عنوان: **آموزش VC++ از مقدماتی تا پیشرفته**
نویسنده: حمیدرضا
تاریخ: ۱۳۹۱/۱۲/۲۶ ۱۲:۳۰
آدرس: www.dotnettips.info
برچسب‌ها: Win32 Project, Win32 Console Application, C

بحثی که بنده قصد آموزش آن را دارم آموزش C ++ در IDE مایکروسافت visual studio می‌باشد . آموزش از پروژه‌های Win32 Console Application شروع شده و قسمت پیشرفته آموزش در پروژه‌های Win32 Project ادامه می‌یابد .

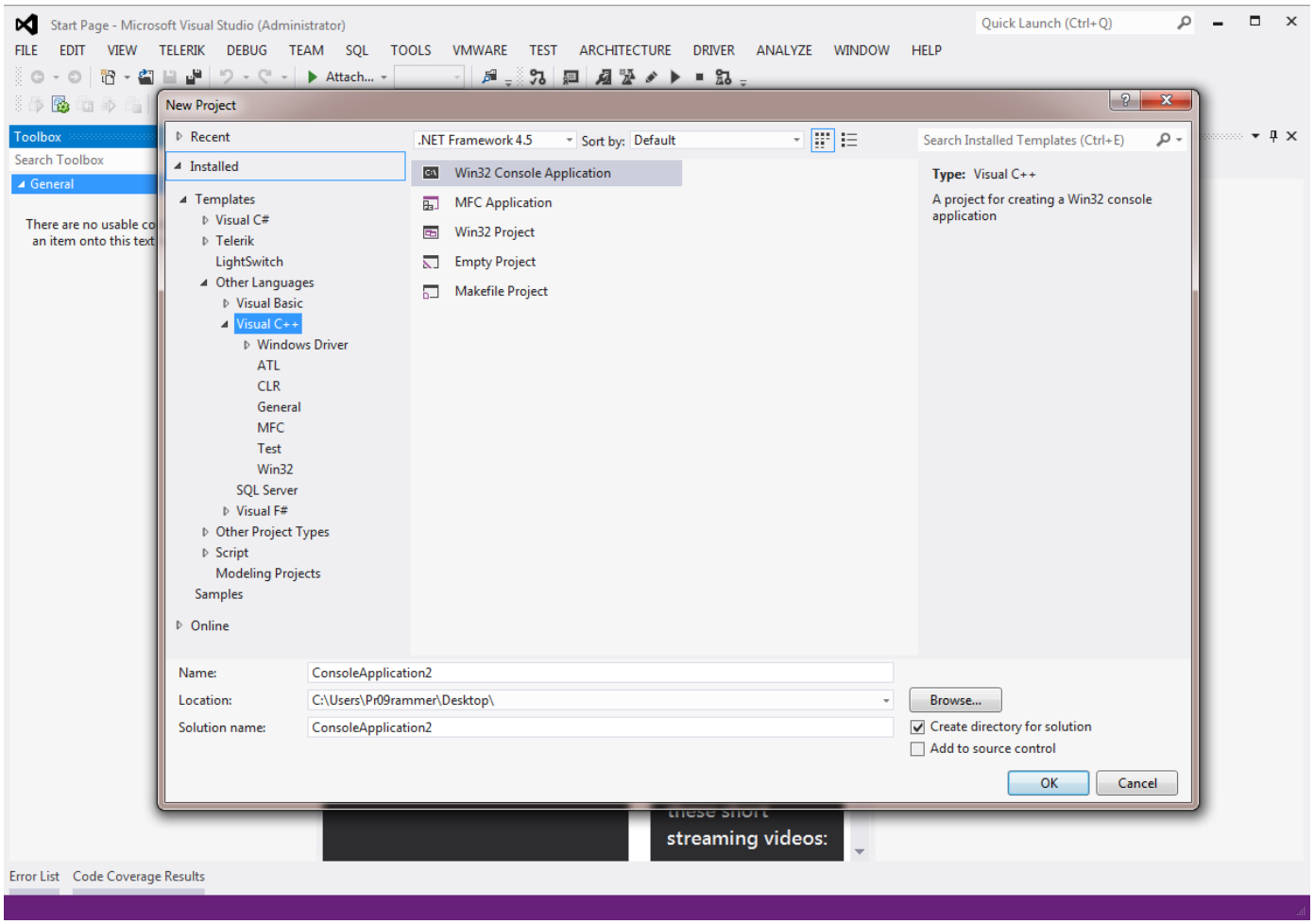
...

