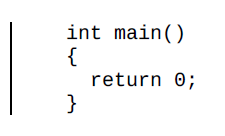
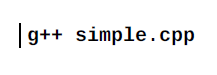
**به نام خالق هستی بخش**

**نوشتن اولین کد به زبان C++**

در زبان C++ ابتدا کدهای تابع main() اجرا می شود، تابع خودش یک مجموعه از دستورات است که یک هدف مشخصی را دنبال می کنند، در نتیجه یک برنامه در زبان C++ می بایست حداقل شامل یک تابع که main() نام دارد باشد. قطعه کد زیر ساده ترین کد ممکن به زبان C++ بوده که می توان آن را با موفقیت کامپایل کرد و اجرا نمود:

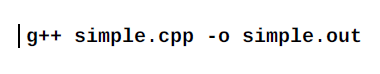


فرض کنید کد بالا در یک فایل با نام simple.cpp ذخیره شده است. ما می توانیم این کد را با استفاده از دستور کنسولی زیر در همان پوشه ای که کد در آن قرار دارد و با استفاده از کامپایلر g++ ترجمه نماییم:



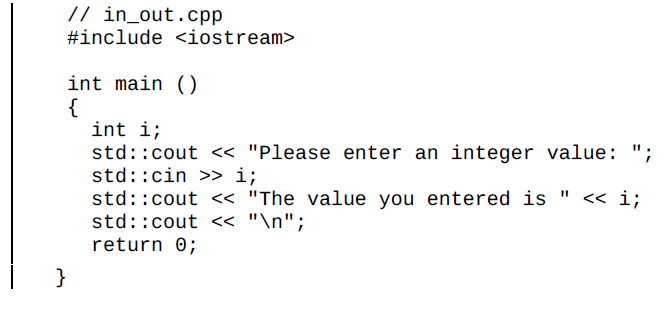
اگر هیچگونه خطایی ظاهر نشد، فایل خروجی به شکل خودکار تولید خواهد شد. چنانچه با خط فرمان ویندوز کد را کامپایل کنیم خروجی نامی شبیه به a.exe( با پسوندexe) خواهد داشت ولی اگر با استفاده از ترمینال لینوکس یا مک فایل را کامپایل کنید خروجی نامی شبیه به a.out خواهد داشت(پسوندout).

ما با استفاده از سوییچ –o به همراه نام مورد نظر، می توانیم دقیقا مشخص کنیم که نام خروجی چه باشد، مثلا دستور زیر خروجی با نام simple.out تولید خواهد کرد:



در واقع وقتی که ما فایل خروجی را اجرا می کنیم(با نوشتن a(نام فایل خروجی) و سپس فشردن کلید اینتر داخل ویندوز و یا نوشتن ./a.out و فشرن اینتر در Shell لینوکس)، هیچ چیزی را در صفحه کنسول(صفحه خروجی) مشاهده نمی کنیم و دلیل آن این است که برنامه ما هنوز دستوری برای چاپ روی صفحه ندارد. برای اینکه به قطعه کد خود معنا ببخشیم بیایید کد قبلی را به گونه ای باز نویسی کنیم که یک مقدار را از کاربر گرفته و سپس همان را دوباره به او نشان دهد.

کد بازنویسی شده باید چیزی شبیه به کد زیر باشد:



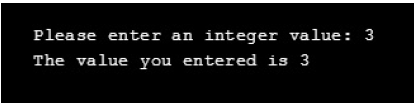
همانطور که در کد بالا می بینیم، ما چند خط کدرا به گونه ای به برنامه خود اضافه کرده ایم تا برنامه ما بتواند مقداری را در خروجی چاپ کند و همچنین کاربر نیز بتواند یک مقدار به برنامه ما وارد کند.

ابتدا بر نامه متنی را به کاربر نشان داده و از او می خواهد یک عدد صحیح وارد کند. بعد از آن کاربر مقدار مورد نظر خود را وارد کرده و اینتر را فشار می دهد وبرنامه آن را نمایش می دهد. ما همچنین( در کد) ما یک متغیر با نام i از نوع int تعریف کرده ایم. این متغیر در جهت نگهداری یک داده به شکل عدد صحیح استفاده می شود( در قسمت بعدی درباره متغیر ها و انواع داده صحبت خواهیم کرد).

فرض کنید کد را با نام in\_out.cpp ذخیره کنیم، می توانید آن را با دستور زیر کامپایل کنیم:



حال اگر بر نامه را اجرا کنیم، خروجی زیر را در صفح کنسول خواهیم دید(برای مثال من 3 را به عنوان ورودی به برنامه داده ام):



تا حالا می دانیم که این برنامه متنی را در کنسول چاپ می کند، ما(برا اینکار از) دستور std :: cout استفاده کرده و باید ورودی هایی را به برنامه بدهیم (که از) دستور std :: cin استفاده می کنیم. همچنین در فایل in\_out.cpp درقسمت ابتدایی کد دستور #include <iostream> را مشاهده می کنیم، این دستور بدان جهت استفاده می شود تا به کامپایلر اطلاع دهد std :: cout و std :: cin در کجا قرار دارند(این دستورات از قبل نوشته شده و ما با دستور #include صرفا محل این دستورات را به کامپایلر اعلام می کنیم)، پیاده سازی این دستورات در سرفایل iostream قرار دارد.

در بالاترین قسمت کد می توانیم ملاحظه کنیم که خط با دو کاراکترslash آغاز شده(//)، این بدان معناست که این خط نباید مانند سایر قسمت های کد در نظر گرفته شود و کامپایلر باید آن را نادیده بگیرد. که این موضوع برای گذاشتن یادداشت ها و برنامه ریزی ها مفید است بدین شکل سایر برنامه نویسان می توانند متوجه کد های ما بشوند!

**بهبود تجربه کد نویسی با استفاده از محیط های توسعه مجتمع(IDE)**

تاکنون یادگرفتیم چگونه کدی را به زبان C++ نوشته، آن را کامپایل کرده و اجرا کنیم! با این وجود، این که با استفاده از دستورات کنسولی کد را کامپایل کرده و سپس اجرا کنیم خسته کننده به نظر می رسد. پس برای تسریع در روند توسعه برنامه خود از یک محیط توسعه مجتمع(IDE) استفاده خواهیم کرد، تا بتوانیم صرفا با یک کلیک برنامه را کامپایل و اجرا کنیم. می توانید به دلخواه خود یکی از IDE های رایگان یا غیر رایگان موجود در بازار را تهیه کنید. اگر چه من شخصا Code::Blocks IDE را انتخاب می کنم زیرا رایگان، متن باز و چند سکویی است به این معنا که برای محیط های مختلف نرم افزاری مانند لینوکس و ویندوز و... در دسترس می باشد!

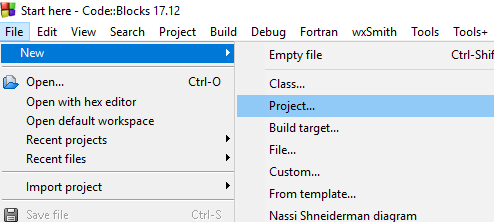
شما می توانید به اطلاعاتی نظیر اینکه چگونه این برنامه را دانلود، نصب و استفاده کنید در وب سایت زیر دسترسی داشته باشید:

<http://www.codeblocks.org>

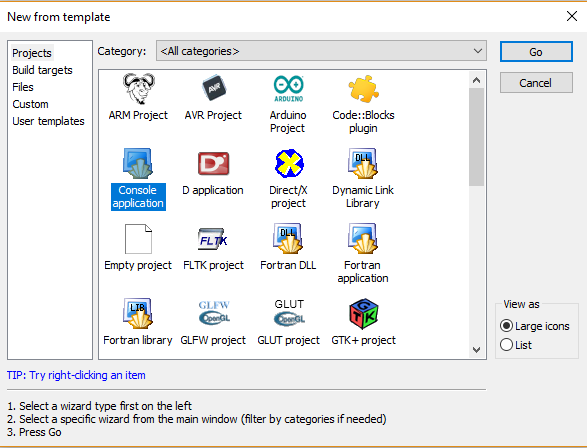
در حیقیت ما می توانیم فرایند کامپایل برنامه را با مجموعه ای از ابزار ها مانند make و cmake به شکل خودکار در آوریم. اگر چه این موضوع به توضیحات بیشتری نیاز دارد ولی از آنجا که این کتاب پیرامون مبحث ساختمان داده ها و الگوریتم نوشته شده صحبت درباره این ابزار ها فقط تعداد صفحات کتاب را افرایش خواهد داد پس اینجا در این باره بحث نمی کنیم.

