có thể khai báo trong cấu trúc Taikhoan những thông tin như ten, ho, diachi, tuoi, gioitinh như sau:

C code:

```
Taikhoan nguoidung = {"", "", "", 0, 0};
```

Cá nhân tôi không thường sử dụng cách này. Tôi thích sử dụng một hàm *taoToadodiem* với chức năng khởi tạo các biến cho biến *diembatky* của tôi.

Để làm được việc đó bạn phải tạo ra biến con trỏ. Nếu tôi chỉ sử dụng biến bình thường, một bản sao sẽ được tạo ra trong hàm (nó không phải là biến bạn đã tạo) và hàm sẽ thay đổi giá trị của bản sao đó chứ không phải giá trị của biến mà bạn đã khởi tạo. Nếu cảm thấy khó hiểu đoạn này, bạn có thể xem lại bài học cũ về con trỏ.

Vì vậy, bây giờ chúng ta sẽ phải học cách sử dụng con trỏ (pointer) trong cấu trúc (struct). Mọi thứ bắt đầu có chút thú vị rồi đây.

Con trỏ cấu trúc (pointer of struct):

Một con trỏ cấu trúc được tạo ra theo cách tương tự như một con trỏ kiểu int, double hay bất kỳ kiểu cơ bản nào khác:

C code:

```
Toadodiem* diembatky = NULL;
```

Chúng ta vừa khởi tạo một con trỏ cho cấu trúc Toadodiem, con trỏ có tên là diembatky.

Để tránh làm các bạn bỡ ngỡ tôi xin phép nói thêm rằng bạn vẫn có thể đặt dấu * trước tên con trỏ như chúng ta thường làm trước đây, nó sẽ như thế này:

C code:

```
Toadodiem *diembatky = NULL;
```

Tôi vẫn thường làm như cách trên, vì trong trường hợp để định nghĩa nhiều con trỏ trên cùng một dòng thì bạn phải đặt dấu * trước mỗi con trỏ đó. VD:

C code:

```
Toadodiem *diembatky1 = NULL, *diembatky2 = NULL;
```

Gửi một hàm vào cấu trúc:

Những gì chúng ta quan tâm ở đây là làm sao để sử dụng một con trỏ cấu trúc trong hàm, từ đó ta có thể thay đổi trực tiếp giá trị của biến.

Chúng ta sẽ thử với vd này: Chúng ta chỉ cần đơn giản tạo ra một biến *Toadodiem* và sau đó gửi địa chỉ của nó vào hàm *taoToadodiem*. Hàm này sẽ qui định các thành phần có giá trị 0.

Hàm *taoToadodiem* sẽ cần một tham số (parameter): tham số đó là một con trỏ đến cấu trúc Toadodiem (a *Toadodiem):

C code:

```
int main (int argc, char *argv[ ])
{
   Toadodiem diembatkyCuatoi;

   taoToadodiem(&diembatkyCuatoi);

   return 0;
}

void taoToadodiem(Toadodiem* diembatky)
{
   // Tao cac bien thanh phan cua cau truc o day
}
```

Biến *diembatkyCuatoi* đã được tạo ra và địa chỉ của nó sẽ được gửi vào hàm *taoToadodiem*, chúng ta gọi biến này là con trỏ (bạn có thể đặt tên nó như thế nào tùy ý, điều này không ảnh hưởng đến hàm).

Nào bây giờ đối với hàm *taoToadodiem*, chúng ta sẽ lần lượt khởi tạo giá trị cho các thành phần.

Đừng quên đặt dấu * trước tên của con trỏ để truy cập vào các biến của nó. Nếu thiếu dấu * bạn sẽ chỉ thay đổi địa chỉ con trỏ và đó không phải là điều chúng ta muốn máy tính thực hiện.

Ok, nhưng mà có một vấn đề ... chúng ta thật sự không thể làm được:

C code:

```
void taoToadodiem(Toadodiem* diembatky)
{
   *diembatky.x = 0;
   *diembatky.y = 0;
}
```

Trông thật đơn giản ... nhưng tại sao chúng ta lại không thể làm điều đó? Nguyên nhân là vì dấu chấm phân cách chỉ làm việc với các ký tự, nó không hiểu dấu * là gì. Nhưng chúng ta cần sử dụng dấu * để truy cập vào và thay đổi giá trị của biến.

Giải pháp cho vấn đề này là chúng ta sẽ đặt một cặp ngoặc đơn để bao phần dấu sao và trước dấu chấm ngăn cách lại. Lúc này chúng ta có thể truy cập vào và thay đổi giá trị của biến:

C code:

```
void taoToadodiem(Toadodiem* diembatky)
{
   (*diembatky).x = 0;
   (*diembatky).y = 0;
}
```

Đoạn code trên đã hoạt động, bạn có thể kiểm tra thử. Những biến kiểu cấu trúc Toadodiem đã được đưa vào hàm và khởi tạo cho chúng giá trị x=0, y=0.