اولین پروژه

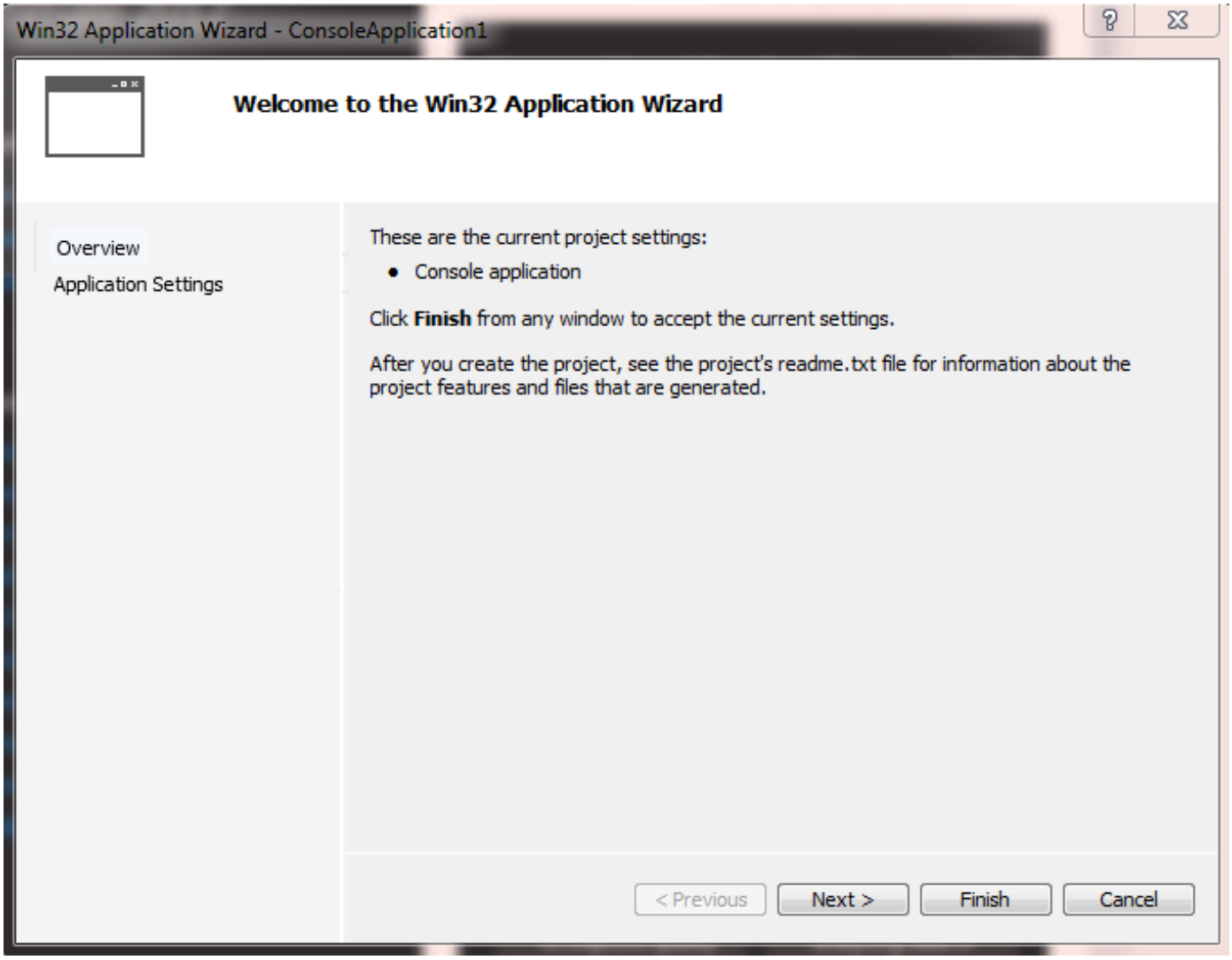
معمولا برای شروع از تاریخچه و توضیحات دیگر استفاده میکنند اما روش آموزشی که در پیش خواهیم گرفت با انجام پروژه‌های عملی بوده و هر جا که نیاز به توضیح باشد ، بیان میکنیم ...

ایجاد اولین پروژه Win32 Console Application

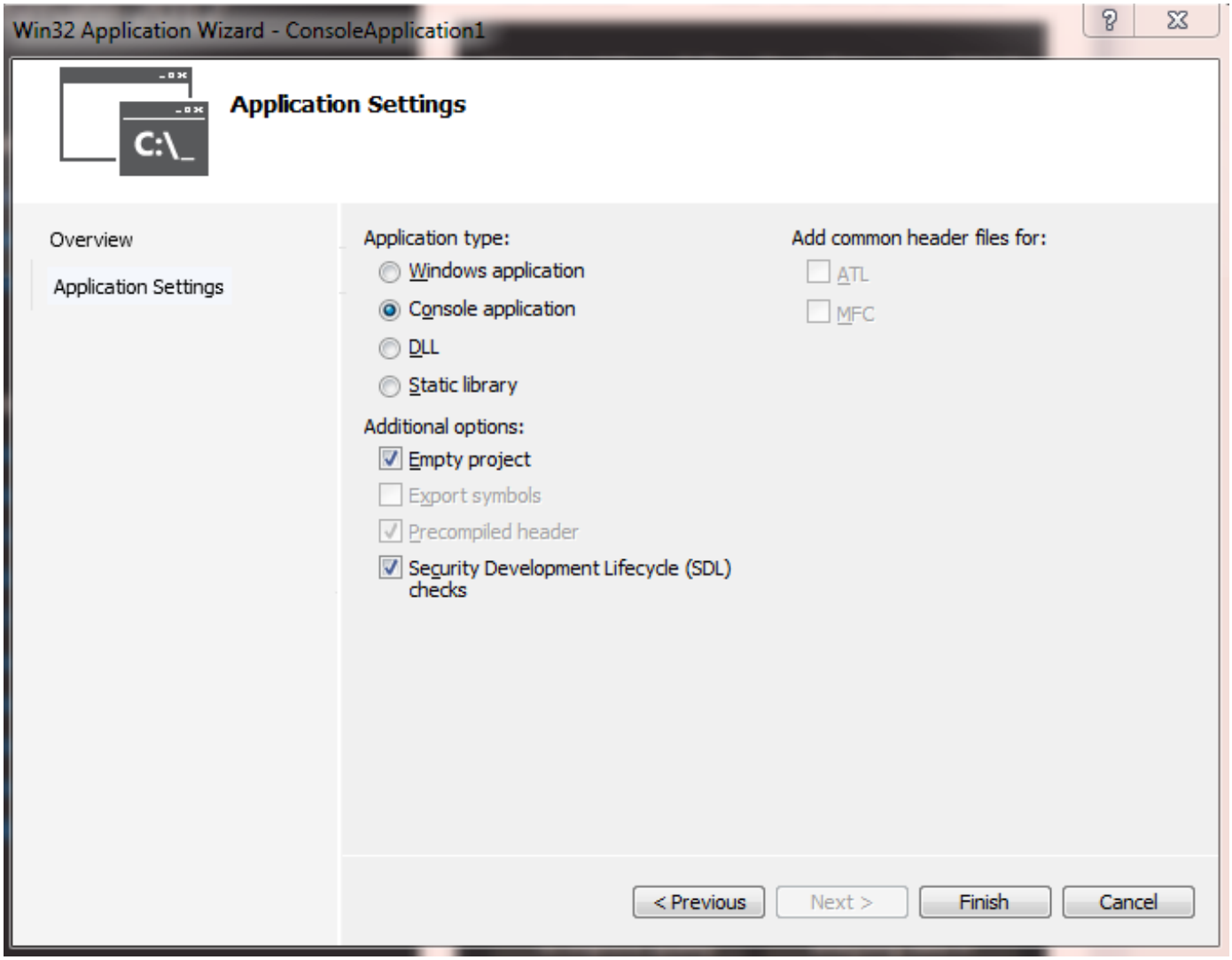
ویژوال استادیو را اجرا نمایید و از گزینه File -> New -> Project و سپس طبق عکس زیر پروژه Win32 Console Application را انتخاب نمایید ، دقت کنید که زبان انتخاب شده Visual C++ باشد.