در این موضع، استفاده از IDE بهترین راه حل جهت خودکار سازی فرآیند کامپایل می باشد. به این دلیل که IDE به شکل مطلوبی از ابزار ها(جهت خودکار سازی) استفاده می کند.

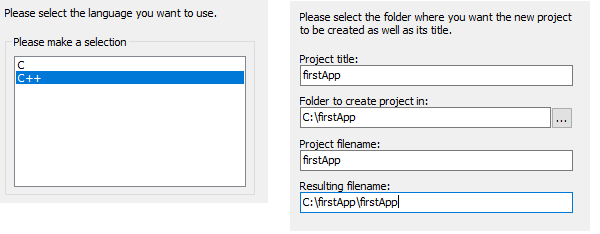
بعد از اینکه Code::Blocks را نصب کردیم، می توانیم با کلیک بر روی منوی File وسپس انتخا گزینه New و در نهایت انتخاب Project یک سند جدید را ایجاد نماییم.



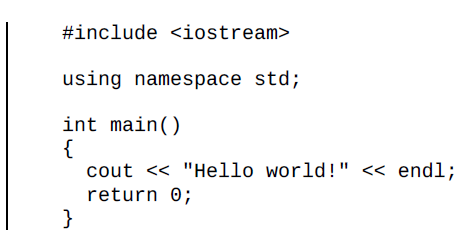
بعد از طی کردن این مراحل یک پنجره جدید یک پنجره جدید ظاهر شده که در آن می توانیم نوع سند مد نظره خود را انتخاب کنیم، برای بیشتر مثال های این کتاب ما از نوع Console Application استفاده می کنیم. حال بر روی دکمه Go کلیک کنید تا به ادامه کار بپردازیم.



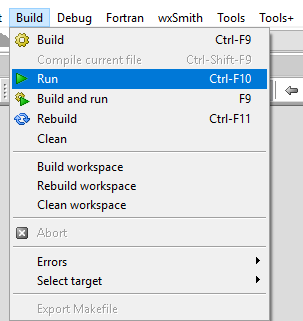
در پنجره پیش رو می توانیم مشخص کنیم که می خواهیم از کدام زبان استفاده کنیم، که خوب ما قصد داریم C++ را به کار بریم! و سپس نام پروژه را تعیین کرده و مکان پروژه روی هاردیسک را نیز مشخص نمایید( من پروژه را firstApp نامیده ام):



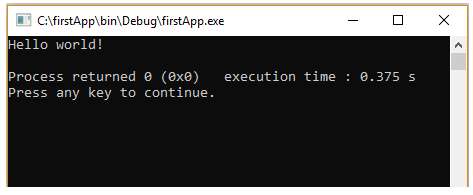
بعد از اتمام این مراحل ما صاحب یک پروژه به همراه فایلی با نام main.cpp می باشیم که این فایل شامل محتویات زیر است:



حال می توانیم صرفا با کلیک بر روی Build و سپس انتخاب گزینه Run این کد را کامپایل و اجرا کنیم



خروجی حاصل به شکل زیر می باشد:



در کد نوشته شده داخل main.cpp بعد از #include <iostream> دستور using namespace std را ملاحظه می کنیم که این موضوع به کامپایلر اطلاع می دهد که کد ما از فضای نامی(namespace)با عنوان std استفاده می کند پس دیگر نیاز نیست به مانند گذشته در هر جا که دستورات cin یا cout وجود داشت ابتدا std:: را بنویسیم. پس کد باید نسبت به قبل ساده تر شده باشد.

تعریف متغیر ها با استفاده از انواع داده اصلی

در مثال قبلی ما با متغیر(مکانی که برای ذخیره یک داده استفاده می شود) سر و کله زدیم تا بتوانیم با استفاده از عملگر ها داده ها را دستکاری کنیم. در زبان سی ما ملزم به تعیین نوع یک متغیر می باشیم در نتیجه هر متغیر فقط می تواند انواع خاصی از داده ها را نگه داری کند. در زیر انواع متغیر های اصلی(نوع داده اصلی) آمده اند. در مثال قبلی بعضی از این انواع را بکار بردیم:

1. **Boolean**(bool)

جهت نگه داری دو پاسخ ممکن در شرط ها( بله یا خیر) استفاده می شود.

1. **Character**(char, wchar\_t, char16\_t **and** char32\_t)

جهت نگه داری یک کاراکتر اسکی(ASCII) استفاده می شود.

1. **Floating Point**(float , double **and** long double)

جهت نگه داری اعداد اعشاری استفاده می شوند.

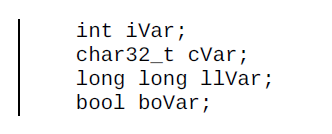
1. **Integer**(short, int , long **and** long long)

جهت نگه داری اعداد صحیح استفاده می شوند.

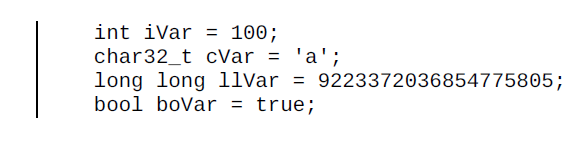
1. **No data**(void)

اصولا در جایی استفاده می شود که می خواهید یک مقدار به نوع خاصی اشاره نکند.

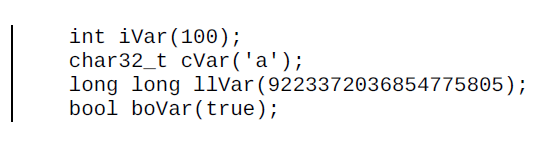
دو راه برای ساخت یک متغیر وجود دارد یکی تعریف آن و دیگری تعریف با مقدار دهی اولیه، تعریف یک متغیر منجر به شکل گیری آن شده بدون اینکه برای متغیر مقداری در نظر گرفته باشد. تعریف با مقدار دهی اولیه متغیر را ساخته و یک مقداری را ابتداً به آن نسبت می دهد. قطعه کد زیر چگونگی تعریف متغیر را بیان می دارد:



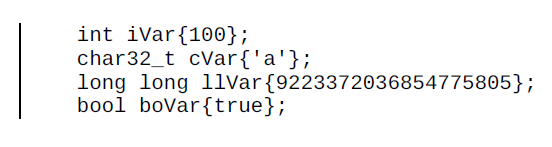
و کد زیر نیز چگونگی تعریف متغیر همراه با مقدار دهی اولیه را روشن می سازد:



کد بالا مقدار دهی اولیه متغیر را به روش Copy Initialization را نشان می دهد، در این روش یک مقدار را با استفاده از علامت مساوی(=) به متغیر نسبت می دهیم. روش دیگر برای مقدار دهی اولیه direct initialization می باشد، در این روش یک مقدار را با استفاده از پرانتز درمتغیر خود قرار می دهیم. کد زیر از این روش برای مقدار دهی اولیه متغیر استفاده کرده است:



در کنار دو روش گفته شده می توان از روشی مشابه با استفاده از آکولاد ها بهره برد. قطعه کد زیر این روش را که brace-initialization نامیده می شود را نشان می دهد:



ما هیچ گاه نمی توانیم یک متغیر را با نوع void تعریف کنیم مثلا void vVar; و دلیل آن نیز این می باشد که هنگامی که یک متغیر را تعریف می کنیم ملزم به تصمیم گیری درباره نوع داده ای که متغیر نگه داری می کند، می باشیم حال اگر متغیری را از نوع void تعریف کنیم این بدان معناست هیچ قصدی برای نگه داری چیزی در آن نداریم.