Trong ngôn ngữ C, chúng ta thường khởi tạo các cấu trúc theo cách đơn giản mà ta đã thấy ở trên.

Tuy nhiên, đối với C++, việc khởi tạo thường được thiết lập trong các hàm.

C++ thật sự không có gì khác ngoài một sự "cải tiến" của C. Tất nhiên là có rất nhiều điều để nói về ngôn ngữ này, ít thì cũng tốn cả 1 cuốn sách để viết về nó, và chúng ta sẽ không thể học tất cả cùng một lúc bây giờ.

Một phím tắt thường được sử dụng phổ biến:

Bạn sẽ thấy rằng con trỏ được sử dụng rất thường xuyên. Thẳng thắn mà nói, ngôn ngữ C hầu như chỉ sử dụng những cấu trúc con trỏ. Tôi đang nói với bạn về điều này một cách rất nghiêm túc (không hề cười nhé, hehe)!

Như đã nói ở trên, khi sử dụng cấu trúc con trỏ thì ta phải viết thế này:

C code:

```
   (*diembatky).x = 0;
```

Nhưng những nhà lập trình viên thiên tài vẫn thấy cách này chưa đủ nhanh, họ cảm thấy khó chịu với những cặp dấu ngoặc đơn. Ngay sau đó, những vị lười biếng thông minh này đã sáng tạo ra phím tắt sau đây để thay thế:

C code:

```
*diembatky -> x = 0;
```

Phím tắt này mô phỏng hình ảnh của một mũi tên, nó là sự kết hợp của một dấu trừ (-) và một dấu lớn (>).

 Khi chúng ta viết diembatky -> x cũng tương đương với (*diembatky).x

Nhớ rằng bạn có thể sử dụng mũi tên (->) khi thao tác với con trỏ và nếu làm việc trực tiếp với các biến, bạn phải sử dụng dấu chấm (.) như chúng ta đã học ở đầu bài.

Nào hãy thử áp dụng những phím tắt vừa rồi vào hàm *taoToadodiem* xem như thế nào:

C code:

```
void taoToadodiem(Toadodiem* diembatky)
{
   *diembatky->.x = 0;
   *diembatky->.y = 0;
}
```

Hãy nhớ rõ cách dùng phím tắt mũi tên này, chúng ta sẽ còn dùng lại nó nhiều lần nữa. Cẩn thận đừng nhầm lẫn giữa việc dùng mũi tên (->) với dấu chấm (.). Mũi tên là dành cho con trỏ, và dấu chấm là dành riêng cho biến.

Cùng xem một ví dụ nhỏ để phân biệt rõ hơn giữa chúng nhé:

C code:

```
int main (int argc, char *argv[ ])
{
   Toadodiem diembatkyCuatoi;
   Toadodiem *contro = &diembatkyCuatoi;

   diembatkyCuatoi.x = 10; // Lam viec voi mot bien ta su dung dau cham
   contro -> x = 10; // Lam viec voi mot con tro ta su dung mui ten
}
```

Giá trị của x được gán bằng 10 theo 2 cách: đầu tiên ta làm việc trực tiếp trên biến, lần thứ hai ta làm việc thông qua con trỏ.

Kiểu liệt kê (enum):

Kiểu liệt kê (enum) là một cách hơi khác để tạo ra các biến riêng của bạn. Kiểu liệt kê (enum) không bắt buộc phải chứa các "biến thành phần" như cấu trúc (struct). Đây là một danh sách "các giá trị phù hợp" cho một biến. Do đó kiểu enum sẽ chiếm 1 địa chỉ bộ nhớ để dùng cho các giá giá trị mà bạn xác định (và chỉ duy nhất mỗi lần 1 giá trị).

Vd:

C code:

```
typedef enum Volume Volume;
enum Volume
{
    LOW, MEDIUM, HIGH
}
```

Bạn thấy rằng chúng ta lại sử dụng typedef như đã dùng trước đó.

Để tạo một danh sách liệt kê, chúng ta dùng cụm enum. Danh sách của chúng ta tên là Volume (cái này trên tivi người ta hay gọi là âm lượng đó mà). Đây là một biến tự tạo giúp chúng ta chọn ra 1 trong 3 giá trị đã được chỉ ra LOW hoặc MEDIUM hoặc HIGH.

Bây giờ chúng ta đã có thể tạo ra một biến Volume (âm lượng) để điều chỉnh độ lớn của âm thanh khi nghe nhạc trên máy tính.