در این مرحله میتوانید محل ذخیره شده پروژه را در قسمت Location تنظیم نمایید و از قسمت Name میتوانید نام دلخواه را وارد کنید در حالت پیش فرض اگر اولین پروژه Win32 Console در مسیر تعین شده‌ی قسمت Location باشد ، نام ConsoleApplication1 قرار گرفته است . پس از تنظیمات Ok کنید .

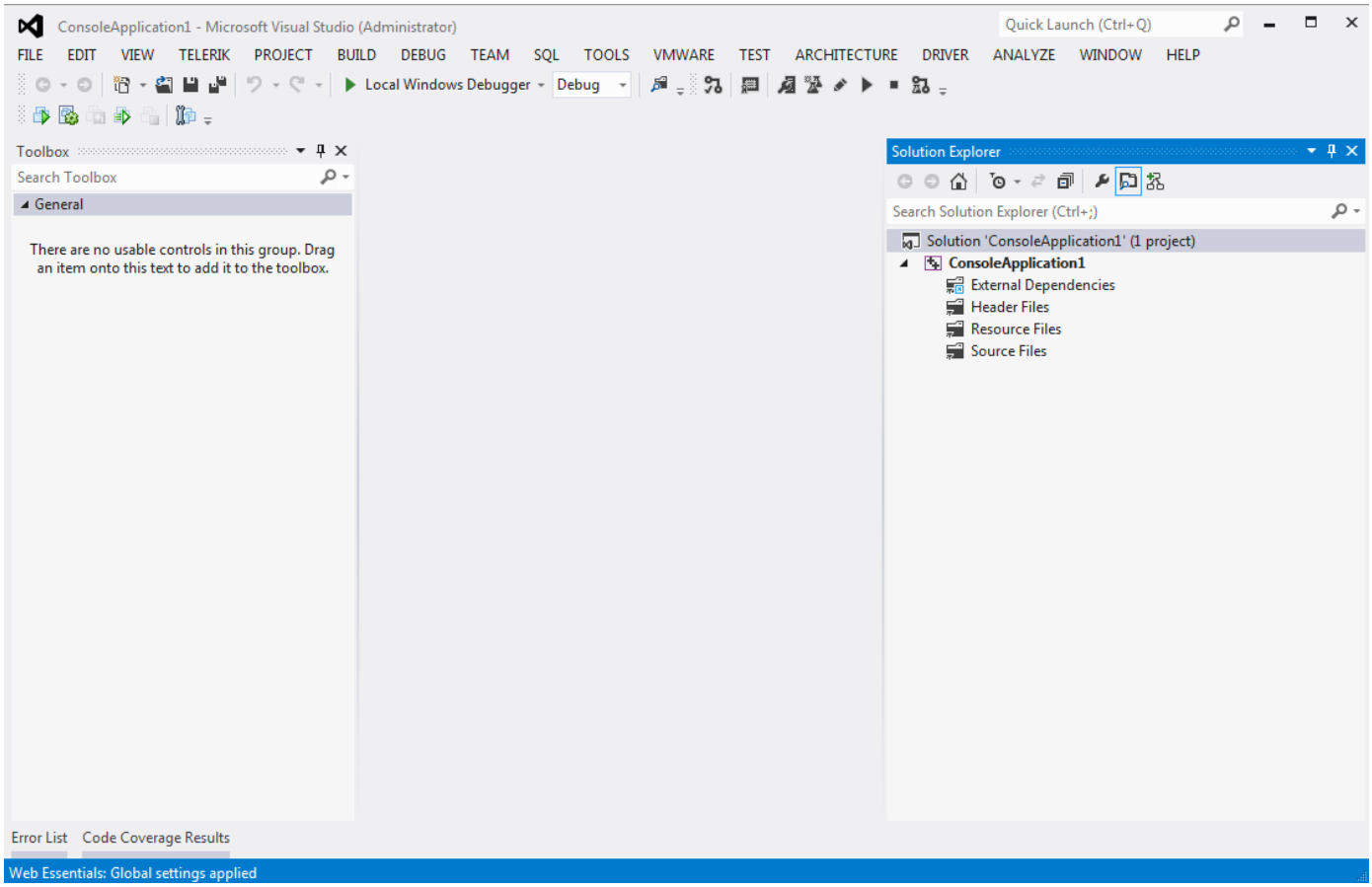


در این مرحله Next را بزنید .

