عنوان: **تغییرات اعمال شده در C++11 قسمت اول (enum)**
نویسنده: یزدان
تاریخ: ۱۳۹۲/۰۳/۰۹ ۱۴:۱۵
آدرس: www.dotnettips.info
برچسب‌ها: ++C++, Win32 Project, Win32 Console Application, C, vc, vc

نوع شمارشی enum

نوع شمارشی، یک نوع صحیح است و شامل لیستی از ثوابت می‌باشد که توسط برنامه نویس مشخص می‌گردد . انواع شمارشی برای تولید کد خودمستند به کار می‌روند یعنی کدی که به راحتی قابل درک باشد و نیاز به توضیحات اضافه نداشته باشد. زیرا به راحتی توسط نام ، نوع کاربرد و محدوده مقادیرشان قابل درک می‌باشند . مقادیر نوع شمارشی منحصربه فرد می‌باشند (unique) و شامل مقادیر تکراری نمی‌باشند در غیر این صورت کامپایلر خطای مربوطه را هشدار میدهد . نحوه تعریف نوع شمارشی :

```
enum typename{enumerator-list}
```

enum کلمه کلیدی ست ، typename نام نوع جدید است که برنامه نویس مشخص میکند و enumerator-list مجموعه مقادیری ست که این نوع جدید می‌تواند داشته باشد بعنوان مثال :

```
enum Day{SAT,SUN,MON,TUE,WED,THU,FRI}
```

اکنون Day یک نوع جدید است و متغیرهایی که از این نوع تعریف می‌شوند میتوانند یکی از مقادیر مجموعه فوق را دارا باشند .

```
Day  day1,day2;
day1 = SAT;
day2 = SUN;
```

مقادیرSAT و SUN و MON هر چند که به همین شکل بکار میروند ولی در رایانه به شکل اعداد صحیح 0 , 1 , 2 , ... ذخیره می‌شوند . به همین دلیل است که به هر یک از مقادیر SAT و SUN و ... یک شمارشگر می‌گویند . وقتی فهرست شمارشگرهای یک نوع تعریف شد به طور خودکار مقادیر 0 و 1 و ... به ترتیب به آنها اختصاص داده میشود . می‌توان مقادیر صحیح دلخواهی به شمارشگرها نسبت داد به طور مثال :

```
enum Day{SAT=1,SUN=2,MON=4,TUE=8,WED=16,THU=32,FRI=64}
```

اگر چند شمارشگر مقدار دهی شده باشند آنگاه شمارشگرهایی که مقدار دهی نشده اند ، مقادیر متوالی بعدی را خواهند گرفت .

```
enum Day{SAT=1,SUN,MON,TUE,WED,THU,FRI}
```

دستور بالا مقادیر 1 تا 7 را بترتیب به شمارشگرها اختصاص میدهد .
میتوان به شمارشگرها مقادیر یکسانی نسبت داد

```
enum Answer{NO=0,FALSE=0,YES=1,TRUE=1,OK=1}
```

ولی نمی‌توان نامهای یکسانی را در نظر گرفت ! تعریف زیر بدلیل استفاده مجدد از شمارشگر YES با خطای کامپایلر مواجه می‌شویم .

```
enum Answer{NO=0,FALSE=0,YES=1,YES=2,OK=1}
```

چند دلیل استفاده از نوع شمارشی عبارت است از :
1- enum سبب میشود که شما مقادیر مجاز و قابل انتظار را به متغیرهایتان نسبت دهید .
2- enum اجازه میدهد با استفاده از نام به مقدار دستیابی پیدا کنید پس کدهایتان خواناتر میشود .

-3 با استفاده از enum تایپ کدهایتان سریع میشود زیرا IntelliSense در مورد انتخاب گزینه مناسب شما را یاری میدهد .