کنترل روند اجرای برنامه

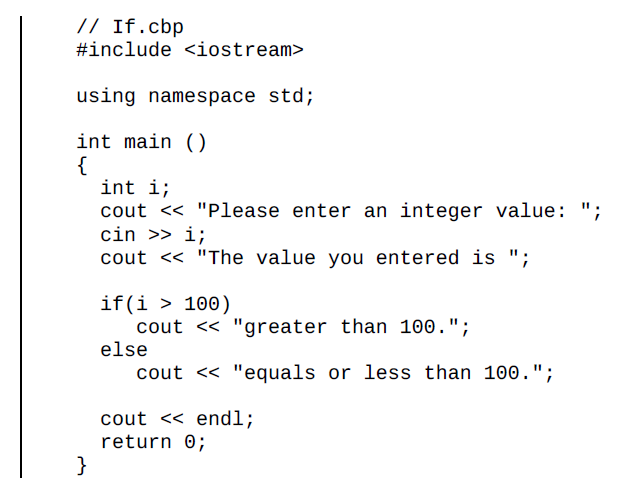
همانطور که قبلا گفتیم، برنامه های نوشته شده در زبان سی از تابع main() شروع به اجرای دستورات یکی پس از دیگری می کنند تا در نهایت به پایان این تابع برسند. این در حالی است که می توانیم این مسیر اجرا را با استفاده از دستورات کنترلی عوض کنیم.

دستورات بسیاری برای کنترل روند اجرای برنامه وجود دارند ولی ما قصد داریم تنها برخی از آنها را مورد برسی قرار دهیم که اغلب در نوشتن و طراحی الگوریتم ها مورد استفاده قرار می گیرند.

دستورات شرطی

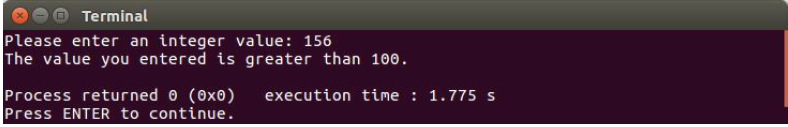
یکی از مواردی که می تواند مسیر اجرای کد ها را عوض کند دستورات شرطی هستند. با استفاده از این دستورات صرفا خطوطی از کد های نوشته شده اجرا می شوند که شرط مورد نظر ما را برآورده کنند. با استفاده از کلمات کلیدی if و else می توان چنین دستوراتی را پیاده سازی کرد.

بیایید کدی که قبلا در فایل in\_out.cpp نوشتیم را به گونه ای تغییر دهیم که دستورات شرطی در آن ها استفاده شده باشد. برنامه به گونه ای تغییر می کند که مشخص کند عدد ورودی از 100 بزرگتر است یا نه، کد مورد نظر باید چیزی شبیه به( شکل) پایین باشد:



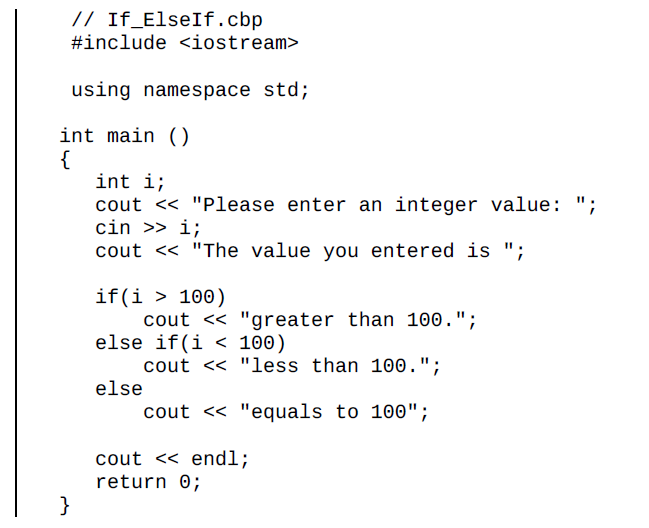
همانطور که مشاهده می کنیم، ما یک جفت از کلمات کلیدی if و else داریم که می توانند مشخص کنند عدد وارد شده از 100 بزرگتر است یا خیر. با اجرای کد فوق فقط یکی از خطوط داخل دستور شرطی می تواند اجرا شود: یا خط پایین if و یا خط پایین else.

اگر کد را کامپایل کرده و اجرا نماییم، خروجی مشابه زیر را خواهیم دید:

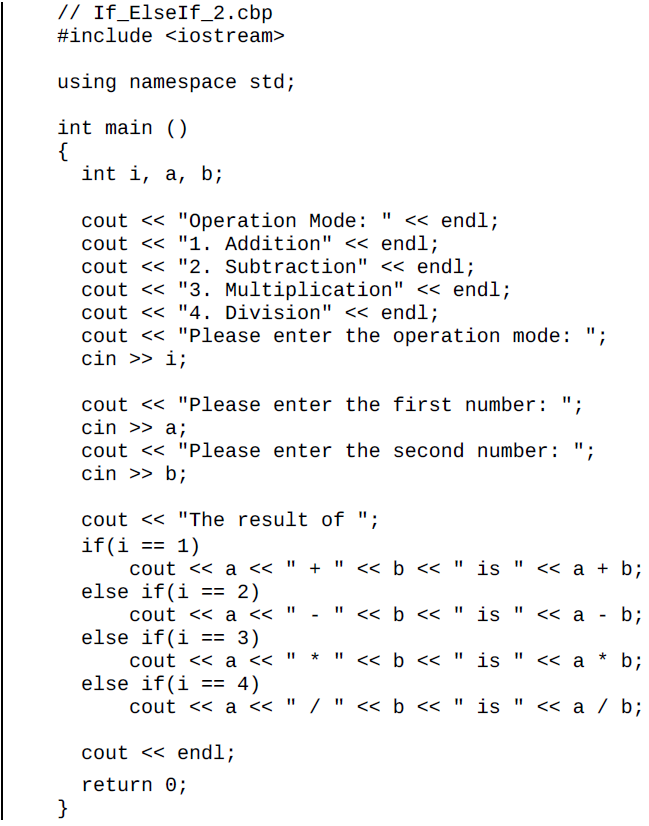


در پنجره خروجی بالا می توان ملاحظه کرد که خط cout << “equals or less than 100” اجرا نشده است و این بدان دلیل است که عدد ورودی ما از 100 بیشتر است.(عدد 165 را وارد کرده ایم).

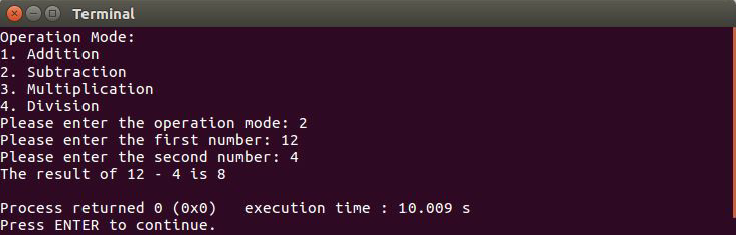
علاوه بر این در دستور شرطی if…else می توان بیش از دو شرط را (برای ورودی) متصور شد، می توان به صورت زیر کد را به گونه ای باز نویسی کنیم که بیش از دو دستور شرطی داشته باشد:



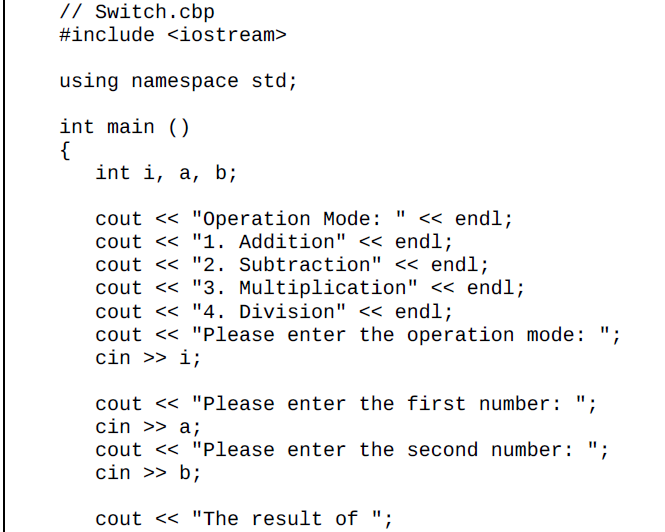
یکی دیگر از دستورات شرطی دستور switch می باشد. قبل از اینکه درباره این دستور شرطی صحبت کنیم، اجازه دهید یک ماشین حساب ساده بنویسیم که بتواند دو عدد را با هم جمع نماید این برنامه همچنین باید بتواند اعمال تفریق، ضرب و تقسیم را نیز انجام بدهد. ابتدا از دستور if…else استفاده خواهیم کرد. کد مطلوب به شکل زیر خواهد بود:

