کارگاه

مبانی کامپیوتر و برنامه نویسی

به زبان C++

بسم الله ارحمن الرحیم

پیشگفتار

از آنجایی که منبع مشخص و منسجمی جهت تدریس کد درس ۱۳۲۲۰۸۸ با عنوان "کارگاه مبانی کامپیوتر و برنامه نویسی" برای دانشجویان رشته مهندسی کامپیوتر در دسترس نیست، بر آن شدیم، چالش‌هایی که دانشجویان عزیز با آن مواجه هستند و همچنین اشکالاتی که ممکن است یک برنامه نویس، در ابتدای راه با آن مواجه شود، را با استفاده از تجربه حاصل از کلاس درس و کارگاه برنامه‌نویسی، در قالب این مجموعه-ی گردآوری شده، رفع نماییم.

در این مجموعه سعی بر آن بوده که از مفاهیم اولیه آغاز و به تدریج مطالب پیچیده‌تر مطرح شود.امتیاز بارز این مجموعه، کاربردی و عملیاتی بودن مطالب آن می‌باشد.

در نهایت، از همه شما اساتید گرامی، که وقت شریف خود را برای مطالعه این اثر می‌گذارید، تقاضا داریم ما را از نظرات سازنده ، پیشنهادات و انتقادات خود، بهره‌مند سازید.

با تشکر

فهرست مطالب:

۱.مقدمه ای راجع به کامپایلر و IDE

کامپایلر چیست ؟

کامپایلر، برنامه یا مجموعه‌ای از برنامه‌های کامپیوتری است که متنی از زبان برنامه نویسی سطح بالا (زبان مبدا) را به زبانی سطح پایین (زبان مقصد)، مثل اسمبلی یا زبان ماشین، تبدیل می‌کند. خروجی این برنامه ممکن است برای پردازش شدن توسط برنامه دیگری مثل پیوند دهنده مناسب باشد یا فایل متنی باشد که انسان نیز بتواند آن را بخواند.

IDE چیست ؟

یک IDE یا به طور کامل محیط توسعه یکپارچه که مخففی از Integrated Development Environment میباشد. برنامه نرم‌افزاری است که برای کمک به برنامه نویسان و توسعه دهندگان جهت ساخت نرم افزار طراحی شده است. اکثر IDEها شامل یک ویرایشگر[۱] کد منبع , یک یا چند کامپایلر[۲] و یک اصلاح کننده خطا[۳] میباشند . البته لازم به ذکر است که IDE هایی نیز وجود دارند که فقط یک محیط ادیتوری یا ویرایشگری برای کدها دارند و بسیاری از آنها نیز هم یک محیط ویرایشی برای کدها دارند و هم قابلیت اجرا گرفتن و خطایابی از کدها را دارند.

تفاوت بین کامپایلر و IDE چیست ؟

کامپایلر وظیفه ی تبدیل کد های برنامه نویسی به زبان قابل فهم ماشین را برعهده دارد اما IDEیک نرم افزار کمکی برای راحتتر شدن برنامه نویسی است . بی شک زبان Cو C++جزء قدرتمندترین و مشهورترین زبان های برنامه نویسی جهان هستند و کامپایلر ها و IDEهای بسیاری برای آن ها عرضه شده است . که تعداد محدودی از آن ها دارای محبوبیت و قدرت کافی هستند . در زیر بهترین کامپایلر ها و IDEهای جهان به اختصار توضیح داده شده اند .

---

۱. Editor
۲. Compiler
۳. Debugger

کامپایلر ها

MinGw: نرم افزار MinGwکامپایلر مخصوص مایکروسافت میباشد که فقط از ویندوز پشتیبانی میکند و برای C RunTimeو برخی دیگر از زبان های RunTimeمیباشد .

GCC: نرم افزار GCCیک کامپایلر رایگان زیر نظر GNUمیباشد که نه تنها کد های C – C++را کامپایل میکند بلکه از زبان های Java , Objective_C , Adaو ..... نیز پشتیبانی میکند .

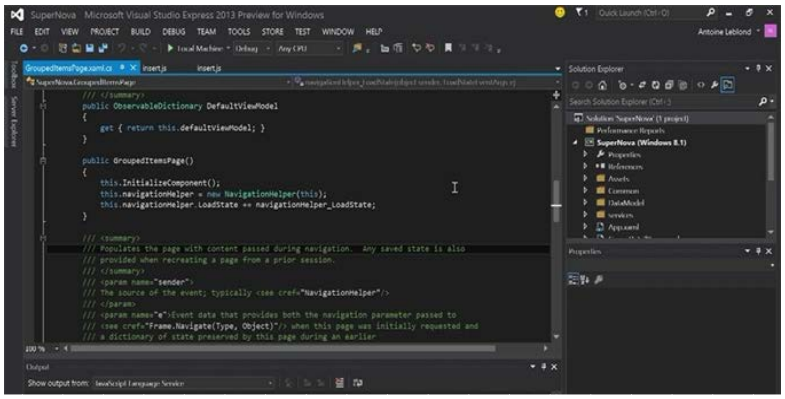
Tiny C Compiler: نرم افزار TCCیکی از بهترین کامپایلر های Cمیباشد که از تمامی پیش پردازنده ها پشتیبانی میکند و در آن از اسمبلر GNUاستفاده شده است . لازم به ذکر است که اسمبلر GNUیکی از بهترین اسمبلر های جهان است .

Ideone: نرم افزارIdeone یک IDEو کامپایلر آنلاین میباشد که از C – C++و ۶۰ زبان دیگر پشتیبانی میکند .

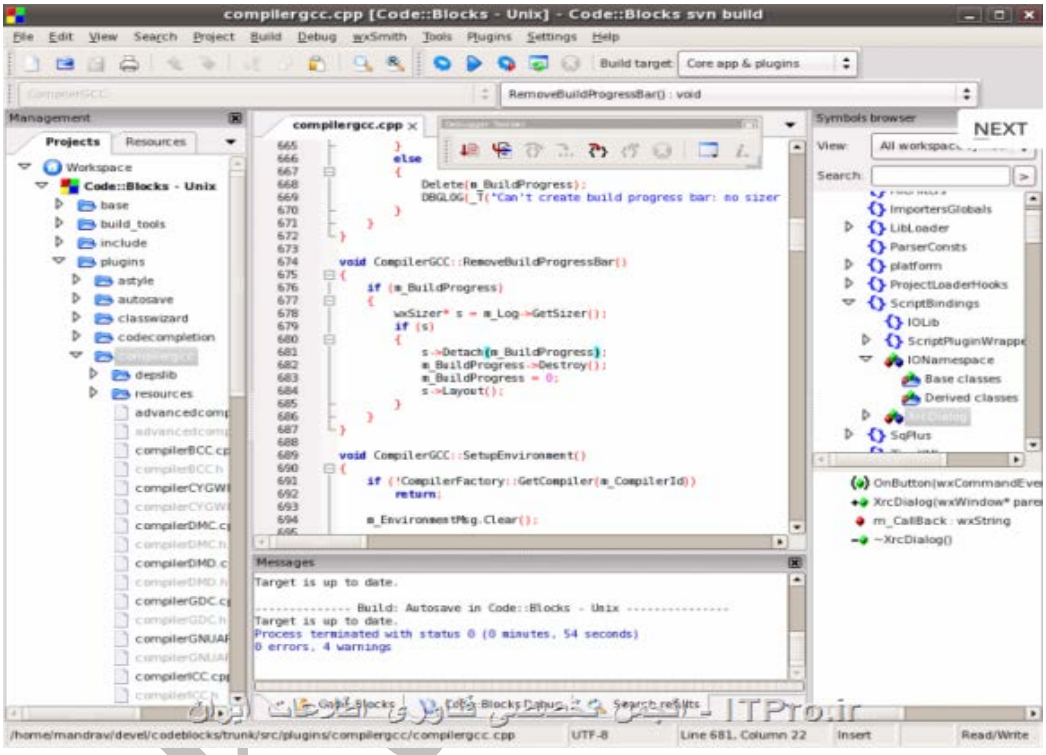
نکته : چهار کامپایلر نام برده شده در بالا از بهترین کامپایلر ها ی جهان هستند که کمترین اشکالات را درمیان دیگر کامپایلر ها داشته اند .

معرفی انواع IDE

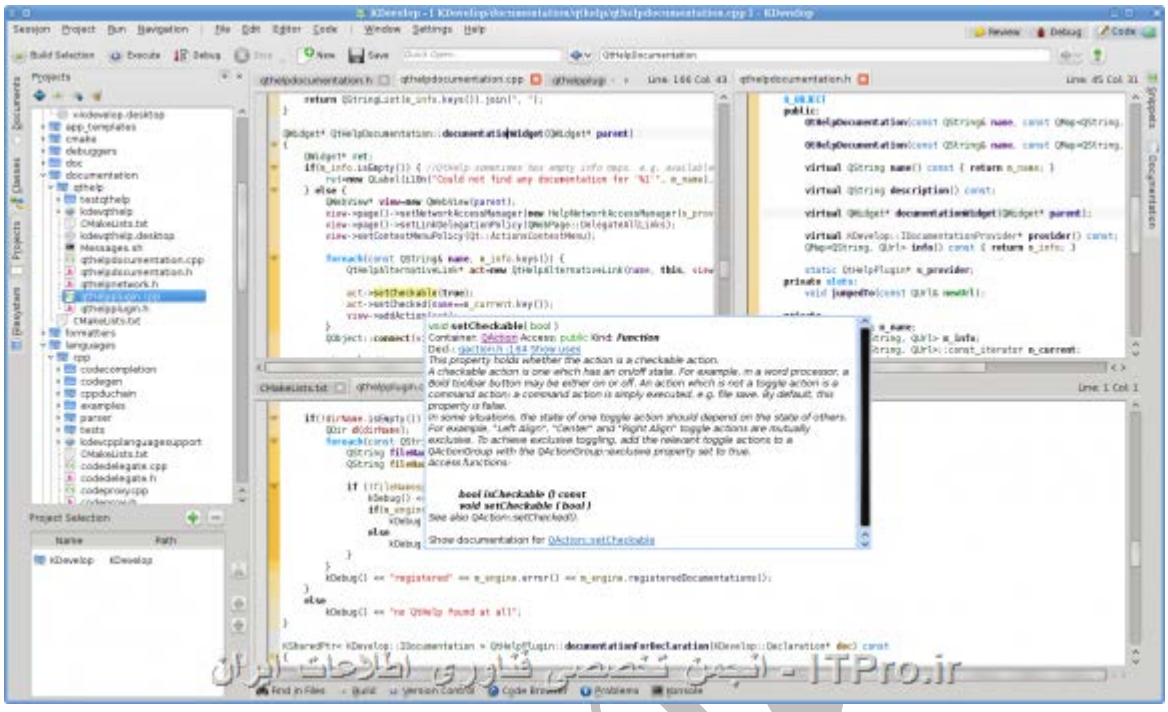
Visual studio: از قابلیت های VSمیتوان به برنامه نویسی برای موبایل , وب و دسکتاپ اشاره کرد و پشتیبانی از زبان های بسیاری هم چون Css , C , C++ , C# , Basic , Asp.net , Python , JavaScript , Ruby , Xmlو ..... و هم چنین قابلیت های بیشمار دیگر اما از بدی های آن میتوان پشتیبانی نکردن از دیگرسیستم عامل ها و کامپایلر ها , حجم بسیار زیاد و قیمت سرسام آور آن اشاره کرد .



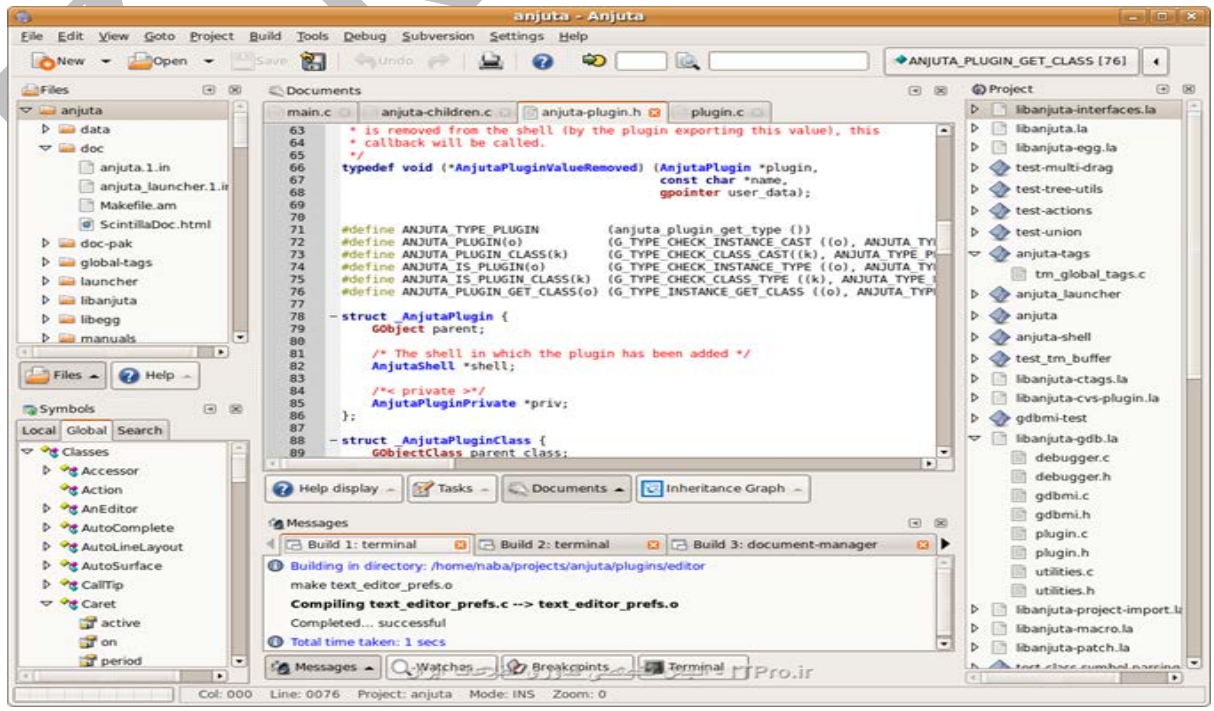
Code blocks: ادیتور C::Bیک ادیتور مخصوص C-C++است که البته در نگارش جدید آن Fortranنیز اضافه شده است سرعت بالا پشتیبانی از تمام سیستم عامل ها , کامپایلرها , حجم بسیار کم و همچنین رایگان و متن باز Open Source بودن آن , آن را در بین برنامه نویسان بسیار محبوب کرده است .



