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 بودن 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).

عنوان: **تغییرات اعمال شده در C++11 قسمت دوم (auto)**
نویسنده: یزدان
تاریخ: ۱۳۹۲/۰۳/۱۲ ۳:۱۰
آدرس: www.dotnettips.info
برچسب‌ها: ++C++, Win32 Project, Win32 Console Application, C, vc, vc

variable

متغیر :

برنامه هایی که نوشته می‌شوند برای پردازش داده‌ها بکار می‌روند،یعنی اطلاعاتی را از یک ورودی میگیرند و آنها را پردازش میکنند و نتایج مورد نظر را به خروجی می‌فرستند . برای پردازش ، لازم است که داده‌ها و نتایج ابتدا در حافظه اصلی ذخیره شوند،برای این کار از متغیر استفاده میکنیم .

متغیر مکانی از حافظه ست که شامل : نام ، نوع ، مقدار و آدرس می‌باشد . وقتی متغیری را تعریف میکنیم ابتدا با توجه به نوع متغیر ، آدرسی از حافظه در نظر گرفته می‌شود،سپس به آن آدرس یک نام تعلق میگیرد. نوع متغیر بیان میکند که در آن آدرس چه نوع داده ای می‌تواند ذخیره شود و چه اعمالی روی آن می‌توان انجام داد،مقدار نیز مشخص میکند که در آن محل از حافظه چه مقداری ذخیره شده است . در C++ قبل از استفاده از متغیر باید آن را اعلان نماییم . نحوه اعلان متغیر به شکل زیر می‌باشد :

```
type  name  initializer ;
```

عبارت type نوع متغیر را مشخص میکند . نوع متغیر به کامپایلر اطلاع میدهد که این متغیر چه مقادیری می‌تواند داشته باشد و چه اعمالی می‌توان روی آن انجام داد .عبارت name نام متغیر را نشان میدهد. عبارت initializer نیز برای مقداردهی اولیه استفاده می‌شود. نوع هایی که در ویژوال استادیو 2012 ساپورت می‌شوند شامل جدول زیر می‌باشند .

TYPE	SIZE IN BYTES	RANGE OF VALUES
bool	1	true or false
char	1	By default, the same as type signed char: -128 to 127. Optionally, you can make char the same range as type unsigned char.
signed char	1	-128 to 127
unsigned char	1	0 to 255
wchar_t	2	0 to 65,535
short	2	-32,768 to 32,767
unsigned short	2	0 to 65,535
int	4	-2,147,483,648 to 2,147,483,647
unsigned int	4	0 to 4,294,967,295
long	4	-2,147,483,648 to 2,147,483,647
unsigned long	4	0 to 4,294,967,295
long long	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
unsigned long long	8	0 to 18,446,744,073,709,551,615
float	4	$\pm 3.4 \times 10^{\pm 38}$ with approximately 7-digits accuracy
double	8	$\pm 1.7 \times 10^{\pm 308}$ with approximately 15-digits accuracy
long double	8	$\pm 1.7 \times 10^{\pm 308}$ with approximately 15-digits accuracy

چند تعریف از متغیر به شکل زیر :

```
int sum(0);   //  یا  int sum=0;
char ch(65);  //  ch is A
float  pi(3.14);  //  یا  float  pi = 3.14;
```

همانطور که مشهود می‌باشد طبق تعریف متغیر ، نوع و نام و مقدار اولیه (اختیاری) ، مشخص گردیده است . تا قبل از C++11 تعریف نوع متغیر الزامی بود در غیر این صورت با خطای کامپایلر مواجه می‌شدیم .

تغییرات اعمال شده در C++11 : معرفی کلمه کلیدی auto

در C++11 کلمه کلیدی auto معرفی و اضافه گردید ، با استفاده از auto ، کامپایلر این توانایی را دارد که نوع متغیر را از روی مقدار دهی اولیه آن تشخیص دهد و نیازی به مشخص نمودن نوع متغیر نداریم .

```
int x = 3;
auto y = x;
```

در تعریف فوق ابتدا نوع متغیر x را int در نظر گرفتیم و مقدار 3 را به آن نسبت دادیم . در تعریف دوم نوع متغیر را مشخص نکردیم و کامپایلر با توجه به مقدار اولیه ای که به متغیر y نسبت دادیم ، نوع آن را مشخص میکند . چون مقدار اولیه آن x می‌باشد و x از نوع int می‌باشد پس نوع متغیر y نیز از نوع int در نظر گرفته می‌شود .

دلایلی برای استفاده از auto :

Robustness : (خوشفکری) به طور فرض زمانی که مقدار برگشتی یک تابع را در یک متغیر ذخیره میکنید با تغییر نوع برگشتی تابع نیازی به تغییر کد (برای نوع متغیر ذخیره کننده مقدار برگشتی تابع) ندارید .

```
int  sample()
  {
      int  result(0);
      // To Do ...
      return  result;
  }

int main()
 {
      auto  result =  sample();
      // To Do ...
      return 0;
 }
```

و زمانی که نوع برگشتی تابع بنا به نیاز تغییر کرد

```
float  sample()
  {
      float  result(0.0);
      // To Do ...
      return  result;
  }

int main()
 {
      auto  result =  sample();
      // To Do ...
      return 0;
 }
```

همانطور که مشاهده میکنید با اینکه کد تابع و نوع برگشتی آن تغییر یافت ولی بدنه main تابع هیچ تغییری داده نشد .

Usability : (قابلیت استفاده) نیازی نیست نگران نوشتن درست و تایپ صحیح نام نوع برای متغیر باشیم

```
flot   f(0.0) ;   //  خطای نام نوع گرفته می‌شود
auto  f(0.0);   //  نیازی به وارد نمودن نوع تایپ نیستیم
```

Efficiency : برنامه نویسی ما کارآمدتر خواهد بود

مهمترین استفاده از auto سادگی آن است .

استفاده از auto بخصوص زمانی که از STL و templates استفاده میکنیم ، بسیار کارآمد می‌باشد و بسیاری از کد را کم میکند و باعث خوانایی بهتر کد می‌شود .

فرض کنید که نیاز به یک iterator جهت نمایش تمام اطلاعات کانتینری از نوع mapداریم باید از کد زیر استفاده نماییم (کانتینر را map در نظر گرفتیم)

```
map<string, string> address_book;
address_book[ "Alex" ] = "example@yahoo.com";
```

برای تعریف یک iterator به شکل زیر عمل میکنیم .

```
map<string, string>::iterator itr = address_book.begin();
```

با استفاده از auto کد فوق را میتوان به شکل زیر نوشت

```
auto itr = address_book.begin();
```

(کانتینرها :(containers) : کانتینرها اشیایی هستند که اشیا دیگر را نگهداری میکنند و دارای انواع مختلفی می‌باشند به عنوان مثال vector, map ... ,)
(تکرار کننده‌ها : (iterators): تکرار کننده‌ها اشیایی هستند که اغلب آنها اشاره گرند و با استفاده از آنها میتوان محتویات کانتینرها را همانند آرایه پیمایش کرد)