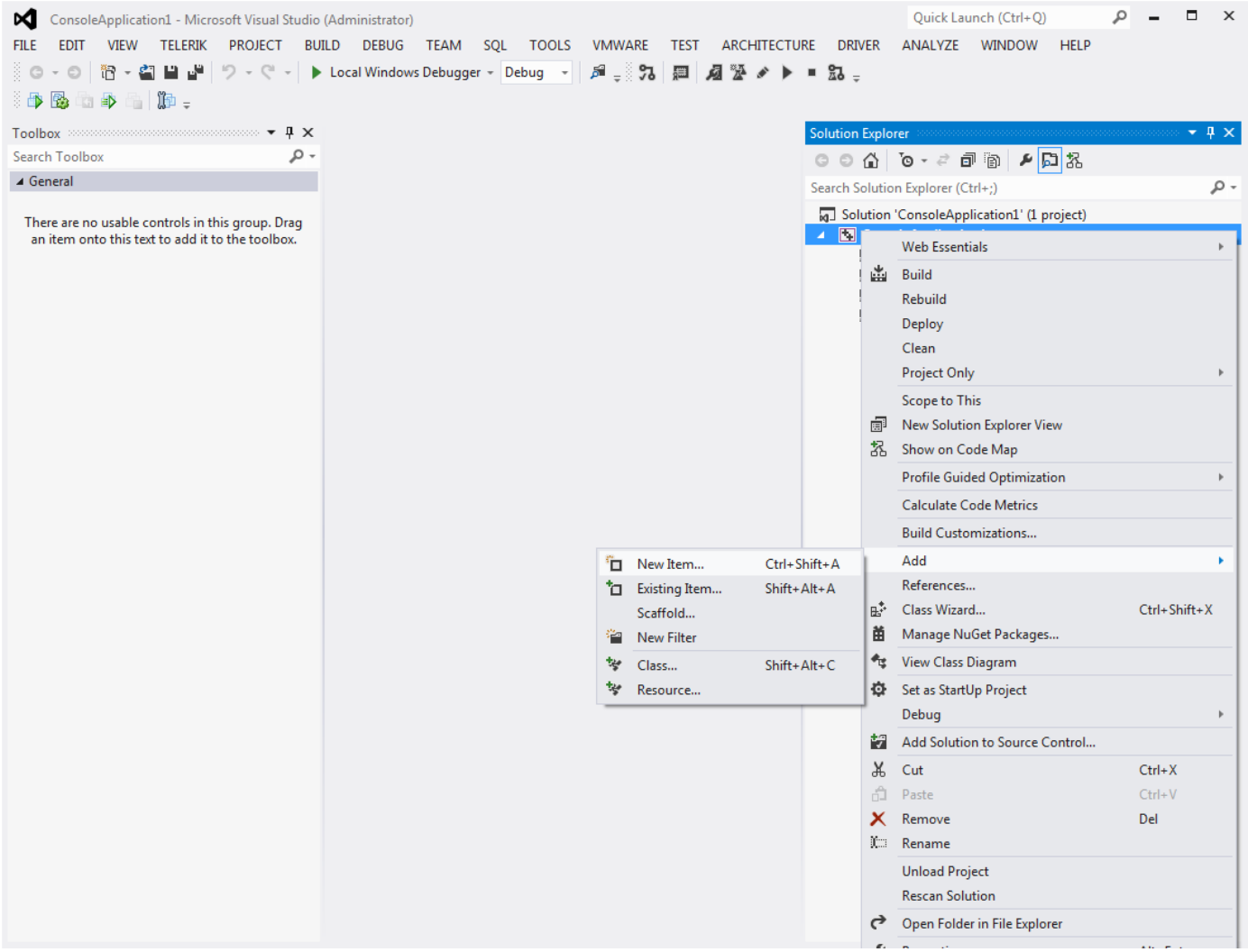


در این مرحله در قسمت Additional options تیک **Empty project** را بزنید ، همانند عکس فوق تنظیمات را انجام دهید .

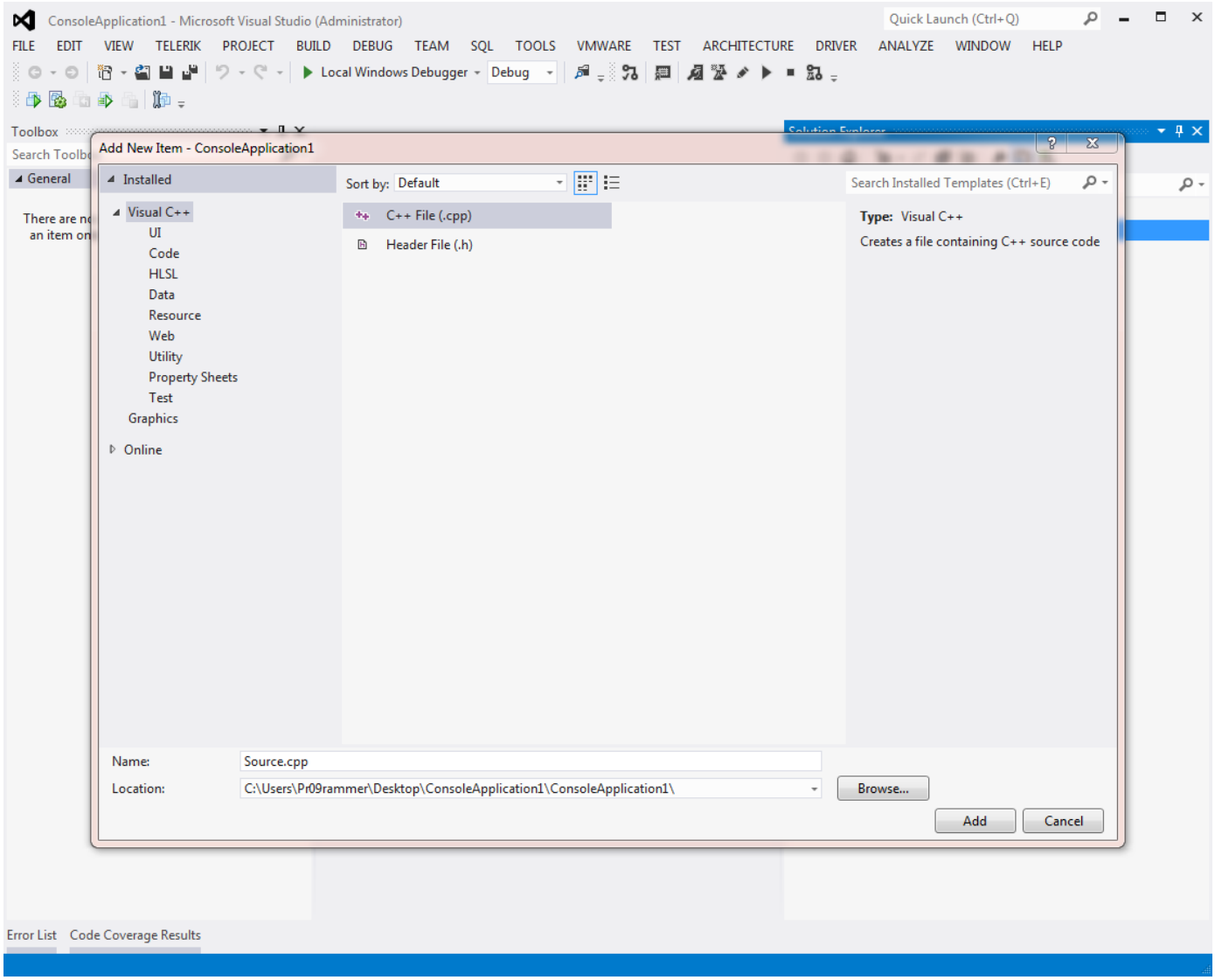
پس از انجام مراحل فوق پروژه بصورت شکل زیر ظاهر میشود .



برای کد نویسی روی نام پروژه که در اینجا ConsoleApplication1 می‌باشد ، راست کلیک میکنیم و گزینه Add و سپس New Item را انتخاب میکنیم .