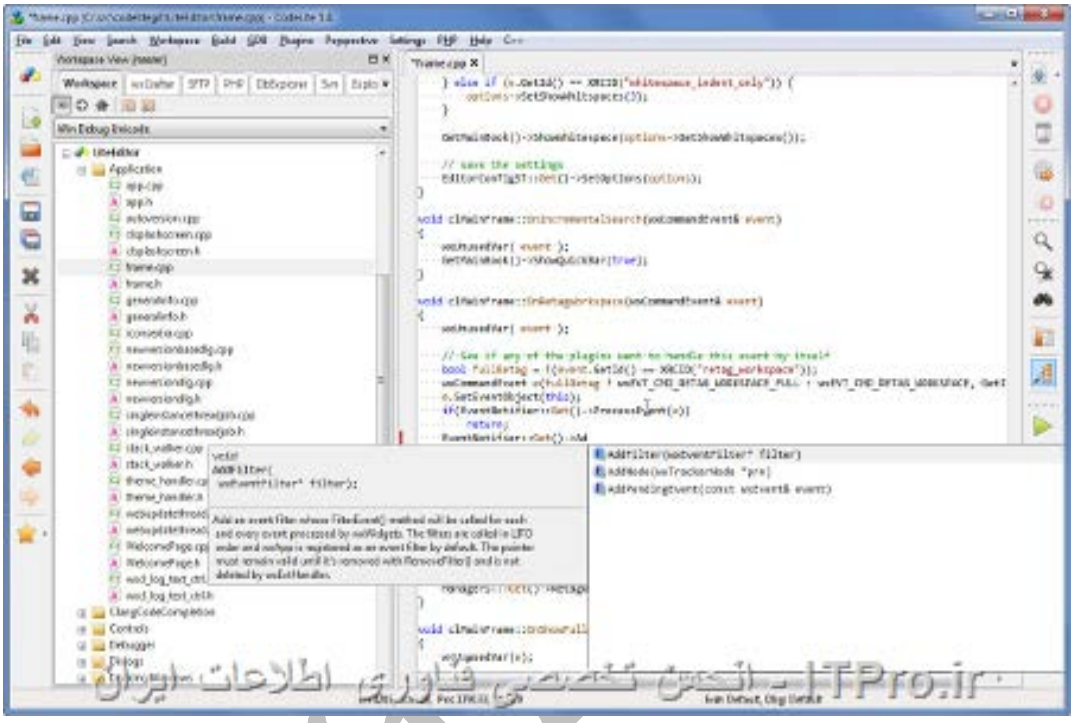
Kdevelop: ادیتور Kdیک ادیتور C - C++رایگان متن باز و کم حجم برای سیستم عامل های خانواده ی لینوکس و Macمیباشد . این ادیتور از فریم ورک قدرتمند Qtنیز پشتیبانی میکند و البته نسخه های مختلفی از آن برای پشتیبانی از زبان های Phpو Pythonنیز ارائه شده است . از بدی های این ادیتور میتوان پشتیبانی نکردن از سیستم عامل محبوب ویندوز نام برد .



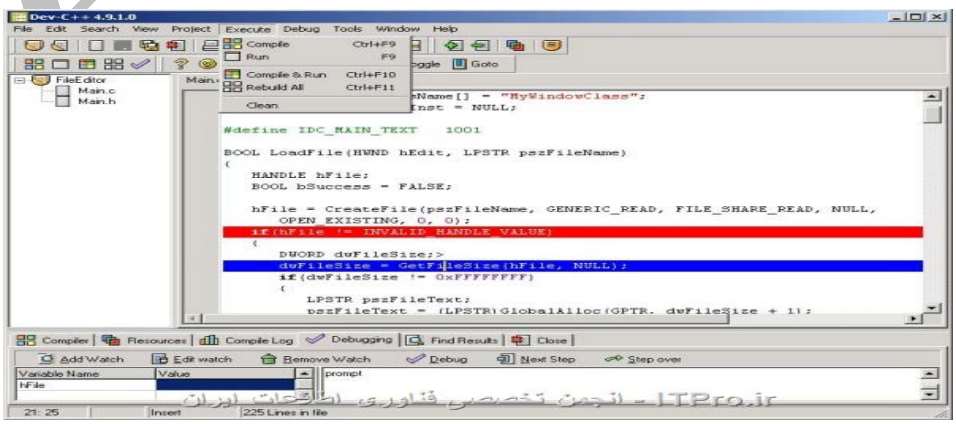
Anjuta Devstudio: نرم افزار ADیکی دیگر از ادیتور های رایگان C-C++میباشد که دارای امکانات بسیاری نظیر مدیریت پروژه , طراح GUI , کنترل نسخه و ..... است و دارای رابط کاربری خوب و حجم کم میباشد.



Code lite: ابزار CL نیز یکی دیگر از ایدیتور های محبوب در بین برنامه نویسان زبان های C-C++ میباشدکه متن باز و رایگان است همچنین دارای حجم کم و پشتیبانی خوبی از سیستم عامل های مختلف و محیطی آسان و راحت میباشد.



Dev C++: ابزار Devیک ادیتور رایگان با ظاهری قدیمی است که فقط از دوکامپایلر MinGwو GCCپشتیبانی میکند . البته دارای امکانات خوبی میباشد اما پشتیبانی نکردن از تمامی نسخه های ویندوز و لینوکس و همچنین پشتیبانی نکردن از Macو نماش کد ها بصورت تک رنگ آن را ادیتوری ضعیف جلوه میدهد اما لازم به ذکر است که برنامه نویسان زیادی از این ادیتور در سرتاسر جهان استفاده میکنند .



۲.راهنمای نصب کامپایلر و اجرای برنامه های C++ , C

در این بخش، به روش نصب کامپایلرهای C++ روی کامپیوترهای شخصی، گوشی های هوشمند و تبلت می پردازیم.

۲–۱ . کامپایلر روی کامپیوترهای شخصی

برای نصب روی کامپیوتر شخصی می توان از هر یک از نرم افزارهای نام برده شده (IDE) در قسمت قبل استفاده نمود. در این جزوه آموزشی نرم افزار مورد استفاده، نرم‌افزار کدبلاکس می باشد.

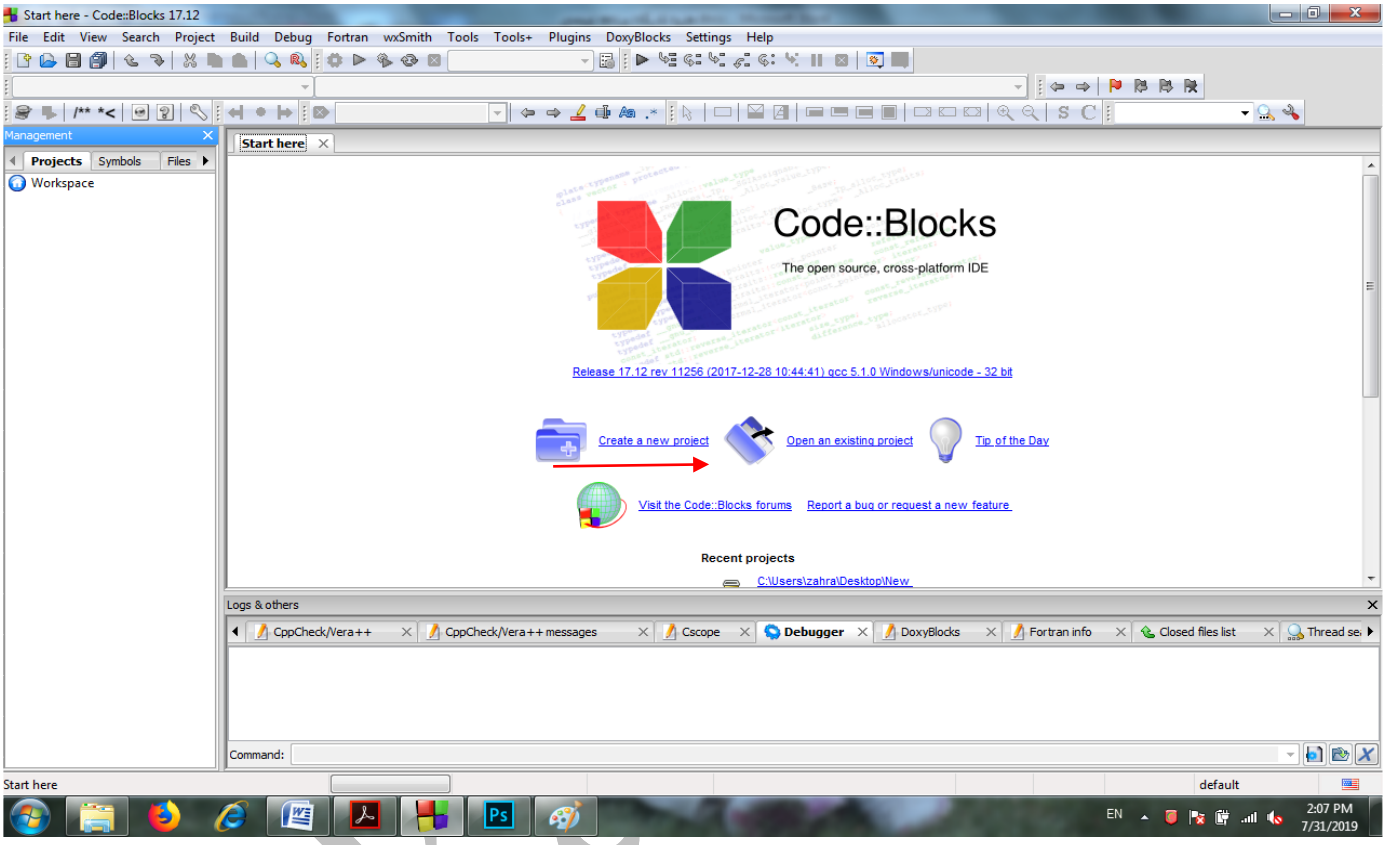
۲–۲ . کامپایلر روی گوشی های هوشمند و یا تبلت

اپلیکیشن‌های زیادی در حوزه برنامه‌نویسی وجود دارند و روز به روز به تعداد آنها افزوده می شود .

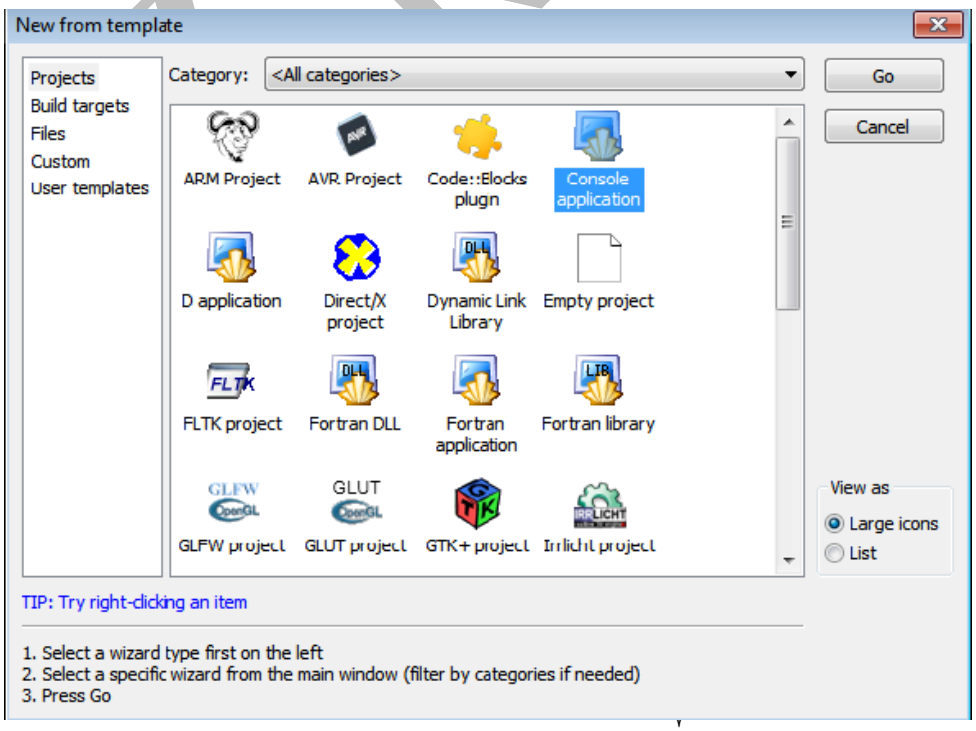
برای نصب نرم افزار روی گوشی های هوشمند خود می توانید به play store و یا بازار مراجعه نمایید سپس اپلیکیشن‌هایی نظیر Dcoder، Code Editor یا Cxxdroid را جستجو نموده و آن را نصب نمایید.

۳. نحوه ایجاد کردن پروژه جدید در نرم‌افزار کدبلاکس و اجرای آن

۱- نرم افزار کدبلاکس را اجرا نموده، روی گزینه Create a new project کلیک نمایید.



۲- در پنجره باز شده انواع گوناگونی از قالب-های پروژه، که هر یک کارایی و عملکرد خاص خود را دارند ، مشاهده می‌کنید.



مثلا:

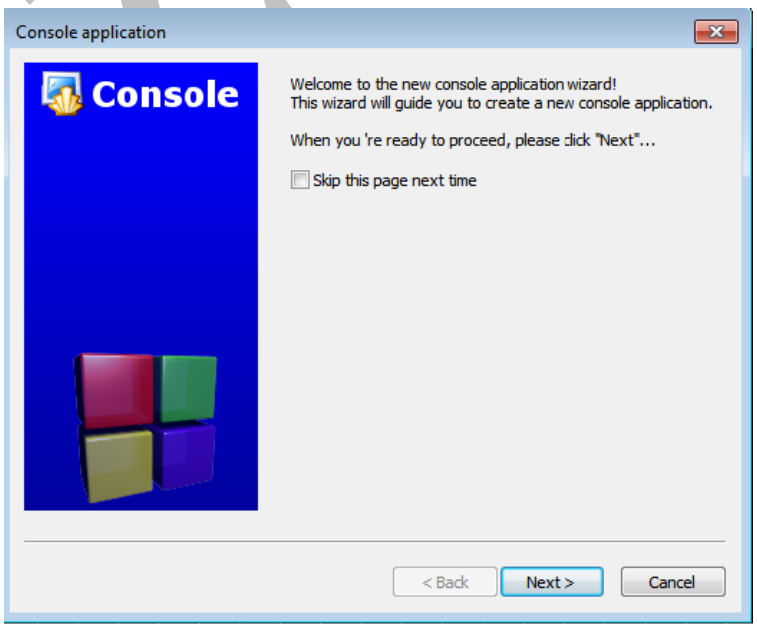
- AVR Project یا ARM Project، جهت برنامه نویسی میکروکنترلرها به کار می‌روند.
- Empty Project صرفا یک پروژه خالی ایجاد می کند.
- Dynamic Link Library برای ایجاد کردن فایل‌های DLL (در بخش‌های بعدی توضیح داده خواهد شد.) به کار می رود.
- Consol Application برای برنامه نویسی کنسولی به کار می رود.

