Kiểu dữ liệu do người dùng định nghĩa trong C++

Tự định nghĩa các kiểu dữ liệu (typedef)

C++ cho phép chúng ta định nghĩa các kiểu dữ liệu của riêng mình dựa trên các kiểu dữ liệu đã có. Để có thể làm việc đó chúng ta sẽ sử dụng từ khoá `typedef`, dạng thức như sau:

```
typedef existing_type new_type_name;
```

Trong đó `existing_type` là một kiểu dữ liệu cơ bản hay bất kì một kiểu dữ liệu đã định nghĩa và new_type_name là tên của kiểu dữ liệu mới.

Ví dụ:

```
typedef char C;
typedef unsigned int WORD;
typedef char * string_t;
typedef char field [50];
```

Trong trường hợp này chúng ta đã định nghĩa bốn kiểu dữ liệu mới: C, WORD, string_t và field kiểu char, unsigned int, char* kiểu char[50], chúng ta hoàn toàn có thể sử dụng chúng như là các kiểu dữ liệu hợp lệ:

```
C achar, anotherchar, *ptchar1;
WORD myword;
string_t ptchar2;
field name;
```

`typedef` có thể hữu dụng khi bạn muốn định nghĩa một kiểu dữ liệu được dùng lặp đi lặp lại trong chương trình hoặc kiểu dữ liệu bạn muốn dùng có tên quá dài và bạn muốn nó có tên ngắn hơn.

Union

Union cho phép một phần bộ nhớ có thể được truy xuất dưới dạng nhiều kiểu dữ liệu khác nhau mặc dù tất cả chúng đều nằm cùng một vị trí trong bộ nhớ. Phần khai báo và sử dụng nó tương tự với cấu trúc nhưng chức năng thì khác hoàn toàn:

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

Tất cả các phần tử của union đều chiếm cùng một chỗ trong bộ nhớ. Kích thước của nó là kích thước của phần tử lớn nhất. Ví dụ:

```
union mytypes_t {
  char c;
  int i;
  float f;
} mytypes;
```

định nghĩa ba phần tử:

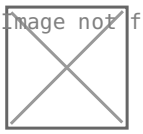
```
mytypes.c
mytypes.i
mytypes.f
```

mỗi phần tử có một kiểu dữ liệu khác nhau. Nhưng vì tất cả chúng đều nằm cùng một chỗ trong bộ nhớ nên bất kì sự thay đổi nào đối với một phần tử sẽ ảnh hưởng tới tất cả các thành phần còn lại.

Một trong những công dụng của union là dùng để kết hợp một kiểu dữ liêu cơ bản với một mảng hay các cấu trúc gồm các phần tử nhỏ hơn. Ví dụ:

```
union mix_t{
  long l;
  struct {
    short hi;
    short lo;
  } s;
  char c[4];
} mix;
```

Định nghĩa ba phần tử cho phép chúng ta truy xuất đến cùng một nhóm 4 byte: mix.l, mix.s và mix.c mà chúng ta có thể sử dụng tuỳ theo việc chúng ta muốn truy xuất đến nhóm 4 byte này như thế nào.



Unions vô danh

Trong C++ chúng ta có thể sử dụng các unions vô danh. Nếu chúng ta đặt một union trong một cấu trúc mà không đề tên (phần đi sau cặp ngoặc nhọn { }) union sẽ trở thành vô danh và chúng ta có thể truy xuất trực tiếp đến các phần tử của nó mà không cần đến tên của union (có cần cũng không được). Ví dụ, hãy xem xét sự khác biệt giữa hai phần khai báo sau đây:

Union:

```
struct {
  char title[50];
  char author[50];
  union {
    float dollars;
    int yens;
  } price;
} book;
```

Union vô danh:

```
struct {
  char title[50];
  char author[50];
  union {
    float dollars;
    int yens;
  };
} book;
```

Sự khác biệt duy nhất giữa hai đoạn mã này là trong đoạn mã đầu tiên chúng ta đặt tên cho union (**price**) còn trong cái thứ hai thì không. Khi truy nhập vào các phần tử **dollars** và **yens**, trong trường hợp thứ nhất chúng ta viết:

```
book.price.dollars
book.price.yens
```

còn trong trường hợp thứ hai:

```
book.dollars
book.yens
```

Một lần nữa tôi nhắc lại rằng vì nó là một union, hai trường **dollars** và **yens** đều chiếm cùng một chỗ trong bộ nhớ nên chúng không thể giữ hai giá trị khác nhau.

Kiểu liệt kê (enum)

Kiểu dữ liệu liệt kê dùng để tạo ra các kiểu dữ liệu chứa một cái gì đó hơi đặc biệt một chút, không phải kiểu số hay kiểu kí tự hoặc các hằng **true** và **false**. Dạng thức của nó như sau:

```
enum model_name {
  value1,
    value2,
    value3,
    . .
} object_name;
```

Ví dụ, chúng ta có thể tạo ra một kiểu dữ liệu mới có tên **color** để lưu trữ các màu với phần khai báo như sau:

```
enum colors_t {
  black, blue, green, cyan, red, purple, yellow, white
};
```

Chú ý rằng chúng ta không sử dụng bất kì một kiểu dữ liệu cơ bản nào trong phần khai báo. Chúng ta đã tạo ra một kiểu dữ liệu mới mà không dựa trên bất kì kiểu dữ liệu nào có sẵn: kiểu **color_t**, những giá trị có thể của kiểu **color_t** được viết trong cặp ngoặc nhọn {}. Ví dụ, sau khi khai báo kiểu liệt kê, biểu thức sau sẽ là hợp lệ:

```
colors_t mycolor;
```

Trên thực tế kiểu dữ liệu liệt kê được dịch là một số nguyên và các giá trị của nó là các hằng số nguyên được chỉ định. Nếu điều này không đực chỉ định, giá trị nguyên tương đương với phần tử đầu tiên là **0** và các giá trị tiếp theo cứ thế tăng lên 1, Vì vậy, trong kiểu dữ liệu **colors_t** mà chúng ta định nghĩa ở trên, **white** tương đương với **0**, **blue** tương đương với **1**, **green** tương đương với 2 và cứ tiếp tục như thế.

Nếu chúng ta chỉ định một giá trị nguyên cho một giá trị nào đó của kiểu dữ liệu liệt kê (trong ví dụ này là phần tử đầu tiên) các giá trị tiếp theo sẽ là các giá trị nguyên tiếp theo, ví dụ:

```
enum months_t {
  january=1, february, march, april, may, june, july, august, september, october, november,
```

```
december
} y2k;
```

trong trường hợp này, biến **y2k** có kiểu dữ liệu liệt kê **months_t** có thể chứa một trong 12 giá trị từ **january** đến **december** và tương đương với các giá trị nguyên từ **1** đến **12**, không phải **0** đến **11** vì chúng ta đã đặt **january** bằng **1**

Revision #1
Created 5 October 2019 16:02:36 by Laptrinh.vn
Updated 5 October 2019 16:09:37 by Laptrinh.vn