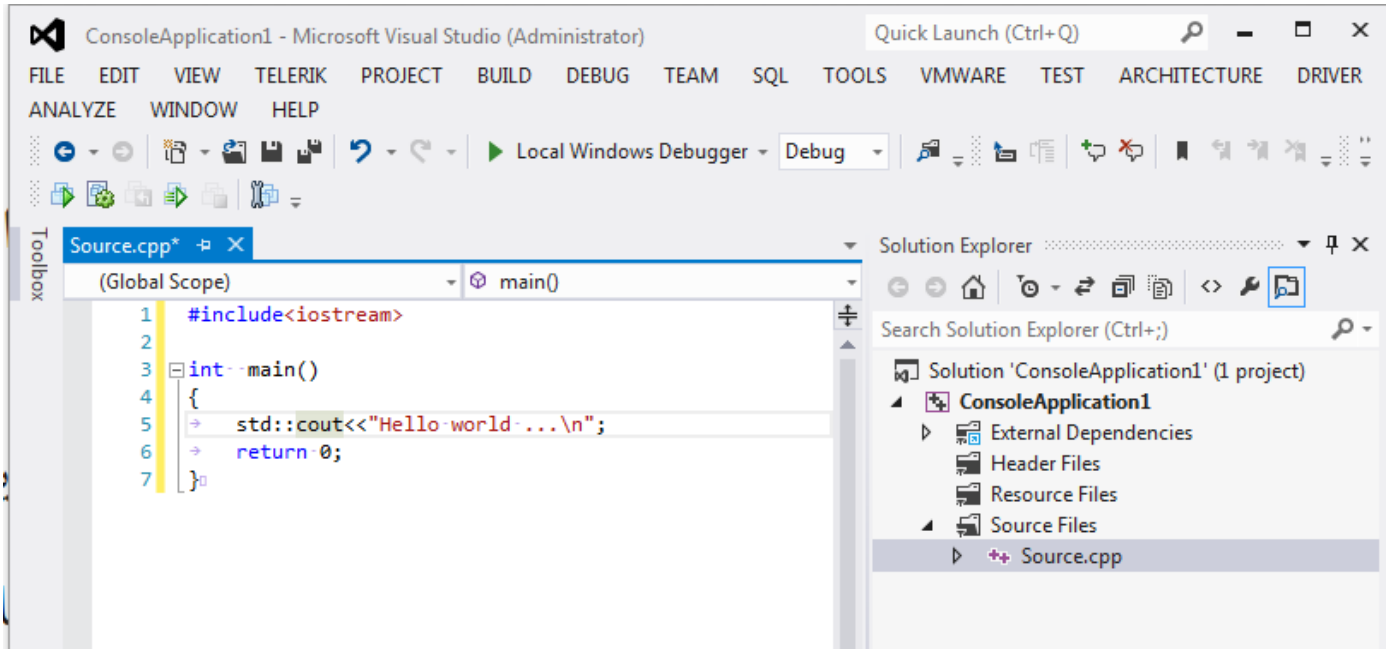
طبق عکس زیر فایل با پسوند cpp را انتخاب و Add میکنیم .



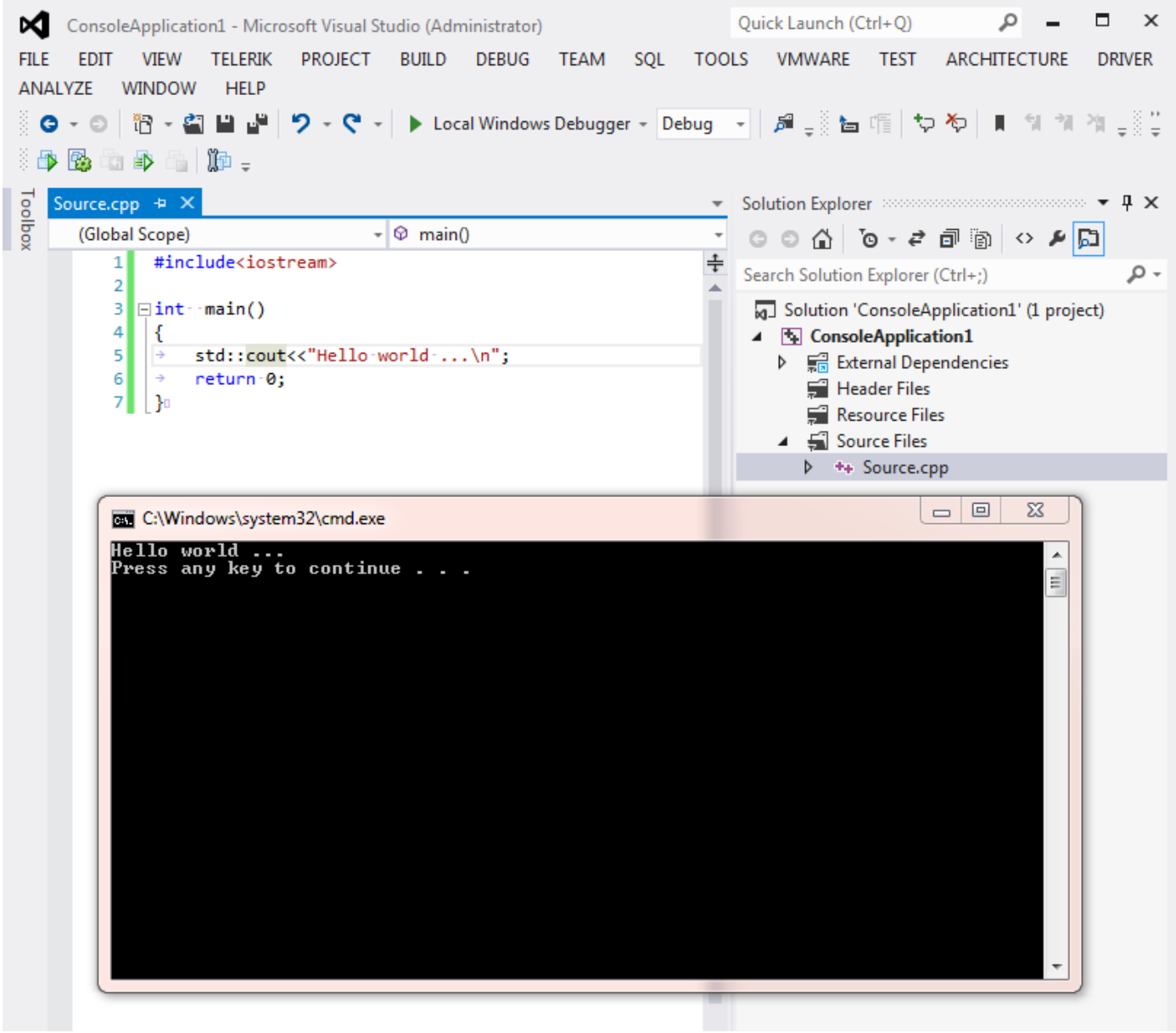
فایلی که اکنون به پروژه اضافه کردیم خالی و با نام پیش فرض Source.cpp می‌باشد ، دستورات زیر را در آن تایپ کنید .حال پروژه به شکل زیر خواهد بود .

```
#include<iostream>

int  main()
{
   std::cout<<"Hello world ...\n";
   return 0;
}
```



برای اجرای پروژه کلید F5 را فشار دهید و اگر میخواهید نتیجه کار را مشاهده کنید کلید Ctrl + F5 را امتحان کنید .