برنامه کنسولی یک نوع از برنامه های کامپیوتری است که در ساده‌ترین حالت گرافیکی موجود در یک سیستم عامل اجرا می شود. در همه سیستم عامل‌ها یک محیط تقریبا بدون گرافیک وجود دارد که مبتنی بر متن می‌باشد. گاهی این محیط را Terminal و یا Consol نیز می نامند. سیستم عامل داس نمونه‌ای از یک محیط کنسولی است. محیط‌های کنسولی GUI ندارند و از طریق Command Line اجرا می‌شوند.

از آنجاییکه محیط کنسول به دور از پیچیدگی‌های گرافیکی است، یادگیری برنامه نویسی در این محیط، راحت‌تر است.

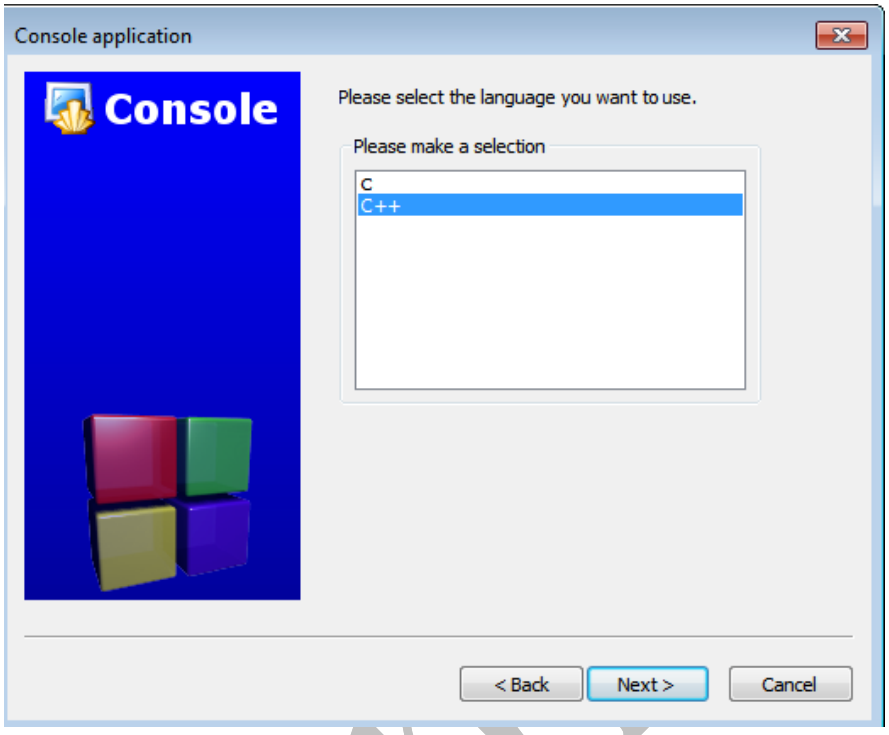
حال با توجه به توضیحات بالا، بر روی Consol Application کلیک می‌کنیم.

پیشنهاد می‌شود دانشجو درباره سایر قالب‌ها نیز تحقیق کرده و نتایج خود را به کلاس ارائه دهد.



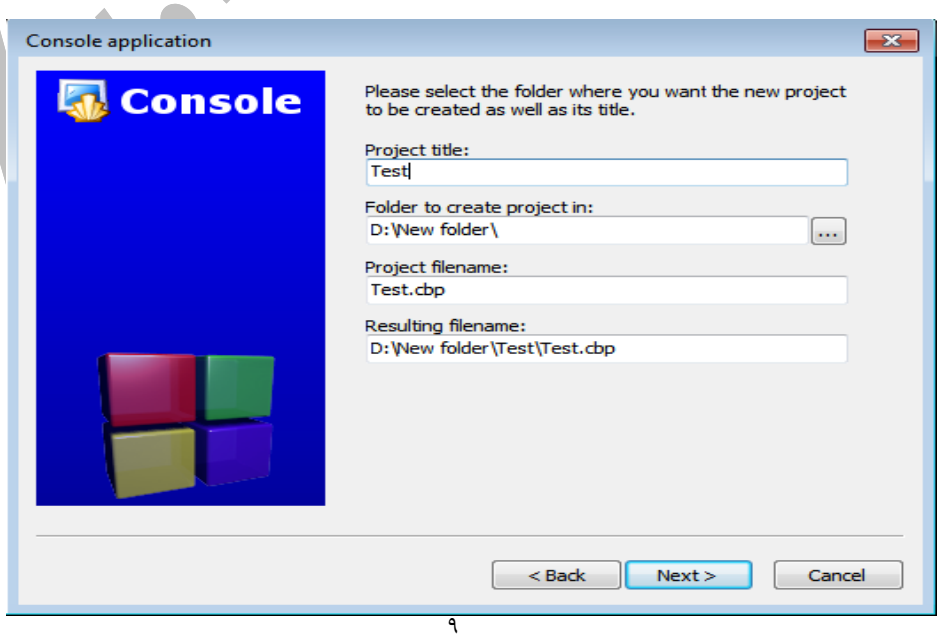
۳- در این پنجره روی Next کلیک کنید.

۴- در پنجره بعدی زبان انتخابی را روی C++ می‌گذاریم و روی Next کلیک می‌کنیم.



۵- در پنجره بعدی در قسمت Project title : نامی برای عنوان پروژه خود وارد نمایید. و در قسمت Folder to create project in : می توانید محل ذخیره سازی پروژه خود را مشخص نمایید. و در نهایت روی Next کلیک کنید.

نکته: پسوند فایل های کدبلاکس cbp. می باشند.



نکته‌ای راجع به عنوان پروژه:

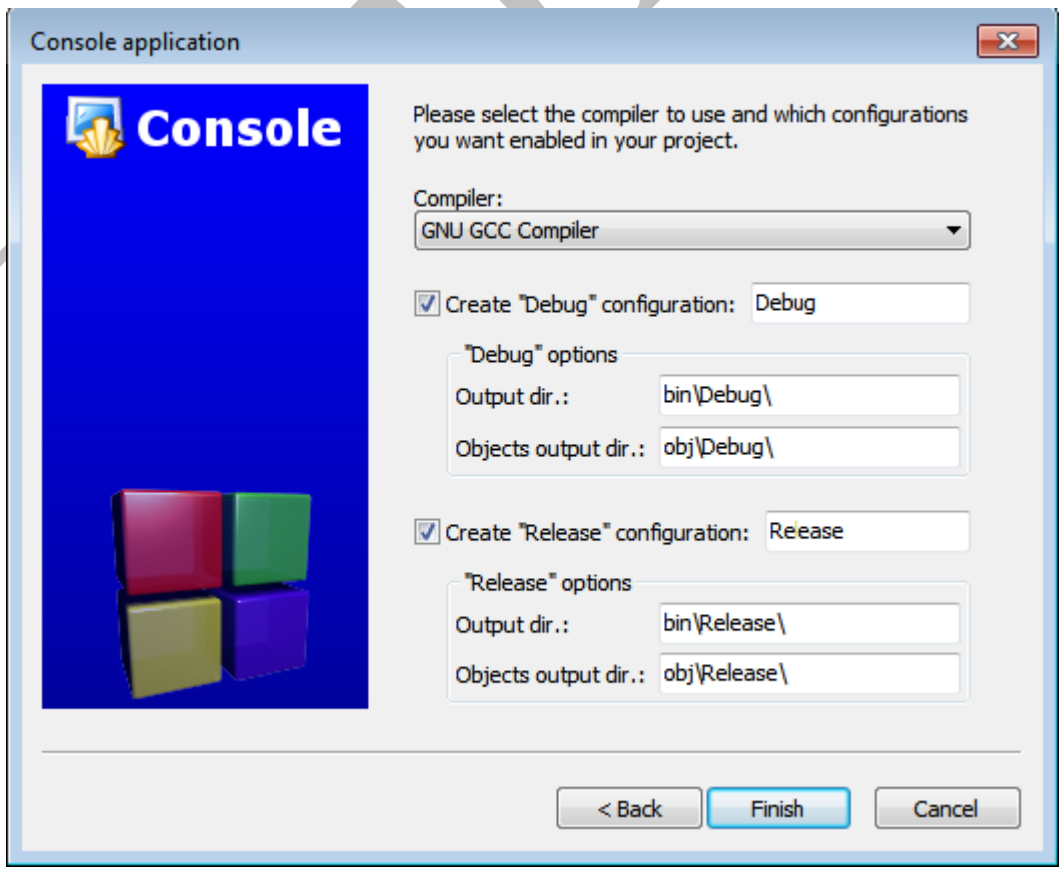
نامی که برای عنوان پروژه خود انتخاب می کنید باید:

- حتما انگلیسی باشد.
- در مسیر ذخیره سازی، فقط باید از نام های انگلیسی استفاده شده باشد.

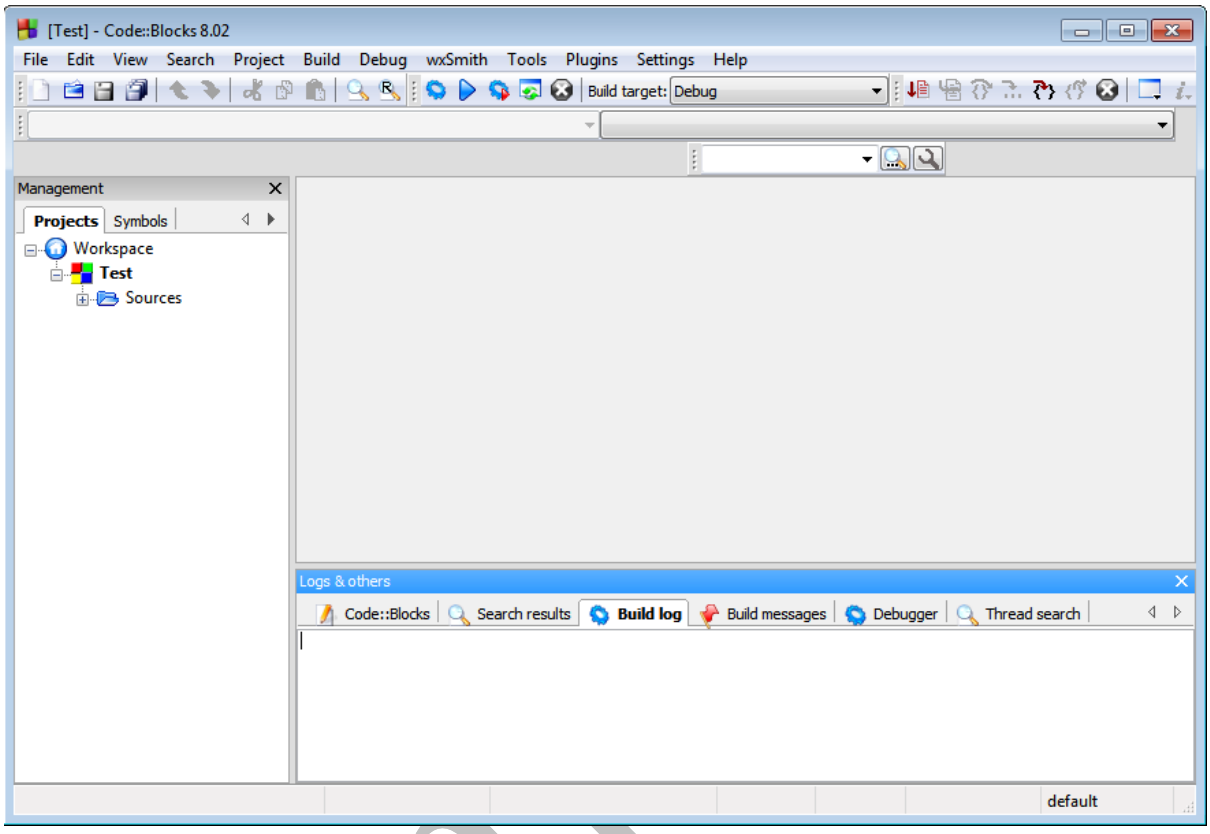
در صورتی که هر یک از موارد بالا رعایت نشود برای رفتن به مراحل بعدی، خطایی داده نمی شود اما زمانی که می خواهید پروژه تان را کامپایل کنید به شما پیغام خطا می دهد.

۶- در این پنجره پیش فرض ها را تغییر نمی دهیم و فقط روی finish کلیک می کنیم.

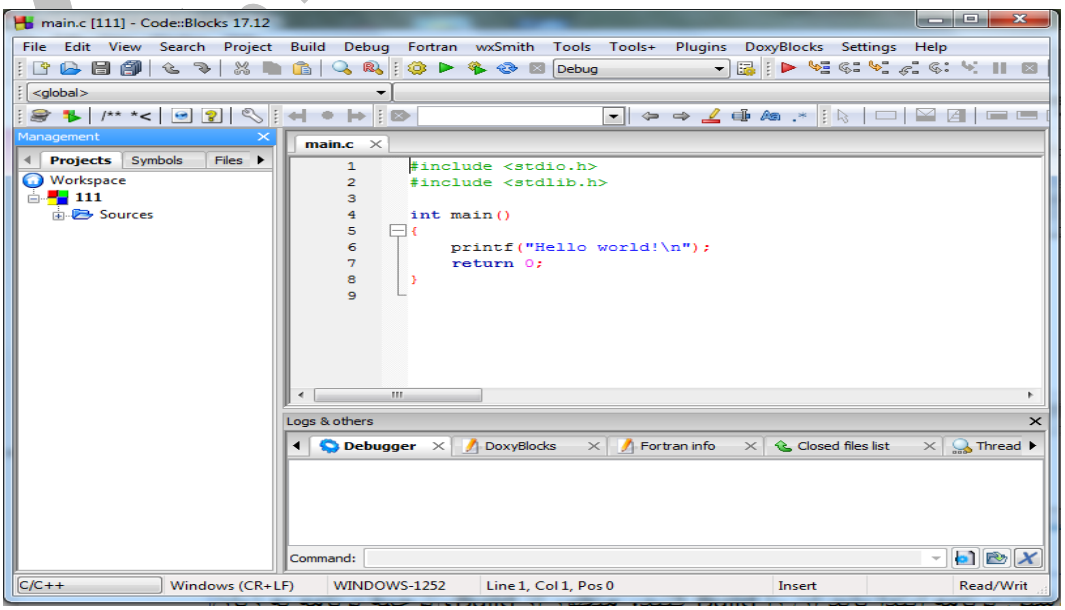
نکته: کامپایلر انتخابی GNU GCC Compiler می باشد.



حال وارد محیط اصلی نرم افزار شده اید و می توانید شروع به برنامه نویسی کنید.



در سمت چپ محیط این نرم افزار، پنل Management را مشاهده می کنید. در این پنل نامی که برای پروژه خود انتخاب کرده بودید را مشاهده می کنید که زیرمجموعه آن پوشه Sources قرار دارد و داخل آن پوشه فایل main.cpp وجود دارد . این فایل در زمان ایجاد پروژه به طور خودکار ساخته می شود.با کلیک کردن روی این فایل و باز کردن آن، محیط برنامه نویسی برای شما در سمت راست باز می شود.