Đây là một ví dụ, khởi tạo biến âm lượng vừa cho nhạc:

C code:

```
Volume music = MEDIUM;
```

Sau này chúng ta vẫn có thể yêu cầu máy tính thay đổi giá trị của âm lượng thành HIGH hoặc LOW.

Sự kết hợp của những giá trị:

Bạn có thấy rằng tôi đã viết IN HOA những giá trị trong danh sách liệt kê. Việc này gợi cho các bạn nhớ về những hằng số (constants) và định nghĩa (defines) đúng không ?

Thật vậy, nó cũng gần như tương tự nhưng không hoàn toàn giống hẳn.

Trình biên dịch sẽ tự động gán một số cho từng giá trị trong danh sách liệt kê.

Trong trường hợp danh sách liệt kê Volume của chúng ta, LOW mang giá trị 0, MEDIUM và HIGH lần lượt mang giá trị 1 và 2. Sự liên kết được máy tính tự động sắp xếp, và nó luôn bắt đầu bằng giá trị 0.

Không giống như #define là một trình biên dịch tạo ra MEDIUM 1, kiểu liệt kê không phải là một tiền xử lý. Nó chỉ gần giống thôi.

Thực tế là khi bạn đã khởi tạo một biến music = MEDIUM, chương trình sẽ đặt giá trị 1 vào ô bộ nhớ đó.

 Vậy, có lợi ích gì khi biết trước giá trị của MEDIUM là 1 và HIGH là 2 … ?

Theo tôi thì không, chúng ta không cần quan tâm tới việc này. Trình biên dịch sẽ tự động liên kết các giá trị và đặt chúng vào từng biến. Chúng ta chỉ cần viết như sau:

C code:

```
if (music == MEDIUM)
{
   // Am thanh cua chuong trinh choi nhac se theo gia tri cua Volume
}
```

Đừng để ý đến giá trị của MEDIUM là bao nhiêu, bạn cứ để cho trình biên dịch tự động quản lý các số giá trị trong danh sách.

Lợi ích của việc này là gì? Việc này giúp cho code của bạn dễ đọc hơn. Thật vậy, mọi người có thể dễ dàng đọc hiểu trước If của bạn (điều kiện được hiểu là nếu biến music là MEDIUM thì chơi nhạc ở mức vừa).

Gán một giá trị cụ thể:

Từ bây giờ, trình biên dịch sẽ tự động đặt số 0 cho giá trị thứ nhất và lần lượt tiếp theo là 1, 2 ,3 ... theo thứ tự.

Nó có thể yêu cầu gán từng con số cụ thể cho mỗi giá trị thành phần của kiểu liệt kê.

Đó là những gì thú vị từ nó sao? Nào để tôi cho bạn thấy, giả sử trên máy tính của bạn âm lượng (volume) được qui định các mức từ 0 đến 100 (mức 0 có nghĩa là câm nín và 100 tức là hát điếc cả tai). Đây chính là cơ hội để chúng ta thử gán giá trị cho các thành phần trong kiểu liệt kê.

C code:

```
typedef enum Volume Volume;
enum Volume
{
    LOW = 10, MEDIUM = 50, HIGH = 100
}
```

Như các bạn thấy, mức volume LOW bằng 10% mức volume của máy tính, MEDIUM thì bằng 50% và tương tự HIGH là 100%. Người ta cũng có thể tạo ra một giá trị mới tên MUTE (câm nín). Chúng ta sẽ gán số 0 cho giá trị này. Bạn đã hiểu ra vấn đề đúng không.

Tổng kết:

- Cấu trúc (struct) là một kiểu biến tự tạo, bạn có thể tự mình tạo ra nó và sử dụng trong chương trình của bạn. Biến này do bạn định nghĩa, không như các kiểu biến cơ bản như int, double …

- Một cấu trúc luôn chứa những "biến thành phần", những biến thành phần này là các biến cơ bản kiểu int, double …, tương tự như với mảng nhưng cấu trúc có thể chứa nhiều biến khác kiểu.

- Khi bạn muốn truy cập vào một biến thành phần trong cấu trúc, bạn có thể sử dụng dấu chấm để ngăn cách tên của biến bình thường và tên của biến thành phần của cấu trúc, VD: bienbinhthuong.bienthanhphan1

- Nếu bạn muốn sử dụng con trỏ để truy cập vào biến thành phần, bạn chỉ cần thay dấu chấm thành mũi tên (->), VD: contro -> bienthanhphan1

- Một kiểu liệt kê cũng là biến tự tạo, bạn có thể đưa vào đây danh sách các giá trị như trong VD trên: LOW, MEDIUM, HIGH.