شما اولین پروژه VC++ را اجرا نمودید (آفرین) .

اما توضیحات :

خط اول برنامه یک راهنمای پیش پردازنده است ، کاراکتر # که نشان میدهد این خط یک راهنمای پیش پردازنده است و بعد عبارت include و نام یک فایل کتابخانه ای که بین علامت <> قرار داده شده ، فایل سرآیند استفاده شده در اینجا iostream میباشد . (به فایل‌های کتابخانه ای ، فایلهای سرآیند (Header Files) نیز گفته میشود.) راهنمای پیش پردازنده خطی است که به کامپایلر اطلاع میدهد در برنامه موجودیتی است که تعریف آن را در فایل سرآیند مذکور جستجو کند . در این برنامه از std::cout استفاده شده ، که کامپایلر در مورد آن چیزی نمیداند لذا به فایل iostream مراجعه نموده ، تعریف آن را می‌یابد و آن را اجرا میکند .

خط 3 :

بخشی از هر برنامه تابع می‌باشد . پرانتزهای واقع پس از آن **main** نشان می‌دهند که **main** یک بلوک برنامه بنام *تابع* است.
برنامه‌ها می‌توانند حاوی یک یا چندین تابع باشند، اما main تابع اصلی برنامه است که وجود آن الزامی میباشد . کلمه کلیدی **int** که

در سمت چپ **main** قرار گرفته، بر این نکته دلالت دارد که **main** یک مقدار صحیح برمی‌گرداند.

خط 5 :

با استفاده از این دستور رشته ای را به خروجی استاندارد که معمولا صفحه نمایش باشد ارسال میکنیم .

خط 6 :

که return 0; میباشد مقدار برگشتی تابع را مشخص میکند در حقیقت این خط که مقدار 0 را برمیگرادند نشان دهنده اتمام موفقیت آمیز برنامه می‌باشد .

به مرور زمان نسبت به موارد بالا بیشتر و مفصل صحبت خواهیم نمود .

نظرات خوانندگان

نویسنده: علي
تاریخ: ۱۳۹۱/۱۲/۲۶ ۱۶:۱۸

دوست عزیز، بهتر نیست مباحث جدید c++0x رو مطرح کنید؟

نویسنده: حمیدرضا
تاریخ: ۱۳۹۱/۱۲/۲۶ ۱۷:۳۹

انشاا... قسمت باشه حتما صحبت میکنیم ، ولی اگر به تیتر موضوع دقت کنید هدف آموزش از مقدماتی تا پیشرفته است که اگر خدا یاری کند ادامه میدم و در خدمت اساتید و دوستان خواهم بود .
در مورد مبحثی که شما فرمودین در اینجا کامل توضیح داده شده است .
مباحث جدید حتما جزئی از آموزش خواهند بود .

نویسنده: CSHARPDOOST
تاریخ: ۱۳۹۲/۰۱/۱۷ ۱۵:۱۴

ممنون . خوبه بی زحمت ادامه بدید.

عنوان: **قسمت دوم -- نگاهی دقیق تر به اولین پروژه VC++ (درک مفهوم فایلهای سرآیند و فضای نام ، ویژگیهای زبان C++ و برخی قوانین برنامه نویسی در C++)**
نویسنده: حمیدرضا
تاریخ: ۱۳۹۱/۱۲/۲۷ ۳:۰
آدرس: www.dotnettips.info
برچسب‌ها: Win32 Project, Win32 Console Application, C

در این قسمت نگاهی دقیق‌تر به فایلهای سرآیند ، فضای نام ، ویژگیهای زبان C++ و برخی قوانین برنامه نویسی C++ خواهیم داشت و همچنین در مورد اولین پروژه توضیحات جامع‌تری ارائه میکنیم .

یک برنامه مجموعه ای از دستورات است که توسط کامپیوتر اجرا میگردد ، برنامه نویسان برای نوشتن این دستورات از زبانهای برنامه نویسی استفاده میکنند ، برخی از این زبانها مسقیما قابل فهم توسط کامپیوتر بوده و برخی نیاز به ترجمه دارند . زبانهای برنامه نویسی را میتوان به سه دسته تقسیم نمود :

1 - زبانهای ماشین
2 - زبانهای اسمبلی
3 - زبانهای سطح بالا

زبانهای ماشین :
زبانی که مستقیما و بدون نیاز به ترجمه قابل فهم توسط کامپیوتر می‌باشد . هر پردازنده یا processor زبان خاص خود را دارد !... در نتیجه تنوع زبان ماشین بستگی به انواع پردازنده‌های موجود دارد و اگر دو کامپیوتر دارای پردازنده‌های یکسان نباشتد ، زبان ماشین آنها با یکدیگر متفاوت و غیر قابل فهم برای دیگری می‌باشد . زبان ماشین وابسته به ماشین یا Machine independent می‌باشد . تمامی دستورات در این زبان توالی از 0 و 1 می‌باشند . برنامه‌های اولیه را با این زبان می‌نوشتند در نتیجه نوشتن برنامه سخت و احتمال خطا داشتن در آن زیاد بود . ار آنجا که نوشتن برنامه به این زبان سخت و فهم برنامه‌های نوشته شده به آن دشوار بود ، برنامه نویسان به فکر استفاده از حروف بجای دستورات زبان ماشین افتادند (پیدایش زبان اسمبلی)