برنامه Hello world به عنوان نمونه برای شما به نمایش درمی آید.حال شما می توانید برنامه مورد نظرتان را در این قسمت تایپ نموده و آن را اجرا نمایید.

نکته: درصورتی که پنل Management وجود نداشت، می توانید آن را از منوی View فعال نمایید.

روش ترجمه و اجرای برنامه:

وقتی برنامه ای نوشته می شود ابتدا باید آن را کامپایل یا ترجمه (ترجمه برنامه به زبان ماشین) نمایید این کار به عهده کامپایلر یا مترجم است.

برای این کار می توانید از سه روش استفاده کنید:

- از منوی Build اولین گزینه را اجرا کنید.
- کلید ترکیبی Ctrl+F9
- از آیکن موجود در بالای صفحه (Build) که به شکل یک چرخ دنده می باشد.

پس از کامپایل کردن برنامه در صورتی که برنامه ارور نداشته باشد نوبت به Run کردن برنامه است.

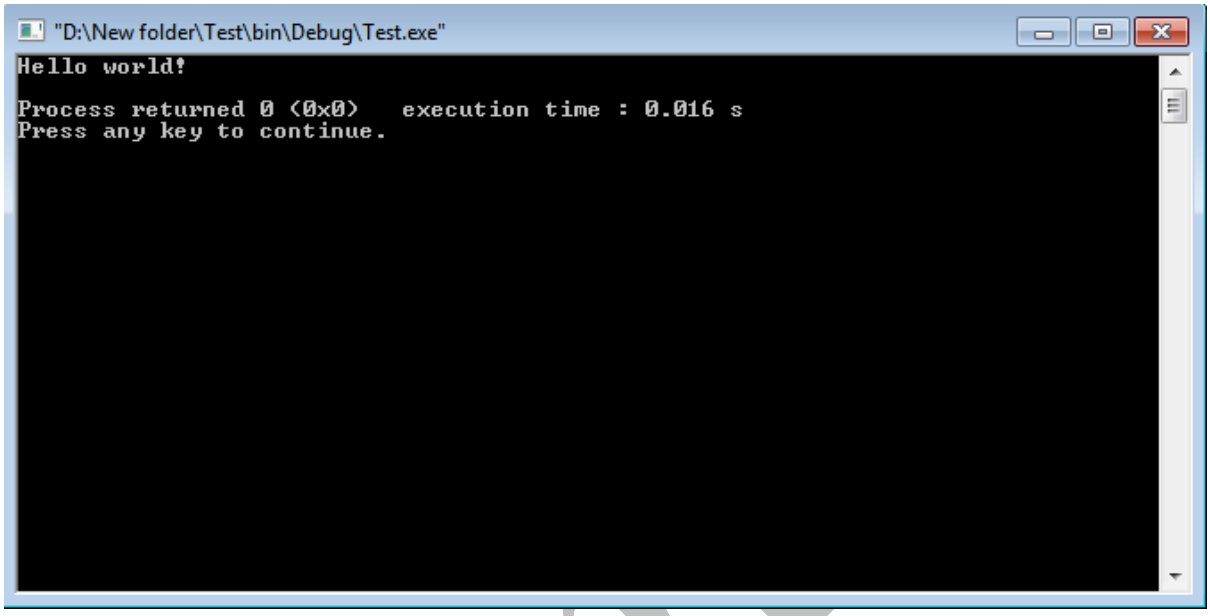
برای این کار می توانید از سه روش استفاده کنید:

- از منوی Build گزینه run را اجرا کنید.
- کلید ترکیبی Ctrl+F10
- از آیکن موجود در بالای صفحه (Run) که به شکل یک مثلث سبز رنگ می باشد.

البته می توان کار کامپایل و اجرا را توامان با هم نیز انجام داد. برای این کار هم می توانید از سه روش زیر استفاده کنید:

- از منوی Build گزینه Buid and run را اجرا کنید.
- کلید F9
- از آیکن موجود در بالای صفحه (Build and Run) استفاده نمایید.(چرخ دنده و مثلث سبز رنگ)

با اجرای برنامه، خروجی در صفحه ای مشکی رنگ به نمایش در می آید.



در صورتی که برنامه خطا داشته باشد، ابتدای خط یا خطوطی از برنامه که دارای اشکال هستند با مربع قرمز رنگ به نمایش در می آیند البته ممکن است خطای مورد نظر واقعا مربوط به آن خط نباشد. مثل خطای عدم گذاشتن سمی کولون در انتهای یک خط .

در قسمت پایین پنجره و در تب Build message خطاها نمایش داده می شوند.

نکته: در صورتی که به هر دلیلی پنل Logs دردسترس نبود، می توانید آن را از منوی View فعال نمایید

نکته ای مهم راجع به کد اجرایی برنامه :

وقتی به سراغ برنامه ذخیره شده می رویم و می خواهیم فایل اجرایی (exe) آن را را باز کنیم بعد از باز کردن، فایل اجرایی سریع بسته می شود.

دلیل این اتفاق این است که برنامه پس از رسیدن به خط آخر خود به خود بسته می شود و پنجره آن هم همینطور. اگر می خواهید صفحه مربوط به خروجی تان را ببینید و باز نگه دارید، می توانید به آخر برنامه تان (قبل از Return 0;) ، یکی از دستورات ورودی را اضافه نمایید.

Scanf() – getch() – getche() – getchar() – system(“pause”) – cin

*main.cpp

```
1   #include <iostream>
2   #include <stdio.h>
3
4   using namespace std;
5
6   int main()
7   {
8       cout << "Hello world!" << endl;
9
10
11      getchar();
12
13
14
15      return 0;
16  }
17
```

۴. خطا یابی و انواع خطاها

خطاها به ۳ دسته کلی تقسیم می شوند:

۱- **خطاهای زمان کامپایل Compile Error**: خطاهایی که در نوشتن دستورات برنامه نویسی ممکن است رخ دهند. به طور کلی رعایت نکردن قواعد زبان برنامه نویسی، سبب بروز چنین خطاهایی می شوند. مثل فراموش کردن سمی کولون در انتهای یک دستور که بسیار رایج است. البته اگر خطا به ننوشتن ; مربوط باشد نرم افزار شماره خط خطا را خط بعد مشخص می‌کند.

این دسته از خطاها را می‌توان در ۳ بخش جای داد:

- **خطای لغوی**: اگر هر یک از کلمات کلیدی موجود در برنامه نادرست تایپ شوند با خطای لغوی مواجه خواهیم شد.(غلط املایی در کلمات کلیدی یا کلمات رزو شده‌ی زبان مورد نظر)
- **خطای دستوری یا گرامری**: همه زبان‌های برنامه‌نویسی یک ساختار دستوری مشخصی دارند. به این ساختار اصطلاحاً syntax گفته می‌شود. اگر در نوشتن یک برنامه به زبان مورد نظر، اصول (دستورات گرامری) آن زبان را رعایت نکنیم، با خطای دستوری، مواجه خواهیم شد.
- **خطای مربوط به عدم وجود فایل‌های کتابخانه‌ای**: یکی دیگر از مقدماتی که به ما مربوط است، نوشتن کلیه هدر فایل‌های یک برنامه و در دسترس بودن کتابخانه‌های استفاده شده در برنامه است. عدم وجود هر یک از فایل‌های کتابخانه ای باعث بوجود آمدن خطا در برنامه می‌شود.

اگر در هنگام کامپایل کردن کد، یک یا چند مورد از موارد ذکر شده، فراهم نباشد، با **کامپایل ارور** مواجه خواهیم شد. وجود هرگونه خطای زمان کامپایل مانع از اجرای برنامه خواهد شد.

۲- **خطاهای زمان اجرا Run Time Error** : هر وقفه ای که در جریان عادی اجرای برنامه پیش آید که معمولا باعث سقط یا توقف برنامه می شود. مثل تقسیم عددی بر صفر

برخی دیگر از خطاهای زمان اجرا برنامه را از کار نمی اندازند بلکه برنامه همچنان کار می کند اما پاسخ های عجیب و نادرست می دهد. مثل عمل سرریزی[۴] برای یک متغیر یا خطای گرد کردن (در ادامه درباره هر یک بحث خواهد شد.)

۳- **خطاهای منطقی Logic Error** : در این نوع خطاها کامپایلر ارور نمی دهد و برنامه اجرا می شود اما نتیجه مورد نظر تولید نمی شود. (خروجی نادرست است) در همچین شرایطی باید برنامه را از لحاظ منطقی و معنایی چک کنیم و با محدود کردن اجرای برنامه و مشاهده مقادیر متغیرهای تعریف شده می توانید خطا را شناسایی کنید.که یافتن چنین خطاهایی نسبت به خطاهای گرامری مشکل تر است. (معمولا به آن باگ[۵] هم گفته می شود.)

[۴]: overflow
[۵].bug

۱-۴. نمونه‌هایی از خطاهای زمان کامپایل

مثال ۱ : برنامه زیر را اجرا نمایید و خطاهای اعلام شده را برطرف نمایید.

main.cpp

```
1   #include <iostream>
2
3   using namespace std;
4
5   int main()
6   {
7       int x,y
8       cin>>x>>y;
9       z=x+y;
10      cout<<z;
11      getche();
12      return 0;
13  }
14
```

Cccc | Build log | Build messages | CppCheck/Vera++ | CppCheck/Vera++ messages

File	Line	Message
C:\Users\zahra...	8	error: expected initializer before 'cin'
C:\Users\zahra...	9	error: 'z' was not declared in this scope
C:\Users\zahra...	9	error: 'y' was not declared in this scope
C:\Users\zahra...	11	error: 'getche' was not declared in this scope
		=== Build failed: 4 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===

پاسخ:

همانطور که قبلا اشاره کرده بودیم، اگر برنامه خطا داشته باشد، در تب Build message، خطای مورد نظر یا لیستی از خطاها نمایش داده می‌شوند. محتوای این تب از ۳ ستون تشکیل شده است:

- ستون اول File: آدرس محل ذخیره‌سازی پروژه را نشان می‌دهد.
- ستون دوم Line: شماره خطی از برنامه که خطا دارد را نشان می‌دهد. با انتخاب هر خط، خطای مربوطه در برنامه با مربع قرمز رنگ نمایش داده می‌شود.

- ستون سوم Message: پیام مربوط به خطا را نشان می‌دهد. (باید توجه داشته باشید که پیغام‌های نمایش داده شده توسط IDEهای مختلف لزوما یکسان نیستند.)

حال به بررسی خطاهای برنامه بالا می‌پردازیم:

خط ۸: عدم رعایت سمی‌کولون در خط قبلی را متذکر می‌شود.

خط ۹: عدم تعریف متغیر z را تذکر می‌دهد.

خط ۱۱: عدم تعریف فایل کتابخانه‌ای تابع getche را تذکر می‌دهد.

مثال ۲: برنامه زیر قرار است یک جدول ضرب ۱۰*۱۰ را نمایش دهد اما به هنگام کامپایل کردن با خطاهایی مواجه می‌شویم که باید آنها را برطرف نماییم.

پاسخ به عنوان تمرین به عهده دانشجو می‌باشد.

main.cpp

```
1   #include <iostream>
2
3
4   using namespace std;
5
6   int main()
7   {
8       for(int i=1 i<=10; i++)
9          {for(int j=1; j<=10; j++)
10              cout<<setw(4)<<i*j;
11          cout<<endl;}
12
13
14      return 0;
15  }
16
```

Code::Blocks | Search results | Cccc | Build log | **Build messages** | CppCheck/Vera++

File	Line	Message
C:\Users\zahra...		In function 'int main()':
C:\Users\zahra...	8	error: expected ';' before 'i'
C:\Users\zahra...	10	error: 'setw' was not declared in this scope
C:\Users\zahra...	11	error: 'endl' was not declared in this scope
		=== Build failed: 3 error(s), 0 warning(s) (0 minute(s), 2 second(s)) ===

مثال ۳: برنامه زیر قرار است ۳ عدد صحیح از ورودی دریافت کرده سپس ماکزیمم مقدار آن را نمایش دهد، اما به هنگام کامپایل کردن با خطاهایی مواجه می‌شویم که باید آنها را برطرف نماییم.

پاسخ به عنوان تمرین به عهده دانشجو می‌باشد.

main.cpp

```
1  #include <iostream>
2
3
4  using namespace std;
5
6  int main()
7  {
8     int a,b,c,max;
9     cin>>a>>b>>c;
10    max=a;
11    if(b>max)
12     max=b;
13    else if(c>max
14     max=c;
15
16    cout<"MAX="<<c;
17
18     return 0;
19
20
```

Logs & others

Code::Blocks | Search results | Cccc | Build log | **Build messages** | CppCheck/Vera++

File	Line	Message
C:\Users\zahra...		In function 'int main()':
C:\Users\zahra...	14	error: expected ')' before 'max'
C:\Users\zahra...	16	error: invalid operands of types 'const char [5]' and 'int' to binary 'operator<<'
C:\Users\zahra...	18	error: expected '}' at end of input
		=== Build failed: 3 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===

۲-۴ . نمونه هایی از خطاهای زمان اجرا

سرریزی عددی

نوع صحیح **long** یا نوع ممیز شناور **double** محدوده وسیعی از اعداد را می توانند نگهداری کنند. به بیان ساده تر، متغیری که از نوع **long** یا **double** باشد، گنجایش زیادی دارد. اما حافظه رایانه ها متناهی است. یعنی هر قدر هم که یک متغیر گنجایش داشته باشد، بالاخره مقداری هست که از گنجایش آن متغیر بیشتر باشد. اگر سعی کنیم در یک متغیر مقداری قرار دهیم که از گنجایش آن متغیر فراتر باشد، متغیر سرریزه می شود. مثل یک لیوان آب که اگر بیش از گنجایش آن در لیوان آب بریزیم، سرریز می شود. در چنین حالتی می گوییم که خطای سرریزی رخ داده است.