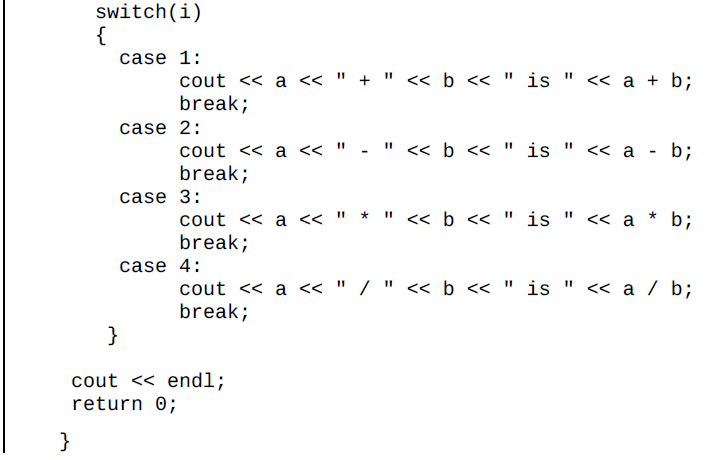


همانور که در کد بالا دیدیم، چهار عمل ممکن برای انتخاب وجود دارد( که ما به دلخواه یکی ازآنان را بر می گزینیم). برای نوشتن این برنامه از دستور if…else استفاده کرده ایم. خروجی برنامه بالا باید به شکل زیر باشد:



با این حال می توانیم از دستور switch نیز به همین شکل استفاده کنیم، کد بعد از بازنویسی بدین شکل خواهد بود:



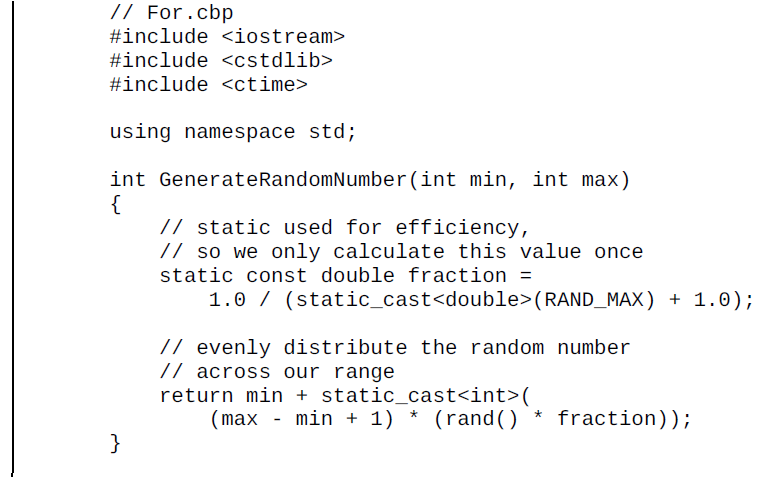


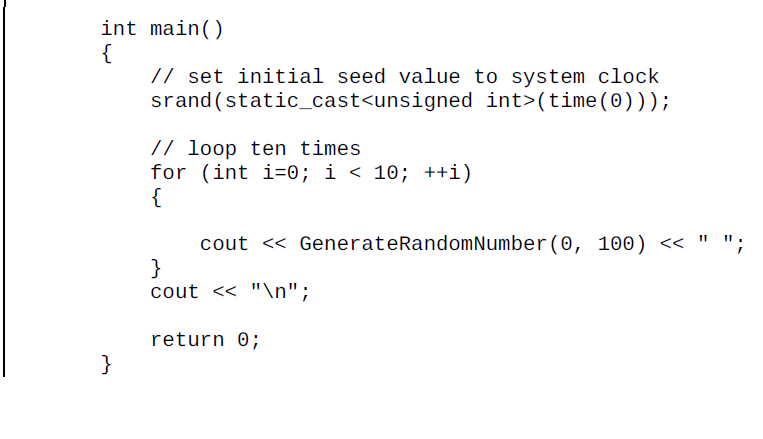
و چنانچه این کد ها را اجرا کنیم. همان نتیجه قبلی را مشاهده خواهیم کرد.

دستورات تکرار

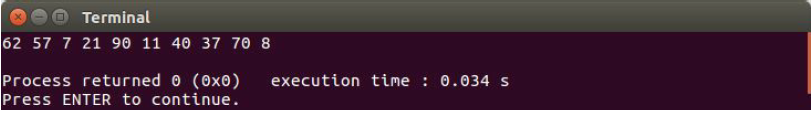
در زبان C++ چندین ساختار برای تکرار وجود دارد(حلقه ها) که از جمله آن ها می توان به for، while، do…while اشاره کرد. حلقه for غالبا زمانی استفاده می شود که می دانیم به چه تعداد می خواهیم دستورات را تکرار کنیم درحالی که حلقه های while و do…while تا هنگامی که شرط مطلوب ما حاصل نشود به تکرار دستورات ادامه می دهد.

در نظر بگیرید که می خواهیم 10 عدد تصادفی بین صفر تا صد تولید کنیم، از آنجا که ما می دانیم چند عدد می خواهیم استفاده از حلقه for بهترین راه حل می باشد. برای این منظور می توانیم کد زیر را بنویسیم:



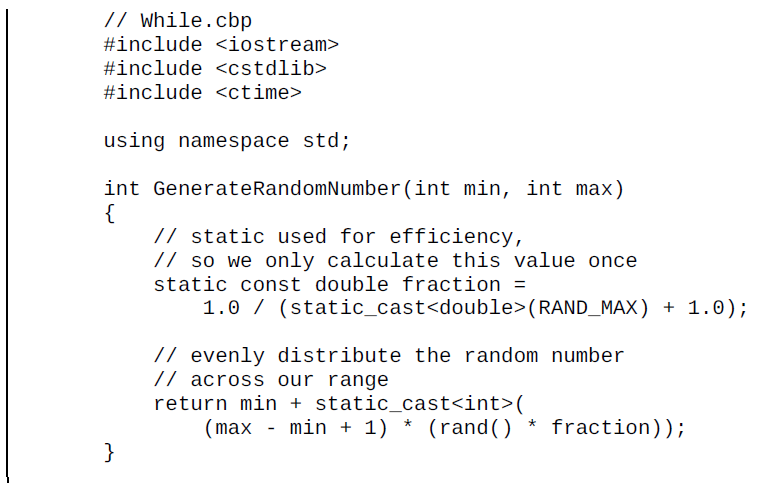


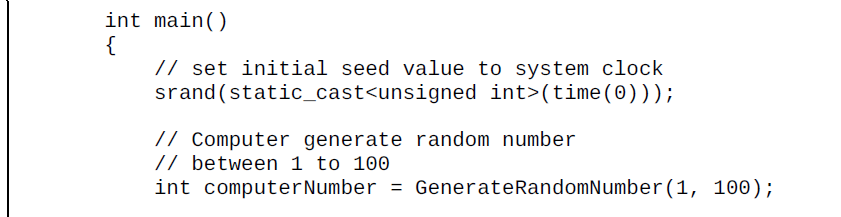
در کد بالا ما یک تابع دیگر با نام GenerateRandomNumber() در کنار تابع main() ایجاد کرده ایم. این کد با استفاده از حلقه for ده بار این تابع را فراخوانی می کند. همانطور که در کد گفته شده هم می بینید خروجی باید چیزی شیبه به شکل پایین باشد:

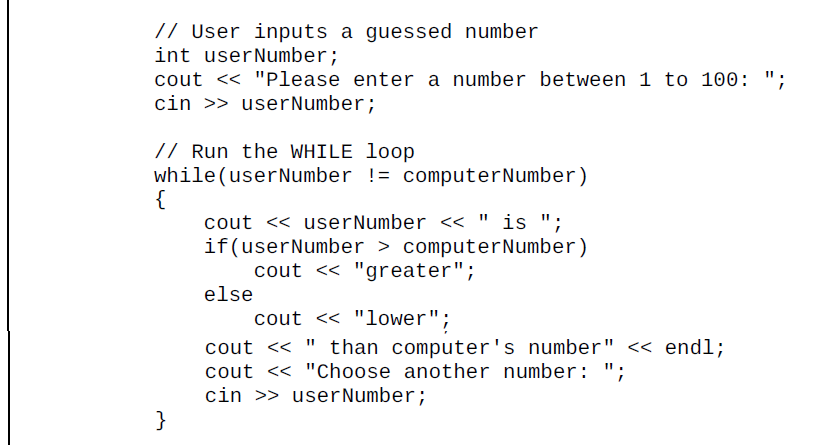


و اما به دیگر حقله هایی که قبلا اشاره کردیم بپردازیم، یعنی حلقه های while و do…while این دو حلقه رفتار هایی مشابه یکدیگر دارند. و اما تفاوت آن ها در این است که هنگام استفاده از حلقه while این امکان وجود دارد که دستورات درون حلقه به هیچ عنوان اجرا نشود. در حالی که در حلقه do...while دستورات داخلی حداقل باید یک بار اجرا بشوند.

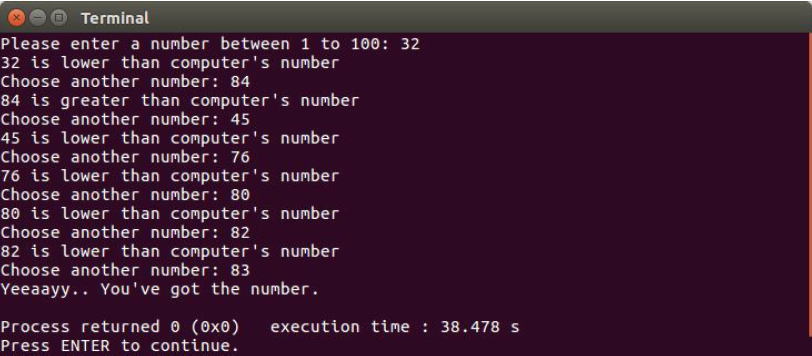
اکنون بیاید یک بازی ساده با استفاده از این حلقه ها بسازیم، کامپیوتر یک عدد تصادفی بین یک تا صد ساخته و کاربر می بایست حدس بزند که کامپیوتر چه عددی را تولید کرده است. هنگامی که کاربر عدد صحیح را وارد کرد کامپیوتر او را آگاه خواهد ساخت. همچنین(در هر مرحله) این برنامه به کاربر خواهد گفت که عدد مد نظر او از عدد ساخته شده توسط کامپیوتر کوچکتر یا بزرگتر است. هنگامی که عددی که توسط کاربر وارد می شود دقیقا برابر با عدد ساخته شده توسط کامپیوتر بشود برنامه پایان می یابد. کد مورد نظر باید مانند شکل زیر باشد:



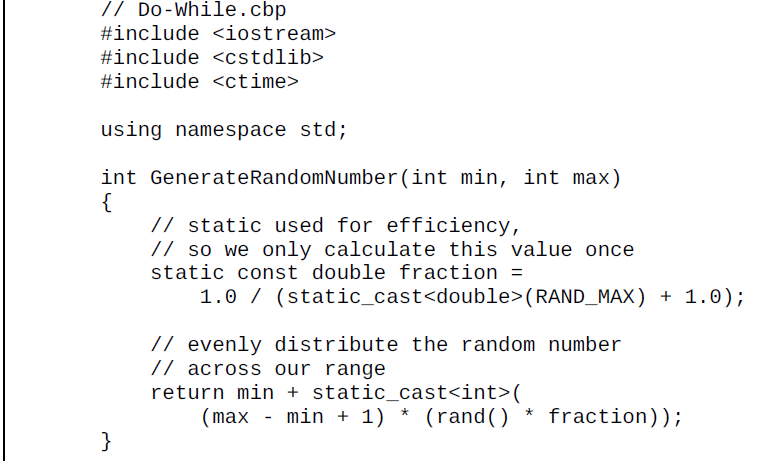


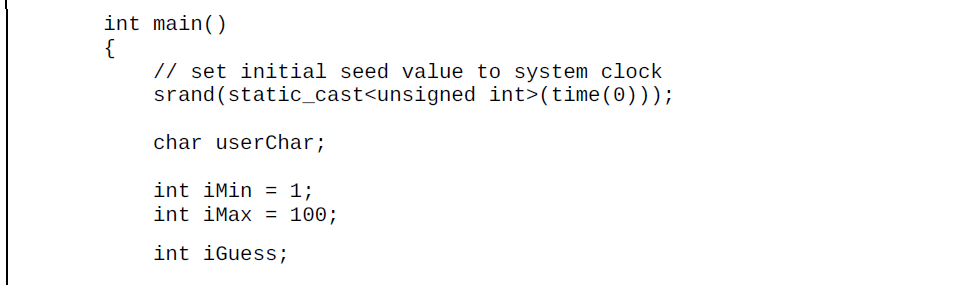


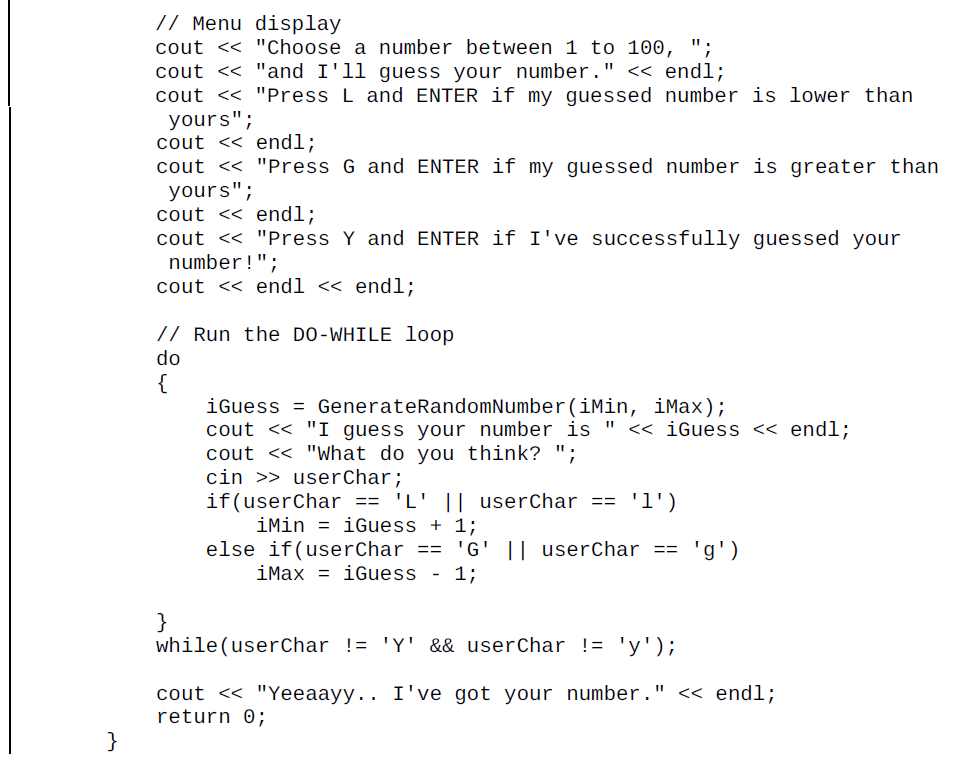
از کد بالا می توان ملاحظه کرد که دو متغیر با نام های computerNumber و userNumber داشته و آن ها را با یک دیگر مقایسه خواهیم کرد، تا آن هنگام که این دو با هم برابر شوند. که در صورت وقوع این اتفاق دستورات درون حلقه while به هیچ وجه اجرا نمی شوند. چگونگی روند اجرای این برنامه را در شکل پایین مشاهده می فرمایید:



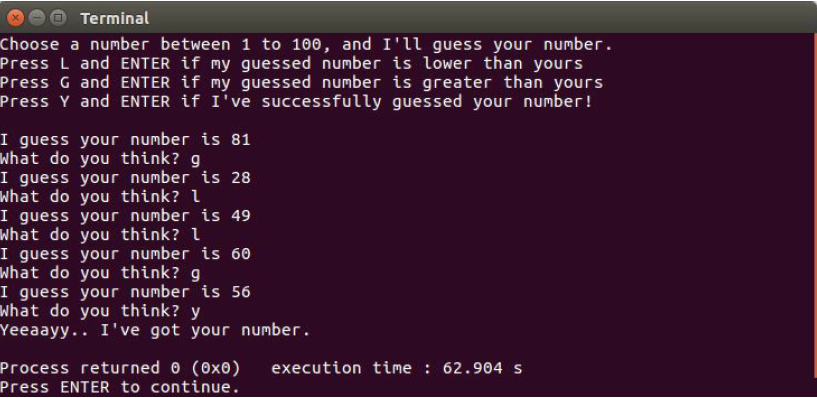
ما به شکل موفقیت آمیزی از حلقه while در برنامه بالا استفاده کردیم. با این وجود حلقه ی do…while نیز همانگونه که قبلا نیز اشاره گردید مشابه با حلقه while می باشد. اگر چه نمی توان صرفا با تعویض دستور do…while و while برنامه را بازنویسی کرد اما می توان پیاده سازی را برای این بازی در نظر گرفت که از حلقه do…while استفاده می کند. در این پیاده سازی کاربر یک عدد را در نظر می گیرد و این کامپیوتر است که باید حدس بزند این عدد چیست. کد این برنامه به شکل زیر است:







همانطور که در برنامه بالا مشاهده می کنید کامپیوتر حداقل یک بار عدد مورد نظر کاربر را حدس می زند. و چنانچه خوش شانس باشد، عدد گفته شده دقیقا مطابق با عدد مدنظر کاربر است. بنابراین در اینجا از حلقه do…while استفاده کرده ایم. وقتی برنامه را ساخته و اجرا کنیم خروجی مشابه زیر را خواهیم داشت:

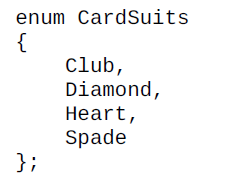


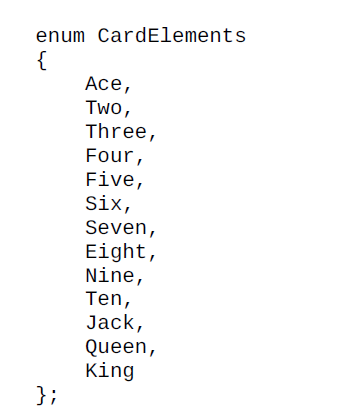
در خروجی بالا، من عدد 56 را انتخاب کردم، برنامه ابتدا حدس زد عدد مزبور 81 باشد. از آنجا که این عدد با عدد مورد نظر من تفاوت داشته و از آن بزرگتر بود، برنامه عدد دیگری را حدس زد که آن عدد 28 بود. و بعد از آن حدس های دیگری طبق اشارات من زد تا در نهایت عدد صحیح را پیدا نمود. این برنامه همانطور که می بینیم به محض اینکه کاربر دکمه y را فشار دهد از حلقه do…while خارج می گردد.

افزودن توانایی متغیر ها با استفاده از انواع داده پیشرفته

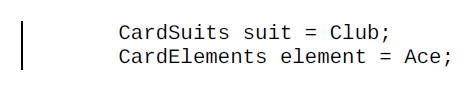
در بخش های گذشته درباره انواع داده اصلی صحبت کردیم. انواع اصلی در تعریف و مقدار دهی متغیر هایی استفاده می شود که از نوع داده ذخیره شده در آن اطمینان داشته باشیم(مثلا نوع صحیح یا اعشاری). اما انواع دیگری از انواع داده وجود دارند که می توان با آن ها متغیری ایجاد کرد. از جمله آن ها می توان به enum[[1]](#footnote-1) ها و [[2]](#footnote-2)struct ها اشاره نمود.

Enumerationها نوعی از داده بوده که می تواند چندین مقدار ممکن را ذخیره کرده و به عنوان ثابت هایی تحت عنوان enumerators در نظر گرفته می شود. این نوع داده جهت ایجاد مجموعه ای از ثابت ها به کار گرفته می شود. در نظر بگیرید می خواهیم ورق بازی را با زبان C++ ایجاد کنیم. می دانیم هر دست شامل 52 کارت را شامل شده، که از یکی از چهار نوع دل[[3]](#footnote-3)، پیک[[4]](#footnote-4)، گیشنیز[[5]](#footnote-5) و خشت[[6]](#footnote-6) می باشد و هر کدام از این ها 13 تا از کارت ها را به خود اختصاص می دهد. می توانیم این مدل را به شکل زیر نماد گذاری کنیم:



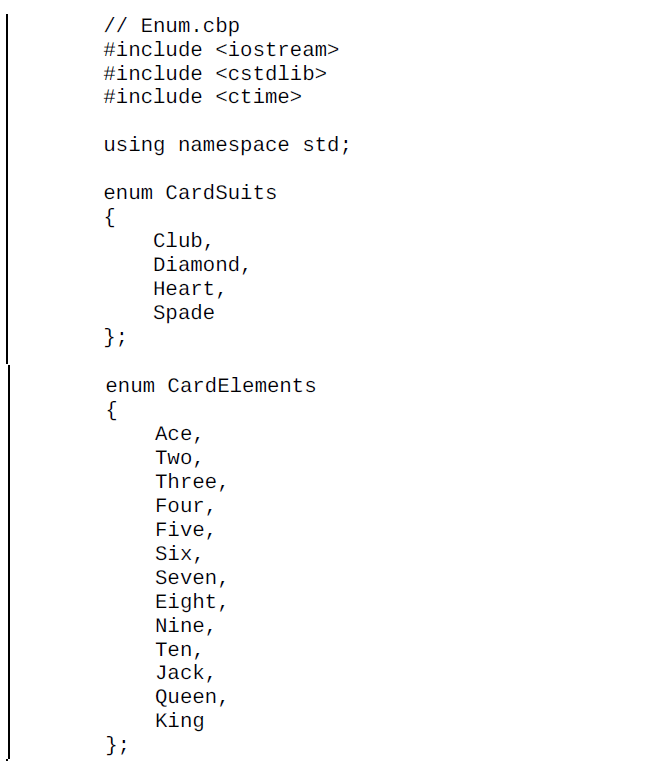


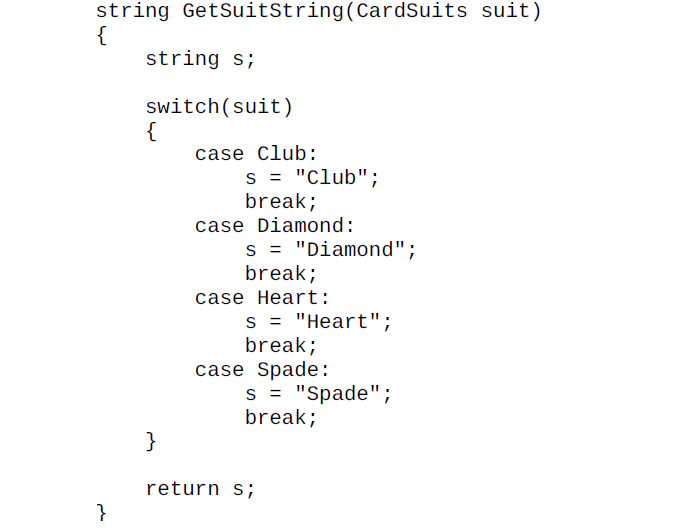
حال چنانچه بخواهیم این انواع داده را (که خودمان ایجاد کرده ایم) به کار بریم(CardSuits و CardElements) می توان از دستورات زیر مدد جست:

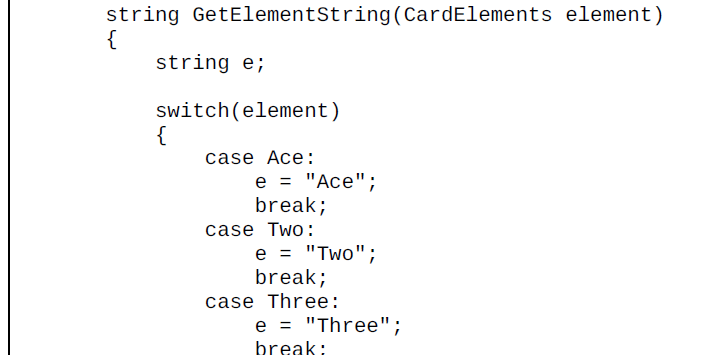


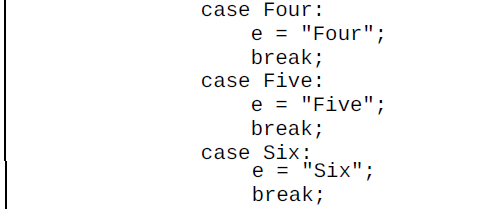
در حقیقت enum ها همیشه مقادیر عددی صحیح را نگه می دارند. رشته هایی که داخل عناصر enum به کار می روند، چیزی جز یک ثابت نیستند. اولین عنصر مقدار 0 را نگه می دارد. مگر اینکه ما به گونه ای دیگر تصمیم گرفته باشیم. عنصر بعدی از افزودن عدد یک به مقدار قبلی بدست می آید. بنابراین در enum ما یعنی CardSuits مقدار Club برابر صفر خواهد بود و Diamond ، Heart، Spade به ترتیب برابر 1، 2 و 3 خواهند بود.

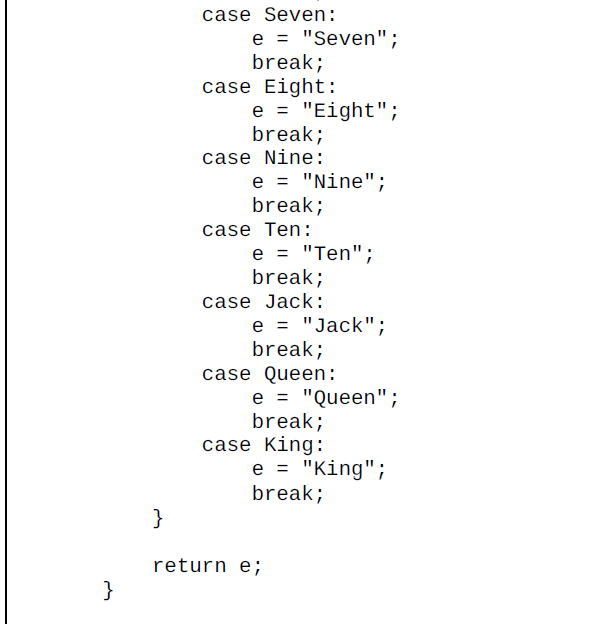
حال بیایید برنامه ای بنویسیم که یک کارت به شکل تصادفی تولید کند. در اینجا می توانیم از همان GenerateRandomNumber() که سابقا داشتیم استفاده کنیم. در زیر کد کامل این برنامه را مشاهده می کنید:

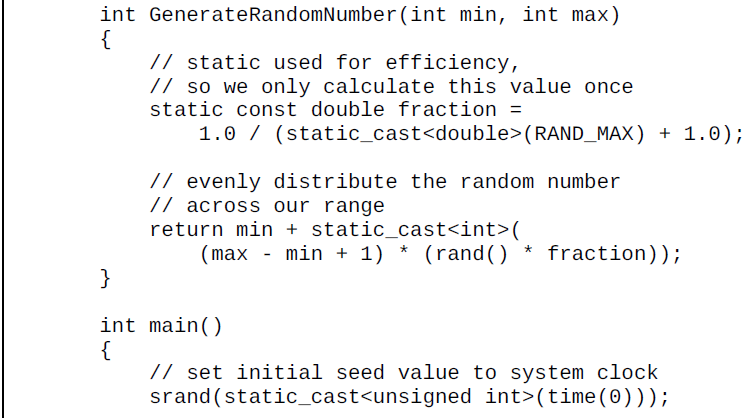


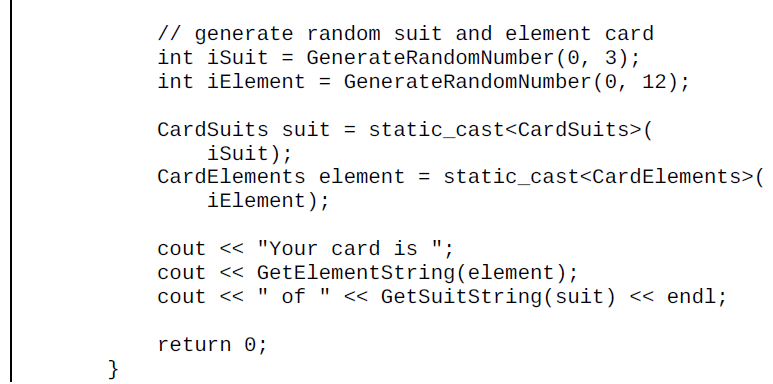




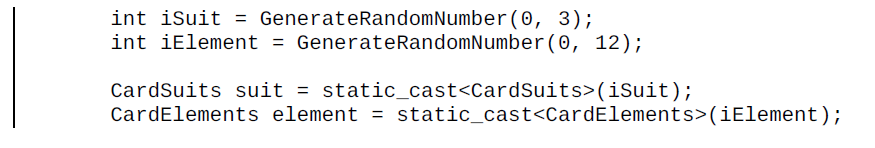




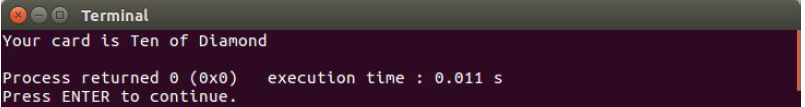




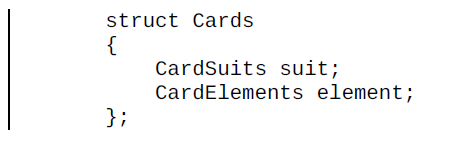
از کد بالا مشاهده می شود که می توان با استفاده از یک عدد صحیح به عناصر enum دسترسی داشت. ولی ما می بایست نوع داده int را تبدیل کنیم تا بتوان به انواع enum آن ها را نسبت داد بنابراین از static\_cast<> به شکل زیر استفاده می کنیم:



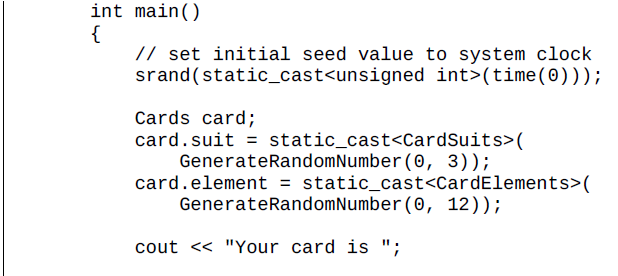
اگر کد را کامپایل کرده و اجرا کنیم با چنین پنجره ای در خروجی روبرو خواهیم شد:

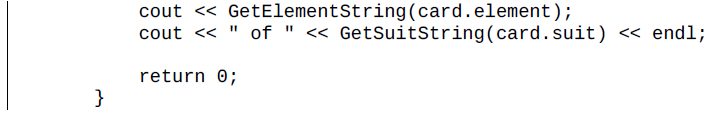


از انواع پیشرفته دیگر که در زبان C++ موجود بوده struct ها می باشند. این نوع داده یک مجموعه ای مرتبط از داده های مستقل که تحت یک گروه گرد آمده شده، می باشد. از کد قبلی ما suit و element را به عنوان متغیر هایی داریم که می توان آن ها را مانند زیر به صورت یک گروه در آوریم:



چنانچه بخواهیم struct بالا را به کد enum سابق اضافه کنیم فقط کافیست تابع main را مانند زیر بازنویسی کنیم:





چنانچه کد بالا را اجرا کنیم، خروجی مانند همان برنامه enum داریم.