زبانهای اسمبلی :
به زبانی که دستورات زبان ماشین را با نمادهای حرفی بیان میکند، زبان اسمبلی (Assembley Language) می‌گویند . چون این زبان مستقیما قابل فهم برای کامپیوتر نیست باید قبل از اجرا آن را به زبان ماشین ترجمه کرد ، به این مترجم اسمبلر گفته میشود . برنامه‌های نوشته شده به این زبان قابل فهم برای برنامه نویس بود اما از آنجا که به ازای هر دستور زبان ماشین یک دستور زبان اسمبلی داشتیم از حجم برنامه‌ها کاسته نشد ! .. بعلاوه چون زبان اسمبلی همانند زبان ماشین از دستورات پایه ای و سطح پایین استفاده میکرد نوشتن برنامه با این زبان هم سخت و مشکل بود . لذا اهل خرد به فکر ابداع نسلی از زبانهای بهتر بودند (پیدایش زبانهای سطح بالا)

زبانهای سطح بالا :
زبانهای سطح بالا قابل فهم بودند و این امکان را داشتند تا چند دستور زبان ماشین یا اسمبلی را بتوان در قالب یک دستور نوشت **(منظور توابع کتابخانه ای در C/C++)** . پس هم فهم ، هم نوشتن برنامه در این زبانها راحت و هم تعداد خطوط کد کمتر شد . این زبانها به زبانهای برنامه نویسی سطح بالا یا High-Level Programming Language معروفند . البته برنامه نوشته شده در این زبان نیز برای کامپیوتر قابل فهم نبوده و باید به زبان ماشین ترجمه شوند ، این وظیفه بر عهده کامپایلر می‌باشد . اولین زبانهای برنامه نویسی سطح بالا مانند PASCAL ، COBOL ، FORTRAN و **C** می‌باشند . زبان برنامه نویسی **C++** تکامل یافته زبان **C** میباشد .
هر یک از زبانهای برنامه نویسی سطح بالا یک روش برنامه نویسی را پشتیبانی میکند به طور مثال زبان C و PASCAL از روشهای برنامه نویسی ساخت یافته ای و پیمانه ای و زبانهای مانند C++ و JAVA از روش برنامه سازی شی گرا یا Object Oriented Programming یا به اختصار (OOP) استفاده میکنند . زبان **C++** چون زبان C را بطور کامل در بر دارد پس از هر سه روش برنامه نویسی ساخت یافته و پیمانه ای و شی گرا استفاده میکند .

تا اینجا با تاریخچه ای از زبانها و مراحل تکامل آنها آشنا شدیم . حال **ویژگیها و دلایل استفاده از زبان C++** را مرور میکنیم :

زبان C در سال 1972 توسط دنیس ریچی طراحی شد . زبان C تکامل یافته زبان BCPL است که طراح آن مارتین ریچاردز می‌باشد ، زبان BCPL نیز از زبان B مشتق شده است که طراح آن کن تامسون بود . (خداوند روح دنیس ریچی را همچون هوگو چاوز با مسیح

بازگرداند ! ...) .

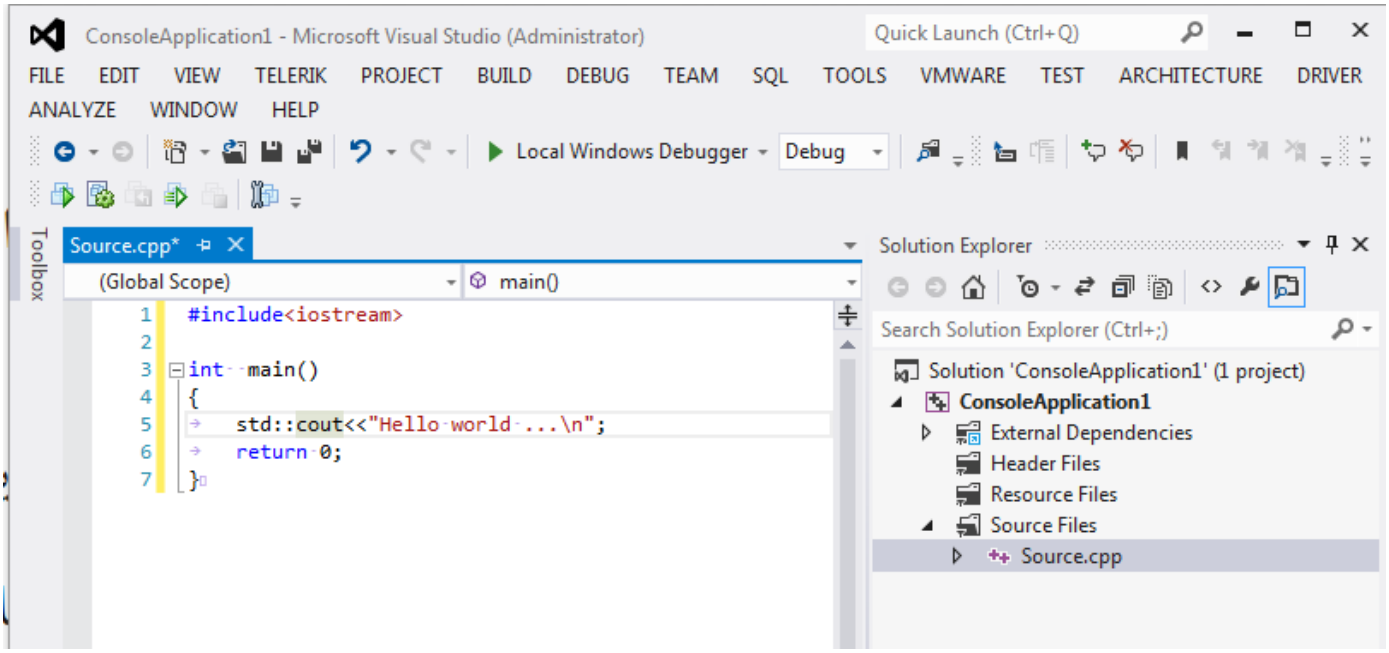
از این زبان برای نوشتن برنامه‌های سیستمی ، همچون سیستم عامل ، کامپایلر ، مفسر ، ویرایشگر ، برنامه‌های مدیریت بانک اطلاعاتی ، اسمبلر استفاده میکنند .