Data Types	Sizes in byte	Sizes in bits	Range formula 2^n-1	Ranges
int	4 bytes	32bits	2^{32}-1	-2,147,483,648 to 2,147,483,647
unsigned int	4 bytes	32 bits	2^{32}-1	0 to 4294967295
float	4 bytes	32 bits	2^{32}-1(5 points)	3.4×10^{-38} to $3.4 \times 10^{+38}$
double	8 bytes	64 bits	2^{64}-1(15 points)	1.7×10^{-308} to $1.7 \times 10^{+308}$
long double	10 bytes	80 bits	2^{80}-1(19 points)	1.7×10^{-4932} to $1.7 \times 10^{+4932}$
char	1 byte	8 bits	2^{8}-1	0 to 255

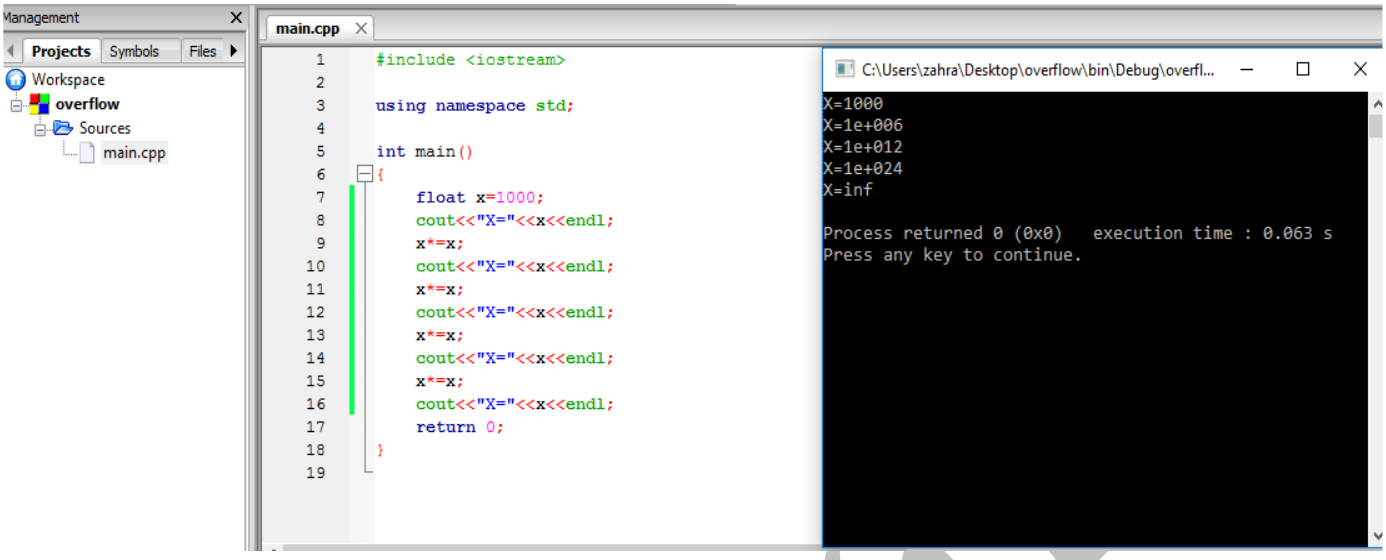
جدول محدوده انواع داده ها در C++

۱-۲-۴. سرریزی عدد اعشاری

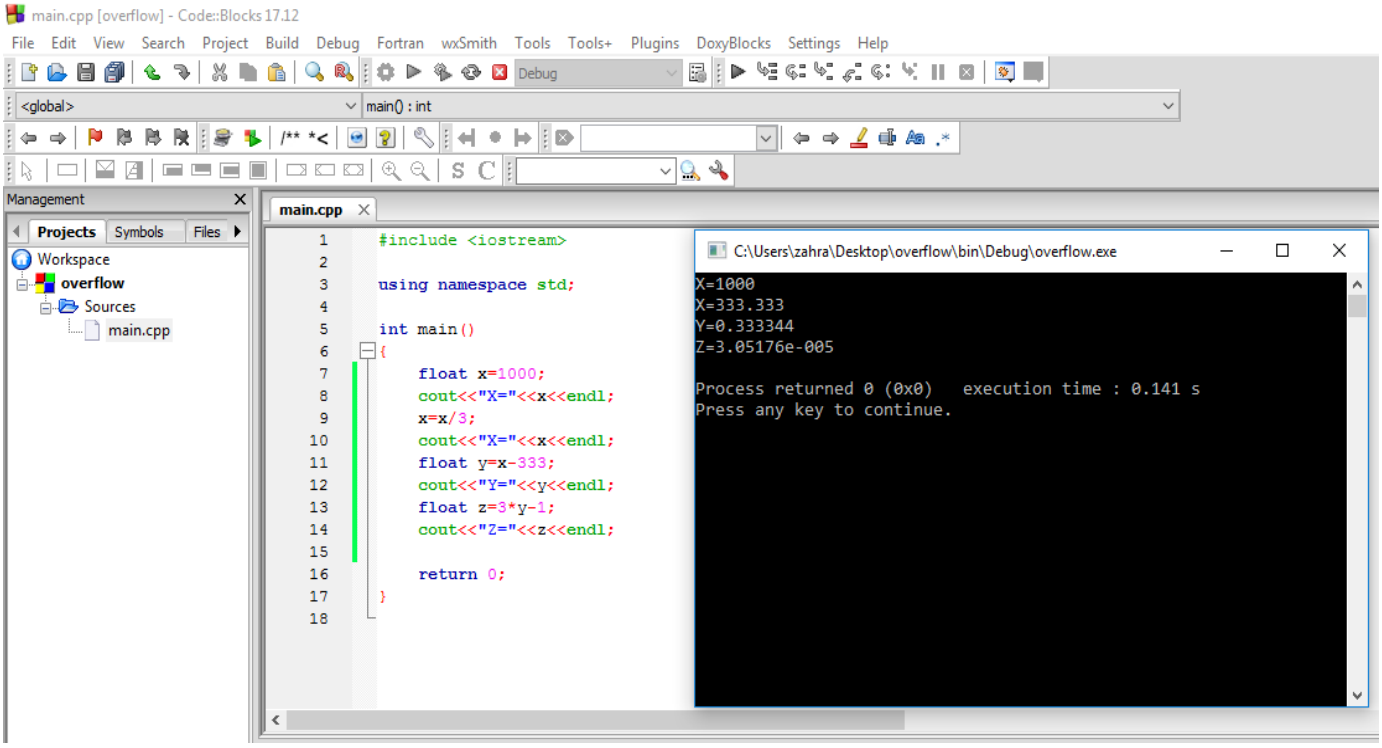
این برنامه x را به طور مکرر به توان می رساند تا اینکه سرریز شود.در آخرین خط خروجی **inf** به معنی بی نهایت [۶] یا بی انتها است.

سرریزی عدد صحیح به عنوان تمرین به عهده دانشجو می باشد.

[۶] infinity



۲-۲-۴. خطای گرد کردن



توضیحات برنامه و دلیل بروز خطا، به عهده دانشجو می باشد.

۳–۴ . روش های یافتن خطاهای منطقی و برطرف کردن آنها (خطایابی در کدبلاکس)

خطایابی منطقی یکی از دشوارترین مراحل خطایابی است. این مرحله از خطایابی، یکی از پرهزینه ترین مراحل برنامه سازی است. یکی از ساده ترین و در عین حال پراستفاده ترین راهکارهای خطایابی منطقی، آزمودن برنامه در برابر تعداد مشخصی ورودی و خروجی آزمون است. به عبارت دیگر، داده آزمون برای یک برنامه عبارت است از تعدادی مشخص ورودی به همراه خروجی های مرتبط. در صورتی که برنامه حتی به ازای یک مورد از ورودی های آزمون، خروجی مناسب تولید نکند، می توان مطمئن بود که برنامه دارای خطای منطقی است.

نکته مهم در خطایابی این است که هنگامی که به یک خطا رسیدیم، نباید ناامید شویم!درست است که یک برنامه ایده آل باید بدون خطا باشد، اما رسیدن به چنین ایده آلی در عمل، اگر نگوییم ناممکن، اما دشوار است. هنگامی که شما خطای برنامه خود را فهمیده اید، در راستای تصحیح آن تلاش می کنید و این نکته، خود ارزشمند است.

ساده ترین و معمول ترین شیوه برای خطایابی برنامه‌ها، اضافه کردن عبارت های متوالی printf() در طول برنامه است. به‌چنین قطعه کدهایی، قطعه کدهای رهگیر گفته می شود. (از طریق دستورات خروجی یا دستورات چاپی می توانید مقادیر متغیرها را در هر قسمت از برنامه به دست آورید و اینگونه به محل بروز خطا برسید.)

استفاده از روش فوق، فرایندی زمانبر است، به ویژه برای مواردی که تعداد خطاها بیشتر یا پیچیدگی بیشتری داشته باشند. به همین منظور، لازم است تا یک روش مناسب برای خطایابی استفاده کنیم. اولین چیزی که به ذهن می رسد، برنامه ای است که خط به خط برنامه مورد نظر را اجرا کند و پس از اجرای هرخط، مقادیر متغیر مورد نیاز را به ما نشان دهد. شاید بهتر باشد، اجرای برنامه را تا خط مشخصی ادامه داده و در آن خط برای وارسی مقادیر متغیرهای برنامه متوقف شود و پس از بررسی، اجرای برنامه را از سر گیرد.خوشبختانه چنین برنامه ای وجود دارد که به آن خطایاب یاDebugger گفته می شود. یکی از خطایاب های بسیار قدرتمند و البته متن باز که درCode::Blocks نیز استفاده می‌شود،GDB یا GNU DeBugger است.

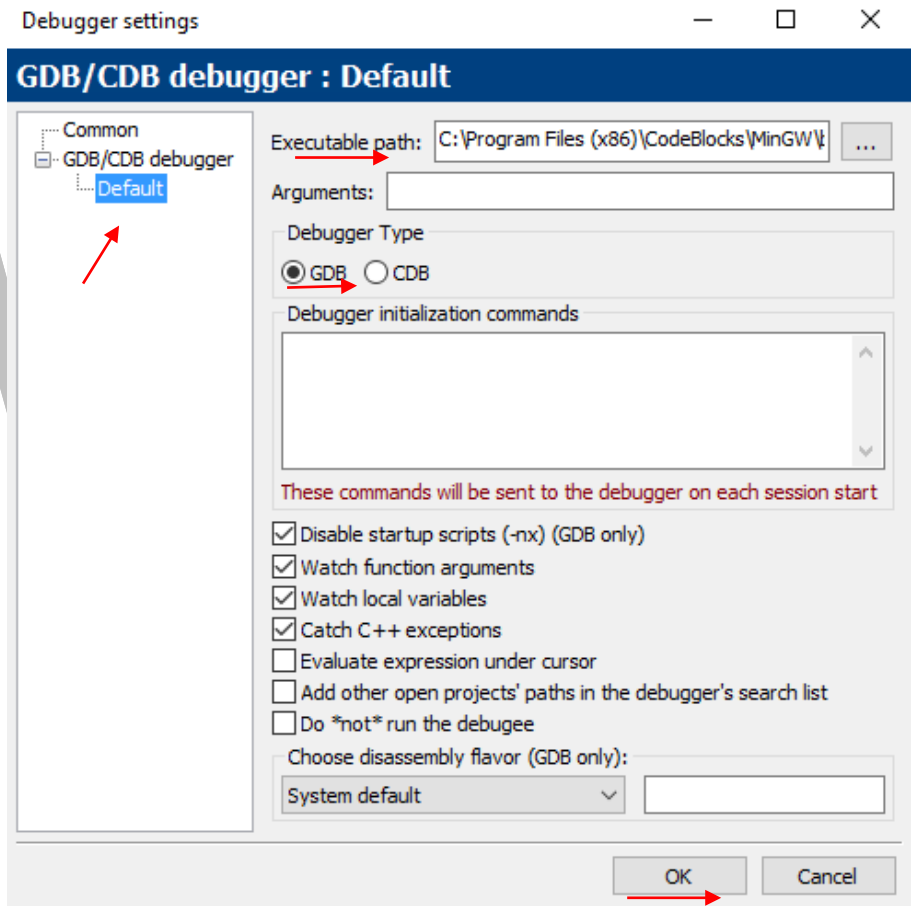
تنظیمات لازم برای خطایاب یا Debugger:

- In the Code::Blocks IDE, navigate **Settings** -> **Debugger**
- In the tree control at the right, select **Common** -> **GDB/CDB debugger** -> **Default**.
- Then in the dialog at the left you can enter **Executable path** and choose **Debugger type** = GDB or CDB, as well as configuring various other options.

C:\Program Files\CodeBlocks\MinGW\bin\gdb32.exe

درصورتیکه خطایاب (Debugger) در کدبلاکس شما فعال نبود، می توانید به روش زیر آن را فعال نمایید:

- وارد منوی **Settings** شده و گزینه **Debuggger…** را انتخاب نمایید.
- در سمت چپ پنجره باز شده به ترتیب روی **Common -> GDB/CDB debugger -> Default.** کلیک می‌کنیم.
- سپس در فیلد Executable path ، مسیر ذخیره‌سازی فایل اجرایی خطایاب (**C:\Program Files\CodeBlocks\MinGW\bin\gdb32.exe**) را می‌دهیم.
- و در نهایت نوع خطایاب (**Debugger type**) را روی گزینه **GDB** تنظیم و Ok می‌کنیم.



این روش با محدود کردن اجرای برنامه از طریق Breakpoint ها و مشاهده مقادیر متغیرهای تعریف شده، خطا را شناسایی می‌کند. برای دیدن مقدار متغیرها از منوی Debug و زیرمنوی Debugging Windows گزینه Watches را انتخاب کنید. در پنجره باز شده نام متغیرها و مقادیر آنها را می توانید مشاهده کنید و به این ترتیب محل بروز خطا را تشخیص دهید.

در قدم بعد باید محدوده اجرای برنامه را مشخص کرد. به این منظور ازBreakpoint ها استفاده می شود. زمانی که برای دستوری Breakpoint مشخص شود برنامه با رسیدن به این دستور مکث می کند و وارد محیط دیباگ می شود. سپس می توان از دستوراتStep و سایرموارد استفاده کرد.

برای تعیین Breakpoint یک دستور می توان از روش های زیر استفاده کرد:

- بر روی خط دستور راست کلیک کرده و گزینه Add Breakpoint
- کلیک در ابتدای خط دستور
- بر روی خط دستور کلیک کرده و کلید F5
- بر روی خط دستور کلیک کرده، از منوی Debug گزینه Toggle breakpoint

در ابتدای خطی که برای آن breakpoint تعیین می کنید یک دایره قرمز رنگ قرار می گیرد.

بعد از تعیین Breakpoint ها، برای اجرای برنامه می توانید:

- از منویDebug گزینه Start/Continue را انتخاب کنید
- از کلید F8 استفاده نمایید.
- یا از آیکن Debug/Continue که در بالای صفحه به شکل مثلث قرمز رنگ می باشد استفاده کنید. 

برنامه تا رسیدن به اولین Breakpoint اجرا می‌شود و صفحه خروجی نمایش داده می شود.

