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)