چند تعریف از enum :

```
enum Color{RED,GREEN,BLUE,BLACK,ORANGE}
enum Time{SECOND,MINUTE,HOUR}
enum Date{DAY,MONTH,YEAR}
enum Language{C,DELPHI,JAVA,PERL}
enum Gender{MALE,FEMALE}
```

تا اینجا خلاصه ای از enum و مفهوم آن داشتیم

اما تغییراتی که در c++11 اعمال شده : Type-Safe Enumerations

فرض کنید دو enum تعریف کرده اید و به شکل زیر می‌باشد

```
enum Suit {Clubs, Diamonds, Hearts, Spades};
enum Jewels {Diamonds, Emeralds, Opals, Rubies, Sapphires};
```

اگر این دستورات را کامپایل کنید با خطا مواجه می‌شوید چون در هر دو enum شمارشگر Diamonds تعریف شده است . کامپایلر اجازه تعریف جدیدی از یک شمارشگر در enum دیگری نمیدهد هر چند برخی اوقات مانند مثال بالا نیازمند تعریف یک شمارشگر در چند enum بر حسب نیاز میباشیم .
برای تعریف جدیدی که در c++11 داده شده کلمه کلیدی class بعد از کلمه enum مورد استفاده قرار میگیرد . به طور مثال تعریف دو enum پیشین که با خطا مواجه میشد بصورت زیر تعریف میشود و از کامپایلر خطایی دریافت نمیکنیم .

```
enum class Suit {Clubs, Diamonds, Hearts, Spades};
enum class Jewels {Diamonds, Emeralds, Opals, Rubies, Sapphires};
```

همچنین استفاده از enum در گذشته و تبدیل آن به شکل زیر بود :

```
enum Suit {Clubs, Diamonds, Hearts, Spades};
Suit var1 = Clubs;
int var2= Clubs;
```

یک متغیر از نوع Suit بنام var1 تعریف میکنیم و شمارشگر Clubs را به آن نسبت میدهیم ، خط بعد متغیری از نوع int تعریف نمودیم و مقدار شمارشگر Clubs که 0 می‌باشد را به آن نسبت دادیم . اما اگر تعریف enum را با قوائد C++11 در نظر بگیریم این نسبت دادنها باعث خطای کامپایلر میشود و برای نسبت دادن صحیح باید به شکل زیر عمل نمود .

```
enum class Jewels {Diamonds, Emeralds, Opals, Rubies, Sapphires};
Jewels typeJewel = Jewels::Emeralds;
int suitValue = static_cast<int>(typeJewel);
```

همانطور که مشاهده میکنید ، Type-Safe یودن enum را نسبت به تعریف گذشته آن مشخص می‌باشد .
یک مثال کلی و جامع‌تر :

```
// Demonstrating type-safe and non-type-safe enumerations
#include <iostream>
using std::cout;
using std::endl;
// You can define enumerations at global scope
//enum Jewels {Diamonds, Emeralds, Rubies}; // Uncomment this for an error
enum Suit : long {Clubs, Diamonds, Hearts, Spades};
int main()
{
// Using the old enumeration type...
Suit suit = Clubs; // You can use enumerator names directly
```

```
Suit another = Suit::Diamonds; // or you can qualify them
// Automatic conversion from enumeration type to integer
cout << "suit value: " << suit << endl;
cout << "Add 10 to another: " << another + 10 << endl;
// Using type-safe enumerations...
enum class Color : char {Red, Orange, Yellow, Green, Blue, Indigo, Violet};
Color skyColor(Color::Blue); // You must qualify enumerator names
// Color grassColor(Green); // Uncomment for an error
// No auto conversion to numeric type
cout << endl
<< "Sky color value: "<< static_cast<long>(skyColor) << endl;
//cout << skyColor + 10L << endl; // Uncomment for an error
cout << "Incremented sky color: "
<< static_cast<long>(skyColor) + 10L // OK with explicit cast
<< endl;
return 0;
}
```

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۳/۱۰ ۲۳:۳

خودمونیم! بد طراحی شده. از المان یک enum می‌شده/میشه مستقیما خارج از enum بدون ارجاعی به اون استفاده کرد؟! به این می‌گن بیش از حد دست و دلبازی و منشاء سردرگمی (که در نگارش 11 به اسم type-saftey بالاخره رفع و رجوعش کردن).