در ادامه، موارد زیر به شما این امکان را می دهند که مراحل پیشروی دستورات را همان گونه که تمایل دارید انتخاب کنید:

- Next line: این گزینه برنامه را خط به خط اجرا می کند. می توانید از منویDebug انتخاب کنید یا کلیدF7 را فشار دهید.
- Step into: عملکرد این گزینه شبیهNext line هست با این تفاوت که درNext line زمانی که با دستور فراخوانی یک تابع روبرو شود، وارد آن تابع نمی شود و مکان نما به خط بعدی منتقل می شود. اما درStep into اجرای برنامه به آن تابع منتقل می شود و مکان نما ابتدای اولین دستور تابع قرار می گیرد و بعد از اجرای کامل آن تابع به تابع فراخوانی کننده بر می گردد. این گزینه را می توانید از منویDebug انتخاب کنید یا از کلیدهای ترکیبیShift+F7 استفاده کنید.
- Step out: این گزینه از جهت منطقی عملکردی بر عکسStep into دارد. اگر با اجرای step into یا دلیل دیگری وارد یک تابع شده اید می توانید با اجرای step out از آن خارج شده و مکان نما به تابع فراخوانی کننده منتقل می شود. این گزینه را می‌توانید از منویDebug انتخاب کنید یا از کلیدهای ترکیبیCtrl+F7 استفاده کنید.
- Next instruction: این گزینه سراغ دستورالعمل بعدی می رود. و همه دستورات را مرحله به مرحله چک می کند.
- Step into instruction : عملکردی مشابه Step into دارد.
- Run to cursor: مکان نما را روی خطی که قصد دارید عمل اجرا تا آن مکان انجام شود قرار دهید. سپس از منویDebug گزینهRun to cursor را انتخاب کنید یا کلیدF4 را فشار دهید. در آغاز خط مشخص شده، مثلث زرد رنگی قرار می گیرد و برنامه بلافاصله تا این دستور اجرا شده و سپس مکث می کند.

هنگامی که خطایاب اجرای برنامه در یک خط را متوقف می کند، در کنار آن خط یک نشانگر زردرنگ قرار می گیرد.

در این قسمت، همزمان با مشاهده چگونگی اجرای برنامه، می توان از پنجره Watches مقادیر متغیرها را نیز مشاهده نمود.

در نهایت برای خروج از این حالت باید روی آیکن Stop Debugger موجود در بالای صفحه یا از منوی Debug گزینه مورد نظر را انتخاب نمود.

در این بخش، ما از خطایاب ها برای یافتن خطای برنامه در حین اجرا استفاده کردیم. یکی دیگر از مزایایی که خطایاب ها می توانند داشته باشند، فهمیدن عملکرد برنامه ها و لگوریتم ها با استفاده از آن ها است.

۱-۳-۴. نمونه هایی از خطاهای منطقی

عنوان برنامه اول: فاکتوریل

هدف: محاسبه‌ی فاکتوریل عدد وارد شده

توضیح برنامه: برای بدست آوردن فاکتوریل یک عدد، لازم است حاصلضرب آن عدد تا یک را محاسبه نماییم. از این رو کافیست از یک حلقه For یا While استفاده نماییم و زیرمجموعه آن، دستور ضرب اعداد (به این صورت که هر دو عدد در هم ضرب می‌شوند و حاصل آن در متغیر دیگری مثل fact قرار می‌گیرند تا اینکه به عدد انتهای بازه حلقه برسیم و حلقه خاتمه یابد) را داشته باشیم.

main.cpp

```
1   #include <iostream>
2   #include <conio.h>
3
4   using namespace std;
5
6   int main()
7   {    int number,fact;
8        cout<<"Enter your number:";
9        cin>>number;
10       for(int i=1; i<=number; i++)
11           fact*=i;
12
13       cout << "fact="<<fact;
14       getch();
15       return 0;
16  }
17
```

```
"C:\Users\zahra\Desktop\fact s\bin\Debug\fact s.exe"
Enter your number:4
fact=104502096
```

اما وقتی به خروجی برنامه دقت می کنیم، مشاهده می‌شود که محاسبه فاکتوریل عدد وارد شده، صحیح نمی‌باشد. در نتیجه باید به دنبال یک یا چند خطای منطقی در روند اجرای برنامه باشیم.

حال به شرح مراحل خطایابی که در بخش قبلی توضیحات کامل آن داده شده است می پردازیم:

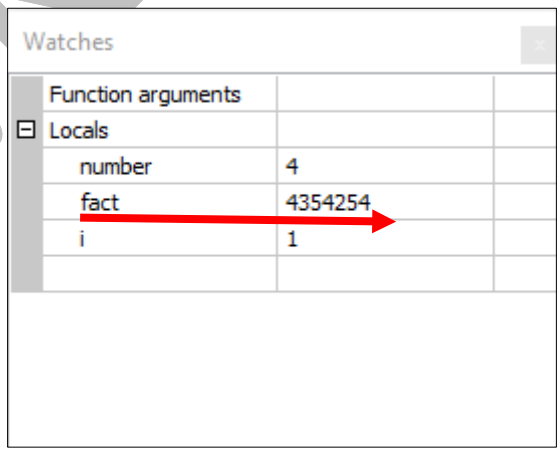
۱- ابتدا یک Breakpoint در یکی از خطوط ابتدایی برنامه (مثلا خط ۹) قرار می دهیم تا خط به خط برنامه را اجرا کنیم.

۲- برای اجرای برنامه بر روی Debug/Continue  کلیک می کنیم تا پنجره خروجی باز شود.

۳- برای اینکه تغییرات مربوط به مقدار متغیرها را نیز مشاهده کنیم، پنجره Watches را نیز باز می‌کنیم.

۴- حال با استفاده از کلید Next Line خط به خط برنامه را اجرا می کنیم و در عین حال، پنجره خروجی و Watches را نیز مورد توجه قرار می‌دهیم تا به خطایمان پی ببریم.

در همان تکرار اول حلقه، با توجه به پنجره Watches ، متوجه مقدار زباله ای که در متغیر fact قرار گرفته می‌شویم و همین مقدار زباله، منجر به تولید نتیجه نادرست می‌شود. پس نتیجه می‌گیریم که باید متغیر fact را حتما مقداردهی اولیه نماییم.



عنوان برنامه دوم: محاسبه فاکتوریل

هدف: محاسبه‌ی فاکتوریل هر عددی که وارد می شود، تا زمانی که عدد ۱- وارد شود.

توضیح برنامه: برای اینکه اعداد، پشت سر هم وارد شوند، به یک حلقه همواره درست نیاز داریم که زیرمجموعه آن، اعداد را از ورودی دریافت کنیم و شرط خاتمه حلقه را وارد شدن عدد ۱- قرار می دهیم. درصورتی که عدد وارد شده مخالف ۱- باشد، وارد حلقه دیگری جهت محاسبه فاکتوریل می‌شود. و در نهایت مقدار محاسبه شده یعنی مقدار متغیر fact چاپ می شود.

main.cpp

```
1   #include <iostream>
2
3   using namespace std;
4
5   int main()
6   {
7       int number;
8       long fact=1;
9       while(1)
10          {
11      cout<<"Enter The Number:";
12          cin>>number;
13      if(number==-1) break;
14          for(int i=2; i<=number; i++)
15              fact*=i;
16          cout<<"fact="<<fact<<endl;
17      }
18      return 0;
19  }
20
```

```
C:\Users\zahra\Desktop\fact\bin\Debug\fact.exe
Enter The Number:2
fact=2
Enter The Number:3
fact=12
Enter The Number:4
fact=288
Enter The Number:-1

Process returned 0 (0x0)   execution time : 18.934 s
Press any key to continue.
```

اما وقتی به خروجی برنامه دقت می کنیم، مشاهده می‌شود که محاسبه فاکتوریل اولین عدد درست و مابقی نادرست هستند. در نتیجه باید به دنبال یک یا چند خطای منطقی در روند اجرای برنامه باشیم.

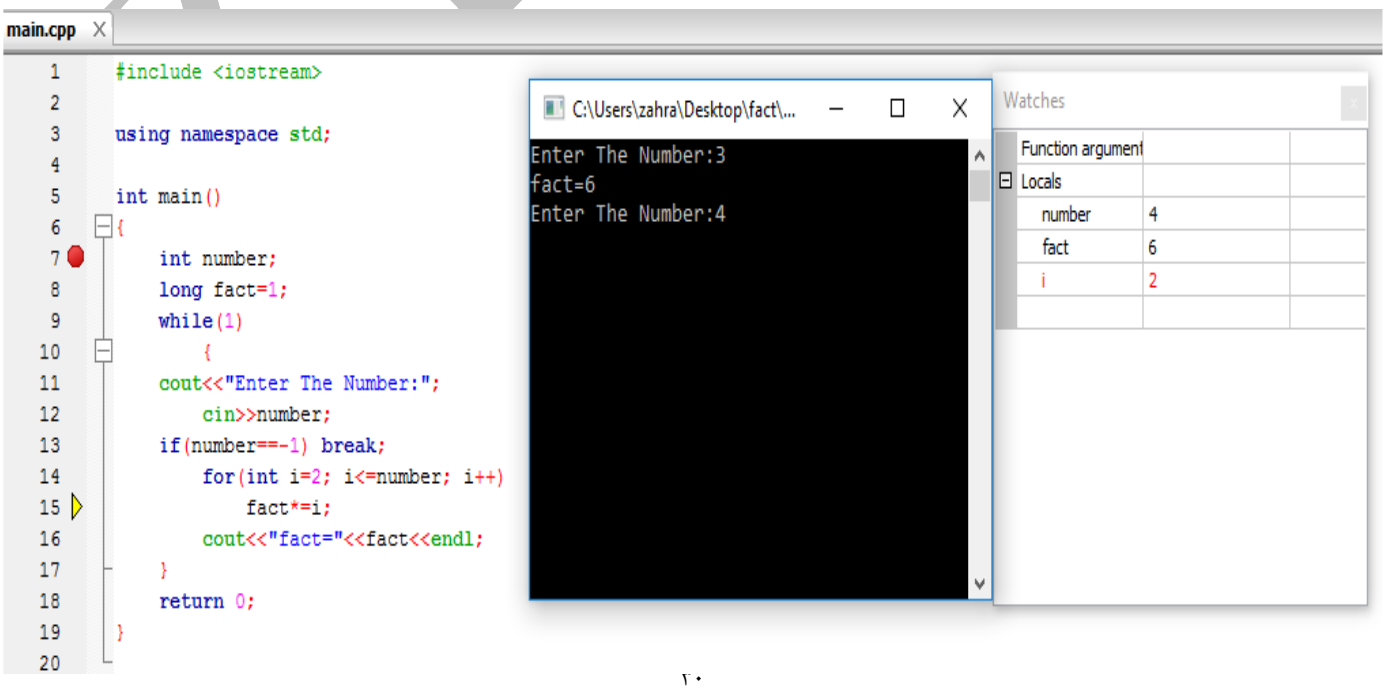
حال به شرح مراحل خطایابی می پردازیم:

۱- ابتدا یک Breakpoint در یکی از خطوط ابتدایی برنامه (مثلا خط ۷) قرار می دهیم تا خط به خط برنامه را اجرا کنیم.

۲- برای اجرای برنامه بر روی Debug/Continue  کلیک می کنیم تا پنجره خروجی باز شود.

۳- برای اینکه تغییرات مربوط به مقدار متغیرها را نیز مشاهده کنیم، پنجره Watches را نیز باز می‌کنیم.

۴- حال با استفاده از کلیدهای Next Line یا Step Into خط به خط برنامه را اجرا می کنیم و در عین حال، پنجره خروجی و watches را مورد توجه قرار می دهیم تا به خطایمان پی ببریم. وقتی عدد اول را وارد می‌کنیم، برنامه به درستی اجرا شده و خروجی درست را نیز تولید می‌کند اما وقتی عدد دوم را وارد می کنیم و به خط ۱۵ برنامه (محاسبه fact) می رسیم از آنجایی که مقدار متغیر fact همان مقدار مربوط به عدد قبل است در نتیجه محاسبات و خروجی ما نادرست می‌شوند.



پس ما باید در مقدار متغیر fact تجدید نظر کنیم و در انتهای حلقه یا ابتدای حلقه، هر بار مقدار متغیر fact را به مقدار ۱ تغییر دهیم تا محاسبات سایر اعداد نیز ارزش درستی پیدا کنند.

main.cpp

```
1   #include <iostream>
2
3   using namespace std;
4
5   int main()
6   {
7       int number;
8       long fact=1;
9       while(1)
10          {
11      cout<<"Enter The Number:";
12          cin>>number;
13      if(number==-1) break;
14          for(int i=2; i<=number; i++)
15              fact*=i;
16          cout<<"fact="<<fact<<endl;
17          fact=1;
18      }
19      return 0;
20  }
21
```

عنوان برنامه سوم: عدد اول

هدف: تشخیص اول بودن هر عددی که وارد می شود، تا زمانی که عدد ۱- وارد شود.

توضیح برنامه: کد برنامه به همراه خروجی آن در ادامه آمده است. توضیح برنامه و تشخیص خطای آن به عهده دانشجو می باشد.

main.cpp

```
5   int main()
6   {
7       int number,flag=0;
8       while(1)
9           {
10      cout<<"Enter The Number:";
11          cin>>number;
12      if(number==-1) break;
13          for(int i=2; i<=number/2; i++)
14              if(number%i==0)
15                  flag++;
16
17           if(flag==0)
18              cout<<"The number is prime"<<endl;
19           else
20              cout<<"The number is not prime"<<endl;
21
22      }
23      return 0;
24  }
25
```

```
C:\Users\zahra\Desktop\prime\bin\Debug\prime.exe
Enter The Number:7
The number is prime
Enter The Number:8
The number is not prime
Enter The Number:2
The number is not prime
Enter The Number:23
The number is not prime
Enter The Number:41
The number is not prime
Enter The Number:-1

Process returned 0 (0x0)   execution time : 75.349 s
Press any key to continue.
```

۵.کامپایل جداگانه توابع و فایل DLL

گاهی اوقات لازم می شود که ما از توابعی که قبلا ساخته ایم در یک برنامه جدید استفاده کنیم و فقط فراخوانی آن تابع را به برنامه اضافه کنیم بدون اینکه تعریف و بدنه تابع را مجددا بنویسیم. برای این کار باید تعریف و بدنه توابع را در فایل های جداگانه ای قرار داد و سپس به برنامه اصلی که آن توابع را به کار می گیرد الصاق کرد.

اغلب این طور است که تعریف و بدنه توابع در فایل های جداگانه ای قرار می گیرد. این فایل ها به طور مستقل کامپایل می شوند و سپس به برنامه اصلی که آن توابع را به کار می گیرد الصاق[۷]می شوند. توابع کتابخانه C++ استاندارد به همین شکل پیاده سازی شده اند و هنگامی که یکی از آن توابع را در برنامه هایتان به کار می برید باید با دستور راهنمای پیش پردازنده، فایل آن توابع را به برنامه تان ضمیمه کنید مانند توابع ورودی و خروجی cin و cout، که در فایل کتابخانه ای iostream تعریف شده اند. این کار چند مزیت دارد. اولین مزیت «مخفی سازی اطلاعات» است. یعنی این که توابع لازم را در فایل جداگانه ای تعریف و کامپایل کنید و سپس آن فایل را به همراه مشخصات توابع به برنامه نویس دیگری بدهید تا برنامه اصلی را تکمیل کند. به این ترتیب آن برنامه نویس از جزییات توابع و نحوه اجرای داخلی آنها چیزی نمی داند و فقط می داند که چطور می تواند از آنها استفاده کند. در نتیجه اطلاعاتی که دانستن آنها برای برنامه نویس ضروری نیست از دید او مخفی می ماند. تجربه نشان داده که پنهان سازی اطلاعات، فهمیدن برنامه اصلی را آسان می کند و پروژه های بزرگ با موفقیت اجرا می شوند.