توسعه انواع داده های انتزاعی

یک داده ی انتزاعی(ADT)[[7]](#footnote-7) نوعی از داده است که شامل مجموعه ای داده و عملگر های مربوط به آن بوده که این عملگرها می توانند این داده ها را دستکاری نمایند( در واقع نوع داده ی انتزاعی ترکیبی از داده و عملگر هاست). این نوع داده ای فقط به عملگر های مربوطه اشاره می کند و قرار نیست چگونه ای پیاده سازی آن را نیز مشخص کند. پیاده سازی خودش در درون آن تعریف می شود و به همین دلیل است که به این نوع داده، نوع داده ی انتزاعی می گویند.(abstract به معنای فشرده و انتزاعی)

یک دستگاه پخش کننده DVD را در نظر بگیرید، این پخش کننده یک کنترل نیز به همراه خود دارد، کنترل دکمه های متنوعی دارد از جمله پخش و ایست ویدیو، کاهش و افزایش صدا و خارج کردن دیسک از دستگاه و سایر موارد. این موضوع به مانند ADT می باشد، هنگامی که ما با دکمه افزایش صدا سعی در افزایش صدای دستگاه داریم ممکن است درباره اینکه این کار دقیقا به چه شکل انجام می شود هیچ اطلاعی نداشته باشیم( دقیقا مانند عملگر ها در ADT). ما صرفا عمل افزایش صدا را با فشردن کلید مربوطه اجرا می کنیم، بدون اینکه نیاز باشد بدانیم این کار چگونه انجام می گردد.

بر اساس آنچه در پایین گفته شده ما می بایست برای هر نوع ADT موارد مخفی سازی اطلاعات، بسته بندی یا کپسوله سازی و انتزاع را در نظر بگیریم. توضیح این موارد در زیر آمده است:

* **انتزاع** به مخفی سازی جزئیات پیاده سازی عملگر های موجود در ADT می باشد.
* **مخفی سازی اطلاعات به معنای مخفی سازی داده ها به گونه ای است که پیاده سازی به دور نگه داشته شود.**
* **کپسوله سازی یا بسته بندی به معنای دسته بندی تمامی داده ها و توابع مربوط به هم تحت یک گروه واحد می باشد.**

به کار گرفتن کلاس ها در زبان C++ جهت ساخت نوع داده ADT تعریف شده توسط کاربر

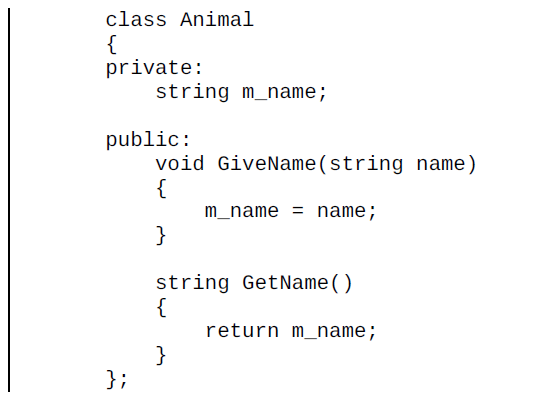
کلاس ها به مانند محفظه ای جهت نگه داری متغیر ها و عملگر(متد) های مربوط به آن ها می باشد. این متغیر ها توسط عملگر ها دستکاری می شوند. همانطور که نوع ADT روش بسته بندی یا کپسوله سازی جهت گروه بندی داده ها و توابع مربوط مشابه می باشد، کلاس ها نیز می توانند به این منظور( یعنی گروه بندی داده ها و توابع مربوط به آن ها) به کار گرفته شوند.

هر کلاس سه نوع کنترل دسترسی را برای متغیر ها و توابع شامل می شود:

* دسترسی عمومی(Public): داده ها و توابع توسط هر آن کس که از کلاس استفاده می کند قابل دسترسی می باشد.
* دسترسی محافظت شده(Protected): داده ها و توابع فقط توسط خود توابع کلاس و همچنین کلاس های در سلسله ی وراثت می تواند مورد استفاده قرار بگیرید.( می توانید این موضوع را فعلا نادیده بگیرید، اگر متوجه نشده اید اشکالی ندارد☺)
* دسترسی خصوصی(Private): داده ها و توابع فقط می تواند در داخل خود کلاس و یا توابع و کلاس های دوست( آن هایی که با قواعد به خصوصی خودمان اجازه دسترسی می دهیم) مورد استفاده قرار گیرد.

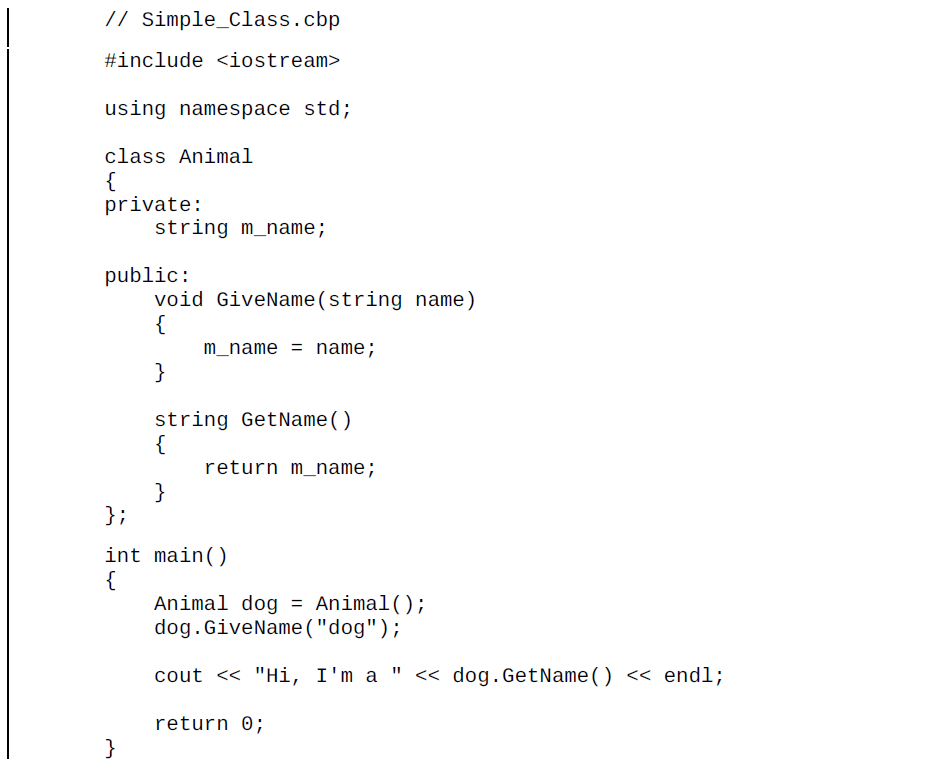
خوب بیایید به همان مبحث قبلی انواع انتزاعی و مخفی سازی و بسته بندی داده ها در کلاس بپردازیم. این امکان وجود دارد که کپسوله سازی را با استفاده از سطح دسترسی private و یا protected پیاده سازی کرد. می توان از این دو کلید واژه در جهت مخفی سازی توابع عضو کلاس(متد ها) در خارج از آن استفاده کرد. همچنین متغیر های کلاس( داده های عضو) را نیز می توان به همین شکل از دسترسی در خارج از کلاس به دور نگه داشت.

حال به مانند زیر یک کلاس ساده با نام Animal ایجاد می کنیم:



همانطور که در قطعه کد بالا مشاهده می کنیم،از آن جا که ما m\_name را private تعریف کرده ایم، به طور مستقیم نمی توانیم به این متغیر( داده ی عضو) دسترسی داشته باشیم. با این حال دو تابع عضو(متد) به شکل public داریم که می توانند به این متغیر در داخل کلاس دسترسی داشته باشند. تابع عضوGiveName() مقدار متغیر m\_name را تغییر می دهد. و تابع GetName() مقدار این متغیر را باز خواهد گرداند.

قطعه کد زیر چگونگی استفاده از این کلاس را بیان می دارد:



1. enumeration [↑](#footnote-ref-1)
2. ساختارها [↑](#footnote-ref-2)
3. heart [↑](#footnote-ref-3)
4. spead [↑](#footnote-ref-4)
5. club [↑](#footnote-ref-5)
6. diamond [↑](#footnote-ref-6)
7. Abstract data type [↑](#footnote-ref-7)