زبان C برای اجرای بسیاری از دستوراتش از توابع کتابخانه ای استفاده میکند و بیشتر خصوصیات وابسته به سخت افزار را به این توابع واگذار میکند لذا نرم افزار تولید شده با این زبان به سخت افزار خاصی بستگی ندارد و با اندکی تغییرات می‌توانیم نرم افزار مورد نظر را روی ماشینهای متفاوت اجرا کنیم ، در نتیجه برنامه نوشته شده با C قابلیت انتقال (Portability) دارند . بعلاوه کاربر میتواند توابع کتابخانه ای خاص خودش را بنویسد و از آنها در برنامه هایش استفاده کند .

برنامه‌های مقصدی که توسط کامپیلرهای C ساخته میشود بسیار فشرده و کم حجم‌تر از برنامه‌های مشابه در سایر زبانهاست ، این امر باعث افزایش سرعت اجرای آنها میشود .

++C که از نسل C است تمام ویژگیهای ذکر شده بالا را دارد ، علاوه بر آن شی گرا نیز می‌باشد . برنامه‌های شی گرا منظم و ساخت یافته اند و قابل آپدیت هستند و به سهولت تغییر و بهبود می‌یابند و قابلیت اطمینان و پایداری بیشتری دارند .

تحلیل اولین پروژه :



در اولین پروژه کد فوق را بکار بردیم ، حال به شرح دستورات آن می‌پردازیم .

```
#include <iostream>
```

دستوراتی که علامت # پیش از آنها قرار میگیرد ، دستورات راهنمای پیش پردازنده هستند . این خط یک دستور پیش پردازنده است که توسط پیش پردازنده و قبل از شروع کامپایل ، پردازش میشود . این کد فایل iostream را به برنامه اضافه میکند . کتابخانه استاندارد C++ به چندین بخش تقسیم شده است و هر بخش فایل سرآیند خود را دارد . دلیل قرار گرفتن این دستور در ابتدای برنامه این است که ، پیش از استفاده از هر تابع و فراخوانی کردن آن در برنامه ، کامپایلر لازم است اطلاعاتی در مورد آن تابع داشته باشد . در خط کد بالا فایل سرآیند iostream استفاده نمودیم زیرا شامل توابع مربوط به I/O (ورودی / خروجی) می‌باشد .

```
int  main()
{
```

```
   return 0;
}
```

دستور فوق بخشی از هر برنامه C++ است ، main تابع اصلی هر برنامه C++ است که شروع برنامه از آنجا آغاز می‌شود . کلمه int در ابتدای این خط ، مشخص میکند که تابع main پس از اجرا و به عنوان مقدار برگشتی (return 0;) یک عدد صحیح باز می‌گرداند .

```
std::cout<<"Hello world ...\n";
```

دستور فوق یک رشته را در خروجی استاندارد که معمولا صفحه نمایش می‌باشد ارسال میکند . std یک فضای نام است . فضای نام محدوده ای است که چند موجودیت در آن تعریف شده است . مثلا موجودیت cout در فضای نام std در فایل سرآیند iostream تعریف شده است .

در زبان C++ هر دستور به ; (سیموکالن) ختم میشود .

نظرات خوانندگان

نویسنده: علیرضا صالحی
تاریخ: ۱۳۹۱/۱۲/۲۷ ۱۳:۱۳

عنوان مطلب صحیح نیست لطفا تغییر دهید،
در فضای برنامه نویسی مراد از VC.NET زبان برنامه نویسی CPP تحت CLI است یعنی با سینتکس CPP و کامپایلر .NET یعنی همان کامپایلری که C#.NET و VB.NET استفاده می‌کنند. یا به عبارتی Managed CPP. در منوی New Project در Visual Studio گزینه CLR باید انتخاب شود.

آموزش هایی که شما ارائه کرده اید مربوط به Native CPP است. برنامه نویسی MFC یا VCL یا Win32 یا.. متفاوت با VC++.NET است.در منوی New Project در Visual Studio گزینه‌های غیر از CLR همگی native هستند.

بهتر است مشابه msdn از واژه Visual C++ استفاده کنید:
Visual C++
.NET Programming in Visual C++

با تشکر

نویسنده: حمیدرضا
تاریخ: ۱۳۹۱/۱۲/۲۷ ۱۳:۴۲

دوست عزیز بنده هم VC++ نوشتم ولی نمیدونم چرا ++ نمایش داده نمیشه .

انشاا... در هر دو مورد کد مدیریت شده و native کد صحبت خواهیم نمود . کد مدیریت شده همانطور که شما فرمودین تحت common language runtime اجرا میشود و برای اجرای برنامه نیاز به نصب دات نت فریمورک روی ماشین مقصد هست ، ولی native کد فقط از توابع کتابخانه ای استفاده میکند و نیازی به نصب .net framework جهت اجرای برنامه بر روی ماشین مقصد ندارد . آموزشهایی که تا کنون داده ام (2 مورد) با توجه به گفته شما مربوط به قسمت native آن می‌باشد .
در ادامه حتما در مورد تفاوت‌های کد مدیریت شده و کد محلی صحبت خواهیم نمود .
تغییر اعمال شد .

نویسنده: علیرضا.م
تاریخ: ۱۳۹۲/۰۱/۰۴ ۲۲:۵۹

سلام
آیا کد native ویژوال سی پلاس پلاس در سایر سیستم عامل‌ها از جمله لینوکس (ابونتو) ران می‌شوند.
اگر نه که به نظرم جاوا خیلی بهتر از سی پلاس پلاس باشد، چون نسخه سازمانی اش توانایی تولید نرم افزارهای native قابل اجرا در تمامی سیستم عامل‌ها رو دارد.

نویسنده: علي
تاریخ: ۱۳۹۲/۰۱/۰۴ ۲۳:۲۹

اگه نه که mono هست و با اون میشه کدهای دات نت رو روی لینوکس هم اجرا کرد.

نویسنده: حمیدرضا
تاریخ: ۱۳۹۲/۰۱/۰۵ ۲:۱۶

با سلام

در جواب : علیرضا.م

در مورد سوال اول برنامه‌های نوشته شده در VC++ ، اینجا رو ببینید .
میتونید از IDEهای دیگه مختص به لینوکس استفاده کنید اینجا رو ببینید .

در مورد سوال دوم باید بگم خود جاوا رو با C نوشتن و برای این منظور که شما فرمودین یا به اصطلاح Cross Platform بودن ، میتونید اینجا و اینجا رو ببینید .

موفق باشید

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)