مزیت دیگر این است که توابع مورد نیاز را می توان قبل از این که برنامه اصلی نوشته شود، جداگانه آزمایش نمود. وقتی یقین کردید که یک تابع مفروض به درستی کار می کند، آن را در یک فایل ذخیره کنید و جزییات آن تابع را فراموش کنید و هر وقت که به آن تابع نیاز داشتید با خیالی راحت از آن در برنامه هایتان استفاده نمایید. نتیجه این است که تولید توابع مورد نیاز و تولید برنامه اصلی، هم زمان و مستقل از هم پیش می رود بدون این که یکی منتظر دیگری بماند. به این دیدگاه «بسته بندی نرم افزار » می گویند.

سومین مزیت این است که در هر زمانی به راحتی می توان تعریف توابع را عوض کرد بدون این که لازم باشد برنامه اصلی تغییر یابد. فرض کنید تابعی برای مرتب کردن فهرستی از اعداد ایجاد کرده اید و آن را

[۷] Linking

جداگانه کامپایل و ذخیره نموده اید و در یک برنامه کاربردی هم از آن استفاده برده اید. حالا هر گاه که الگوریتم سریع تری برای مرتب سازی یافتید، فقط کافی است فایل تابع را اصلاح و کامپایل کنید و دیگر نیازی نیست که برنامه اصلی را دست کاری نمایید.

چهارمین مزیت هم این است که می توانید یک بار یک تابع را کامپایل و ذخیره کنید و از آن پس در برنامه های مختلفی از همان تابع استفاده ببرید. وقتی شروع به نوشتن یک برنامه جدید می کنید، شاید برخی از توابع مورد نیاز را از قبل داشته باشید. بنابر این دیگر لازم نیست که آن توابع را دوباره نوشته و کامپایل کنید. این کار سرعت تولید نرم افزار را افزایش می دهد.

برای این منظور بهتر است با نحوه ساخت و بکارگیری فایل DLL [۱](کتابخانه پیوند پویا) که در ادامه توضیح داده شده است، آشنا شویم.

[۱]. Dynamic Link Library

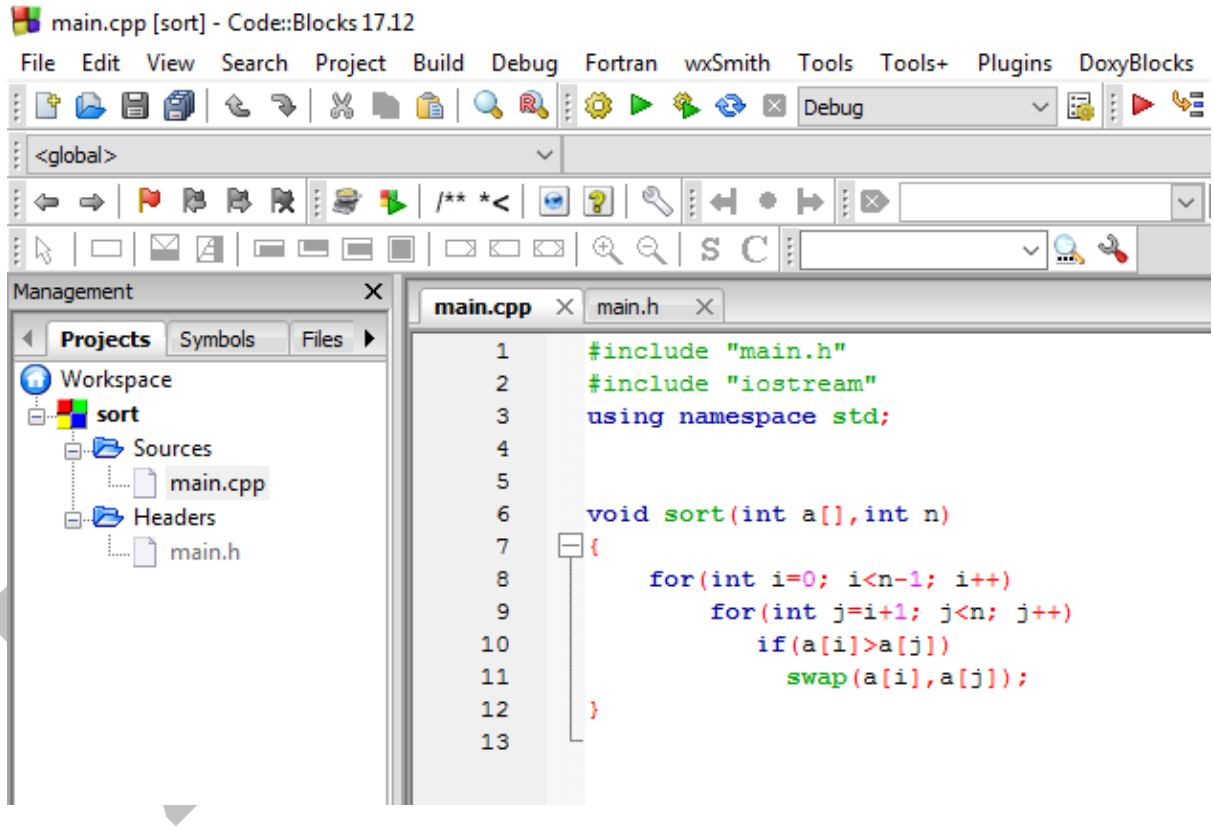
۱–۵ . نحوه ساخت فایل DLL

برای ساخت فایل DLL به روش زیر عمل می کنیم:

File ⟶ New Project ⟶ Dynamic Link Library

ادامه مراحل شبیه به ساخت پروژه جدید برنامه نویسی می باشد که آنها را ادامه می دهیم. ⟵ Finish

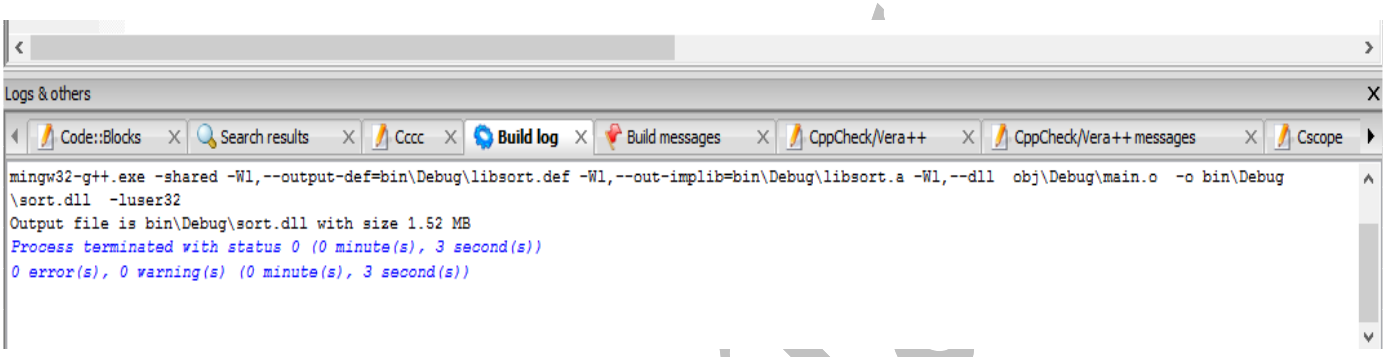
در پنل Management مشاهده می کنیم که بر اساس نامی که برای فایل dll انتخاب کرده‌ایم (مثلا sort) ، یک فایل dll ساخته شده است. که شامل دو زیرمجموعه به نام های Sources و Headers می‌باشد.



در ادامه به سراغ فولدر Sources می رویم و فایل main.cpp را باز می کنیم سپس همه کدها به جز #include “main.h” را پاک می کنیم و کدهای مربوط به تعریف تابع (مثلا تابع مرتب سازی) را می-نویسیم.(می توانیم تعریف چندین تابع را در یک فایل dll داشته باشیم.)

نکته : ممکن است تعریف یک تابع به فایل‌های کتابخانه ای خاص خودش احتیاج داشته باشد که باید آنها را نیز در ابتدای برنامه داشته باشیم.

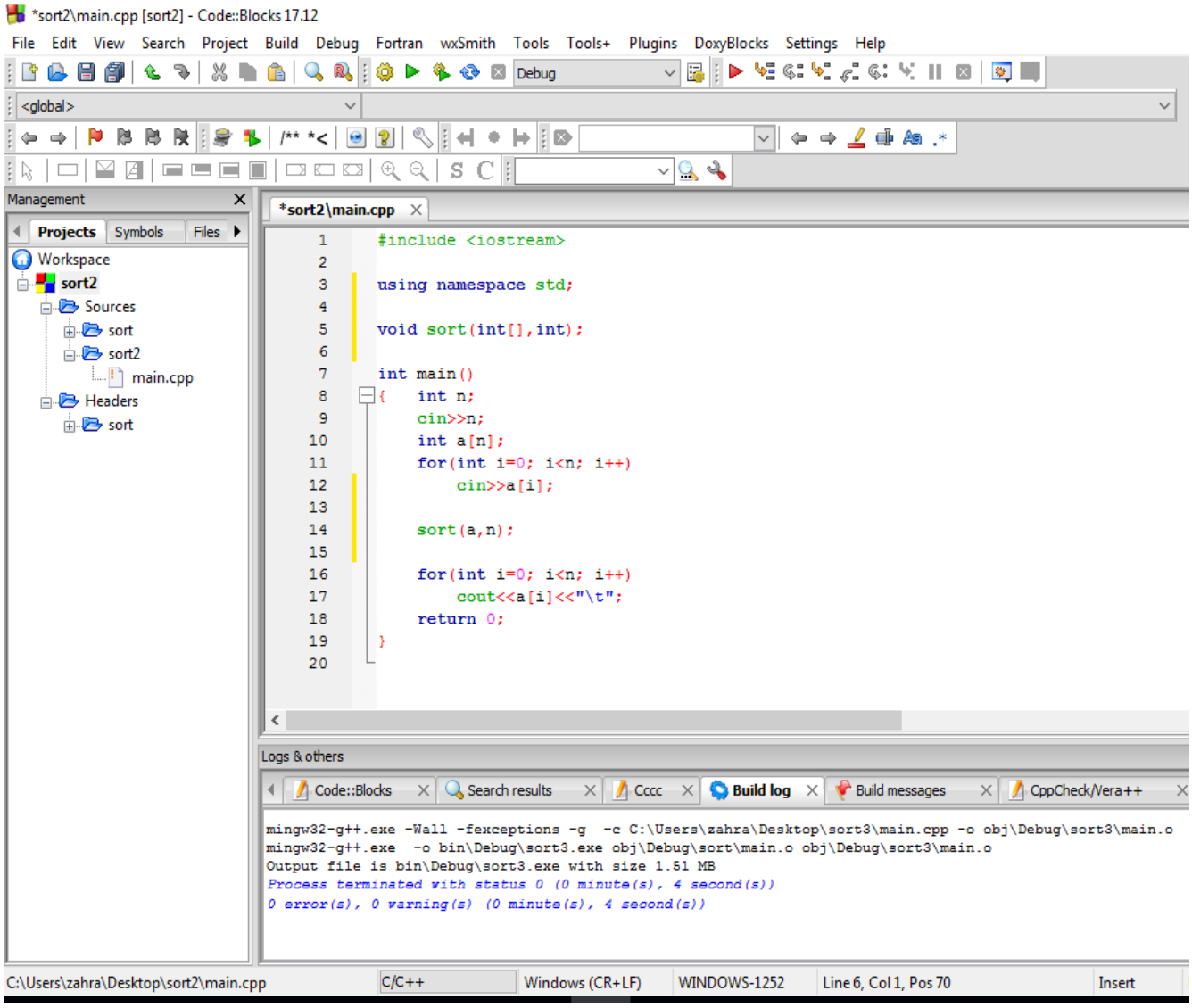
در نهایت تابع موردنظر را کامپایل (compile) کرده و در صورتی که با موفقیت کامپایل شود، به این ترتیب فایل dll ما آماده استفاده در هر برنامه دیگری می باشد.



۲–۵. نحوه استفاده از فایل DLL در برنامه های دیگر

ابتدا پروژه برنامه نویسی موردنظرمان را ایجاد می کنیم (مثلا با نام sort2) . سپس روی نام پروژه در پنل Management راست کلیک کرده و گزینه Add Files را انتخاب می کنیم. حالا باید آدرس فایل DLL که قبلا درست کرده‌ایم را بدهیم و دو فایل main.cpp و main.h را Open کنیم تا در زیرمجموعه پروژه اصلی ما قرار گیرند. حال در برنامه اصلی، تابع موردنظر را اعلان و فراخوانی کرده و سپس پروژه را اجرا (run) می‌کنیم.

نکته: ما هر تعداد فایل dll می توانیم به پروژه مان Add کنیم و هر تابع را در هر جایی که خواستیم (مثلا در برنامه اصلی یا در هر تابع دیگری (dll)) می توانیم فراخوانی کنیم.



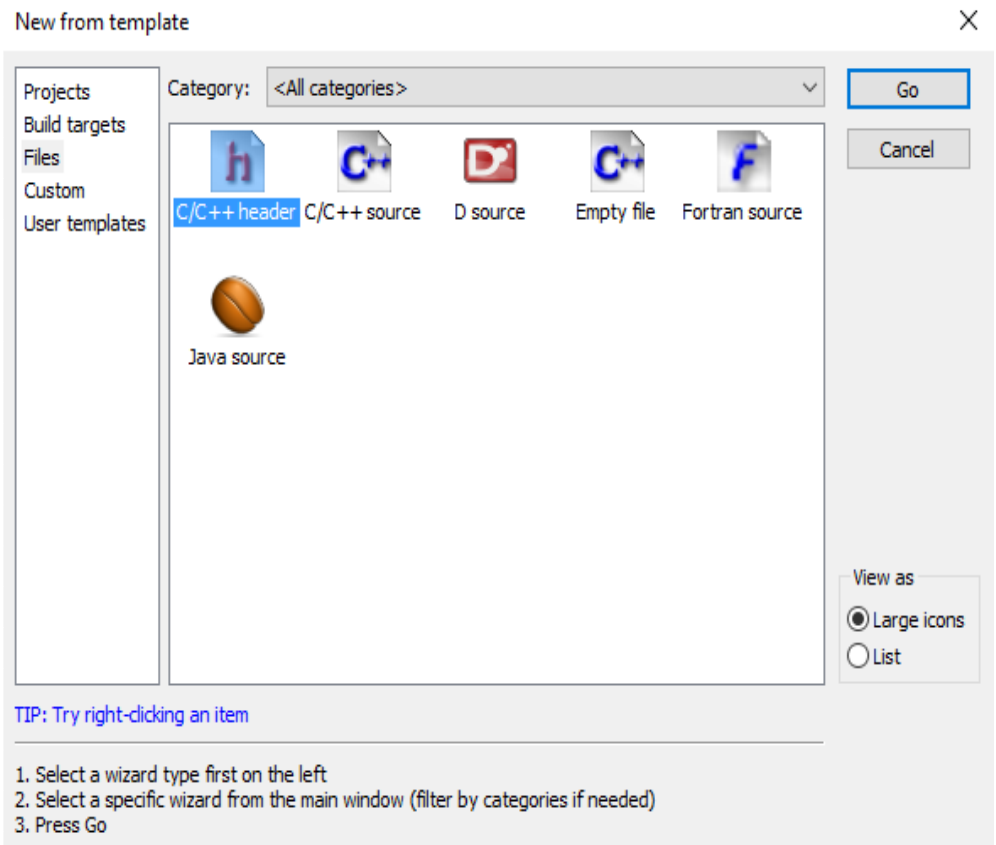
۶. ساختن فایل کتابخانه ای (فایل سرآیند[۱])

یک فایل کتابخانه ای، شامل توابعی است که در برنامه منبع بکار رفته اند و تعریف آن توابع در آن فایل کتابخانه‌ای یا فایل سرآیند آمده است. لذا وقتی در برنامه ای از توابع از پیش تعریف شده استفاده می کنیم باید ابتدای برنامه، کد مربوط ارجاع به کتابخانه را داشته باشیم. درغیر این صورت کامپایلر ارور می دهد.

در جدول زیر تعدادی از معروفترین کتابخانه ها را مشاهده می کنید:

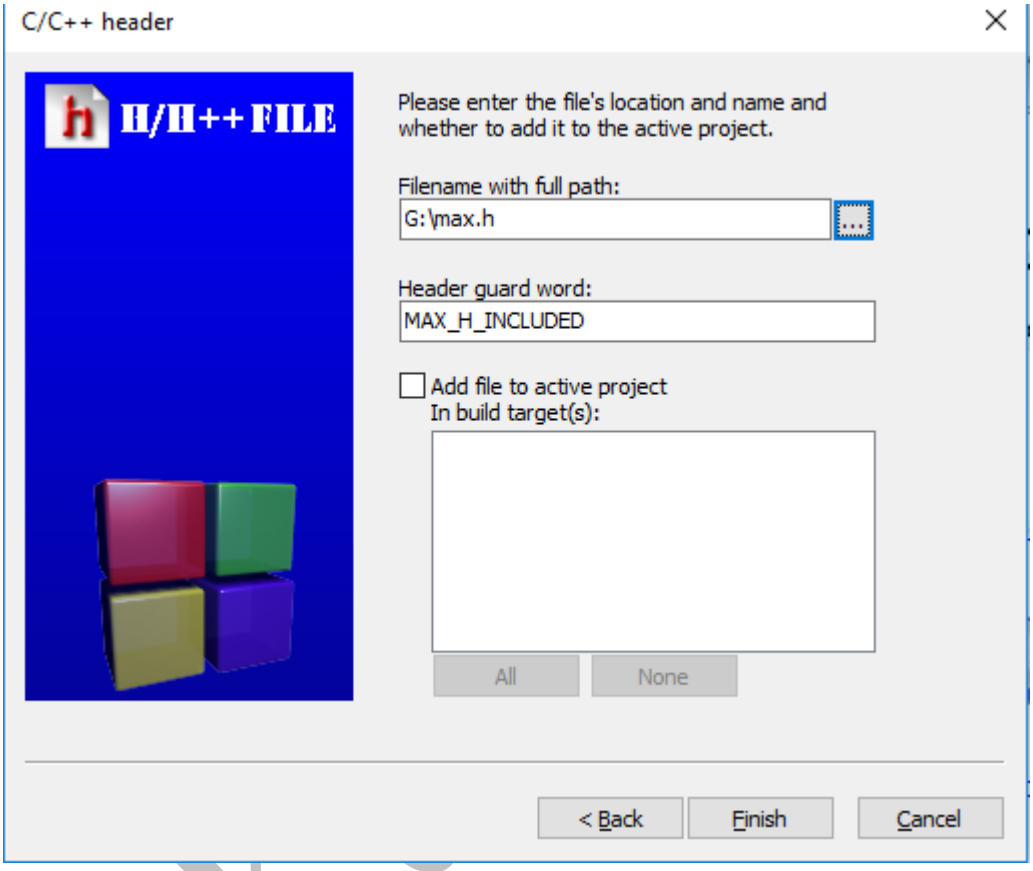
iostream stdio.h conio.h	تعریف توابع ورودی – خروجی
stdlib.h	تعریف توابع کاربردی مثل system(“”pause”)
iomanip	تعریف توابع فرمت بندی مثل setw();
cmath	تعریف توابع ریاضی

ما می توانیم به جز فایل های کتابخانه ای از پیش تعریف شده، خودمان نیز فایل کتابخانه‌ای بسازیم. برای این کار کافی است به هنگام ساختن پروژه جدید، از بخش Files بر روی گزینه C/C++ header کلیک کنیم. (مانند شکل رو به رو)



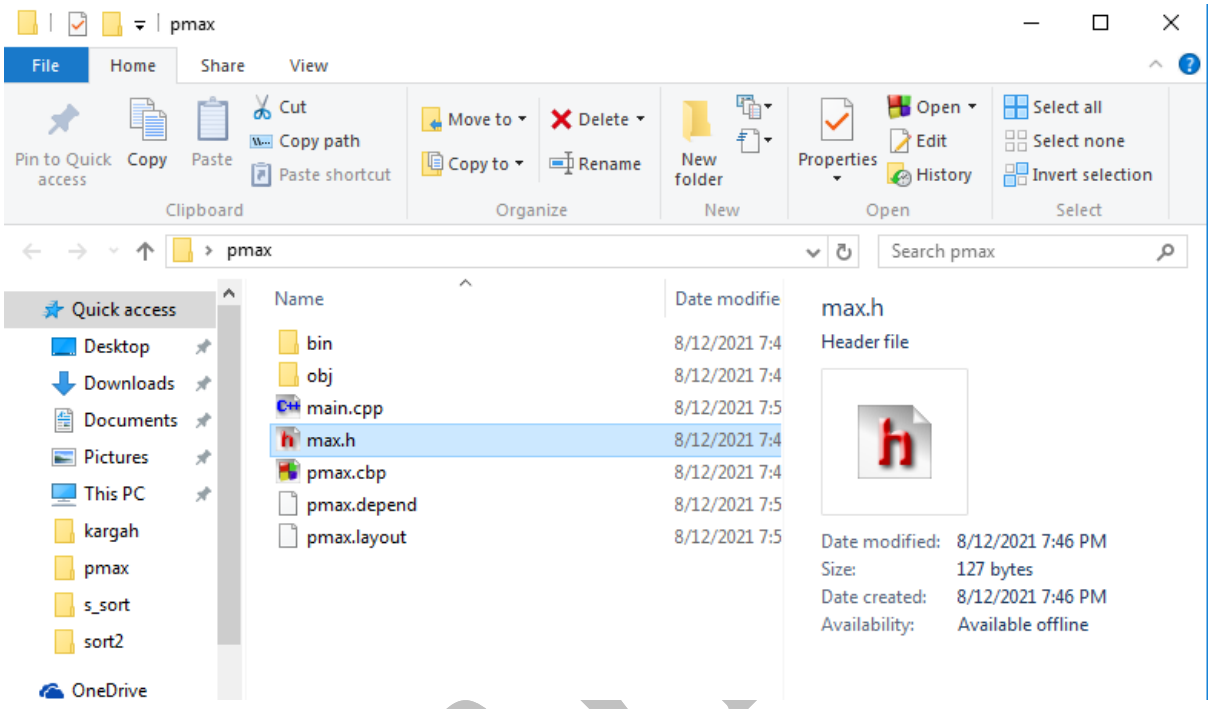
[۱]. Header file

سپس در پنجره باز شده در فیلد Filename with full path ، نامی برای هدرفایل انتخاب نموده و محل ذخیره سازی را نیز مشخص می‌کنیم.(مانند شکل زیر) و در نهایت بر روی Finish کلیک می‌کنیم.



بعد از ساختن فایل کتابخانه ای مورد نظر، برای استفاده از آن کافیست :

- هدر فایل مربوطه را (مثلا max.h) را به فولدر پروژه ضمیمه نماییم.



- در خطوط ابتدایی برنامه نیز هشتگ مربوط به فایل سرآیند را اضافه نماییم. (مثلا **#include "max.h"**) . حتما توجه داشته باشید که نام هدر فایل باید در گیومه قرار بگیرد. گذاشتن گیومه به این معناست که کامپایلر در مسیر خود پروژه به دنبال هدر فایل مورد نظر باشد.

main.cpp

```
1  #include <iostream>
2  #include "max.h"
3  using namespace std;
4
5  int main()
6  {    int a=5,b=7;
7       cout <<max(a,b);
8       return 0;
9  }
10
```

max.h

```
1  #ifndef MAX_H_INCLUDED
2  #define MAX_H_INCLUDED
3
4  int max(int a,int b)
5  {
6      return(a>b?a:b);
7  }
8
9  #endif // MAX_H_INCLUDED
10
```

هدر فایل یا فایل سرآیند

پروژه اصلی

نکته: پیرو مطلب ذکر شده در صفحه قبل، لازم است بدانید، الزامی به قرار دادن فایل کتابخانه‌ای در فولدر پروژه اصلی نیست. در این شرایط کافیست آدرس مسیر ذخیره‌سازی هدر فایل (فایل کتابخانه‌ای) را به صورت کامل در گیومه قرار دهیم.

مانند تصویر زیر:

main.cpp

```
1   #include <iostream>
2   #include "D:\\max.h"
3   using namespace std;
4
5   int main()
6   {
7       int a,b;
8       cin>>a>>b;
9       cout<<max(a,b);
10      return 0;
11  }
12
```

۷. تفاوت‌های فایل کتابخانه ای و فایل DLL

مکانیزم اجرا و سرعت آن‌ها:

از آنجایی که فایل کتابخانه‌ای به صورت کامل و یکجا به برنامه الحاق می‌شود در نتیجه در روند اجرای برنامه نیازی به مراجعه مداوم به هدر فایل و بازگشت به برنامه اصلی نیست لذا سرعت اجرای برنامه نیز افزایش می‌یابد اما سربار برنامه زیاد می‌شود.

مکانیزم اجرا در فایل DLL به اینگونه است که، هر بار که در برنامه فراخوانی‌ای صورت گیرد، به فایل DLL مراجعه می‌کند و سپس به برنامه برمی‌گردد و این عمل با توجه به برنامه، ممکن است بارها صورت گیرد در نتیجه این مراجعه به فایل DLL و بازگشت‌ها به برنامه اصلی، سرعت اجرا کاهش می‌یابد.

۸. تمرینات برنامه نویسی

۱- برنامه ای بنویسید که با استفاده از کاراکترهای فرمت‌بندی عبارت "Hello world" را چاپ نماید.

۲- برنامه ای بنویسید که مجموع اعداد زوج از ۱ تا n را محاسبه نماید.

۳- برنامه ای بنویسید که n عنصر از ورودی دریافت نموده سپس تعداد اعداد زوج و فرد را شمارش نماید.

۴- برنامه ای بنویسید که یک عدد از ورودی گرفته کامل بودن آن را بررسی نماید. (عدد کامل عددی است که مجموع مقسوم علیه های آن به جز خودش، با خود عدد برابر است. مثل ۶= ۲+۳+۱)

۵- برنامه ای بنویسید که جدول ضرب اعداد ۱ تا ۱۰ را چاپ نماید.

۶- برنامه ای بنویسید که یک رقم بین ۰ تا ۹ از ورودی دریافت نموده سپس معادل حرفی آن را چاپ نماید. (با استفاده از switch)

۷- برنامه ای بنویسید که اعداد رو به رو را چاپ نماید.

۱

۲ ۲

۳ ۳ ۳

۴ ۴ ۴ ۴

۸- برنامه ای بنویسید که دو عدد X , y را از ورودی خوانده سپس بزرگترین مقسوم علیه بین آنها را محاسبه و چاپ نماید. (محاسبه ب.م.م)

۹- برنامه ای بنویسید که عددی را از وروی دریافت کرده سپس قدرمطلق آن را محاسبه و چاپ نماید.

۱۰- برنامه ای بنویسید که عددی بین ۱ تا ۷ را از ورودی دریافت کرده و روز معادل آن را چاپ نماید. (با استفاده از دستور switch-case)

۱۱- برنامه ای بنویسید که n عدد از ورودی دریافت کرده سپس حاصل ضرب اعداد مثبت آن را محاسبه کند.

۱۲- برنامه ای بنویسید که عددی را از ورودی دریافت کرده سپس فاکتوریل آن را محاسبه نماید. (فاکتوریل یک عدد برابر است با حاصلضرب متوالی عدد یک تا خود آن عدد)

۱۳- برنامه ای بنویسید که یک عدد از ورودی دریافت کرده سپس میانگین اعداد از ۱ تا آن عدد را محاسبه کند.

۱۴ - برنامه ای بنویسید که عددی را از ورودی دریافت کرده سپس آن را به مبنای ۲ ببرد.

۱۵ - برنامه ای بنویسید که عددی را از ورودی دریافت کرده سپس مشخص کند آیا عدد اول است یا خیر و آن را با پیغام مناسبی نمایش دهد.(عدد اول عددی است که جز خودش و ۱ مقسوم علیه دیگری ندارد مثل ۳، ۷، ۱۱ و ...)

۱۶ - برنامه ای بنویسید که n جمله اول سری فیبوناچی را نمایش دهد. (در سری فیبوناچی جمله اول و دوم برابر ۱ است و بقیه جمله ها از جمع دو جمله قبلی آن بدست می آید.

۱ ۱ ۲ ۳ ۵ ۸ ۱۳ ...)

۱۷ - برنامه ای بنویسید که فاکتوریل عدد صحیح n را محاسبه نماید. (با استفاده از تابع)

۱۸ - برنامه ای بنویسید که مجموع اعداد طبیعی از ۱ تا n را محاسبه نماید.(با استفاده از تابع)

۱۹- برنامه ای بنویسید که عنصر x را در آرایه ای به طول n جستجو نماید.

۲۰- برنامه ای بنویسید که اسامی n دانشجو را از ورودی دریافت نموده سپس آنها را مرتب نماید.

راه های ارتباطی:

Telegram channel: @kamalpnu

Telegram ID: @zahrakamal

E_mail: zahra.kamal@yahoo.com