

# C++

## به زبان ساده

منبع کتاب : [w3-farsi.com](http://w3-farsi.com)

مؤلف : یونس ابراهیمی

# راه‌های ارتباط با نویسنده

سایت : [w3-farsi.com](http://w3-farsi.com)

جیمیل : [younes.ebrahimi.1391@gmail.com](mailto:younes.ebrahimi.1391@gmail.com)

## Contents

۸	..... چیست C++
۹	..... نصب و پیکربندی MinGW
۱۶	..... ساخت یک برنامه ساده
۲۰	..... توضیحات
۲۱	..... کاراکترهای کنترلی
۲۳	..... متغیر
۲۴	..... انواع ساده
۲۶	..... استفاده از متغیرها
۳۰	..... ثابت
۳۱	..... عبارات و عملگرها
۳۲	..... عملگرهای ریاضی
۳۵	..... عملگرهای تخصیصی
۳۶	..... عملگرهای مقایسه ای
۳۸	..... عملگرهای منطقی
۴۵	..... تقدم عملگرها
۴۷	..... گرفتن ورودی از کاربر
۴۸	..... ساختارهای تصمیم
۴۹	..... دستور if

۵۳	..... دستور if else
۵۴	..... عملگر شرطی
۵۵	..... دستور if چندگانه
۵۷	..... دستور if تو در تو
۵۹	..... استفاده از عملگرهای منطقی
۶۱	..... دستور Switch
۶۵	..... تکرار
۶۶	..... حلقه While
۶۷	..... حلقه do while
۶۹	..... حلقه for
۷۰	..... حلقه‌های تو در تو (Nested Loops)
۷۲	..... خارج شدن از حلقه با استفاده از break و continue
۷۳	..... آرایه‌ها
۷۶	..... آرایه‌های چند بعدی
۸۱	..... متد
۸۲	..... مقدار برگشتی از یک متد
۸۵	..... پارامترها و آرگومان‌ها
۸۸	..... ارسال آرگومان‌ها به روش ارجاع
۹۰	..... ارسال آرایه به عنوان آرگومان
۹۲	..... محدوده متغیر
۹۲	..... پارامترهای اختیاری
۹۴	..... سربارگذاری متدها
۹۵	..... بازگشت (Recursion)
۹۷	..... شمارش
۱۰۱	..... (Enumeration)
۱۰۱	..... پیش پردازنده ها
۱۰۳	..... اشاره گرها
۱۰۳	..... ساختارها
۱۰۵	..... برنامه نویسی شیء گرا

۱۰۶	..... (Object Oriented Programming)
۱۰۶	..... کلاس
۱۰۸	..... سازنده ها (Constructors)
۱۱۱	..... مخرب ها (Destructors)
۱۱۴	..... فیلدهای فقط - خواندنی
۱۱۶	..... سطح دسترسی
۱۱۸	..... کپسوله کردن
۱۲۰	..... (Encapsulation)
۱۲۲	..... خواص (Properties)
۱۲۴	..... فضای نام
۱۲۸	..... وراثت
۱۳۱	..... سطح دسترسی
۱۳۳	..... Protect
۱۳۵	..... اعضای Static
۱۳۷	..... کلاس Static
۱۳۹	..... سربارگذاری عملگرها
۱۴۲	..... چند ریختی (Polymorphism)

تقديم به:

# همسر و پسر عزيزم



فصل اول



# مبانی زبان سی پلاس پلاس

## C++ چیست

C++ یک زبان برنامه نویسی است که از paradigm یا سبک‌های مختلف برنامه نویسی پشتیبانی می‌کند. C++ نسخه توسعه یافته زبان C می‌باشد و بیشتر کدهای زبان C به راحتی می‌تواند در C++ کامپایل شود. در C++ از ویژگی‌های مهمی که به C اضافه شده است می‌توان به برنامه نویسی شی گرا، سربارگذاری عملگرها، وراثت چندگانه و مدیریت خطاها اشاره نمود. توسعه C++ در سال ۱۹۷۹ آغاز شد و ۷ سال پس از زبان C به نمایش گذاشته شد. با وجود قدیمی بودن زبان‌های C و C++ هنوز هم به صورت گسترده ای در نرم افزارهای صنعتی مورد استفاده قرار می‌گیرد. این زبان‌ها برای ساخت هر چیزی از سیستم عامل گرفته تا نرم افزارهای توکار، برنامه‌های دسکتاپ و بازی‌ها مورد استفاده قرار می‌گیرد.

در مقایسه با زبان‌های جدید تر، برنامه‌های نوشته شده با C++ اغلب پیچیده تر می‌باشند و زمان بیشتری برای توسعه نیاز دارد. در عوض، C++ زبانی است که به شما اجازه می‌دهد که هم به صورت High-level (نزدیک به زبان انسان) و هم به صورت Low-level (نزدیک به زبان ماشین) سخت افزار را تحت کنترل خود قرار دهید. همچنین با پشتیبانی از سبک‌ها یا پارادایم‌های مختلف برنامه نویسی از جمله رویه ای، شی گرا یا عمومی، دست برنامه نویس را در انتخاب سبک مورد نظرش آزاد می‌گذارد. اکنون ۵ نسخه از استاندارد این زبان منتشر شده است؛ و استاندارد C++17 نیز برای انتشار در سال ۲۰۱۷ برنامه ریزی شده است.

نام غیر رسمی	استاندارد C++	سال
C++98	ISO/IEC 14882:1998	1998
C++03	ISO/IEC 14882:2003	2003
C++07/TR1	ISO/IEC TR 19768:2007	2007
C++11	ISO/IEC 14882:2011	2011
C++14	ISO/IEC 14882:2014	2014
C++17	هنوز تعیین نشده.	2017

برای اجرای کدهای C++ نیاز به یک کامپایلر داریم. کامپایلرها و محیط‌های برنامه نویسی (IDE) گوناگونی برای زبان C++ وجود دارند از بین معروفترین آنها می‌توان موارد زیر اشاره نمود:

- Turbo C
- Turbo C++
- Borland C++
- Microsoft C++/C



زبان C++ وابسته به یک سیستم عامل نیست یعنی شما بعد از نوشتن برنامه خود به زبان C++ ، اگر کد استاندارد نوشته باشید می‌توانید با توجه به سیستم عامل، کدتان را کامپایل کنید. می‌توان کد C++ را در هر محیطی، مثلاً در NotePad در ویندوز و یا gEdit در گنو/لینوکس نوشته و بعد آن را بوسیله یک کامپایلر کامپایل کنیم، ولی برای راحتی کار ما می‌توانیم از یک IDE مناسب، نیز بهره ببریم. البته در این سری آموزشی ما از راحت‌ترین روش برای کامپایل کدها استفاده می‌کنیم.

## نصب و پیکربندی MinGW

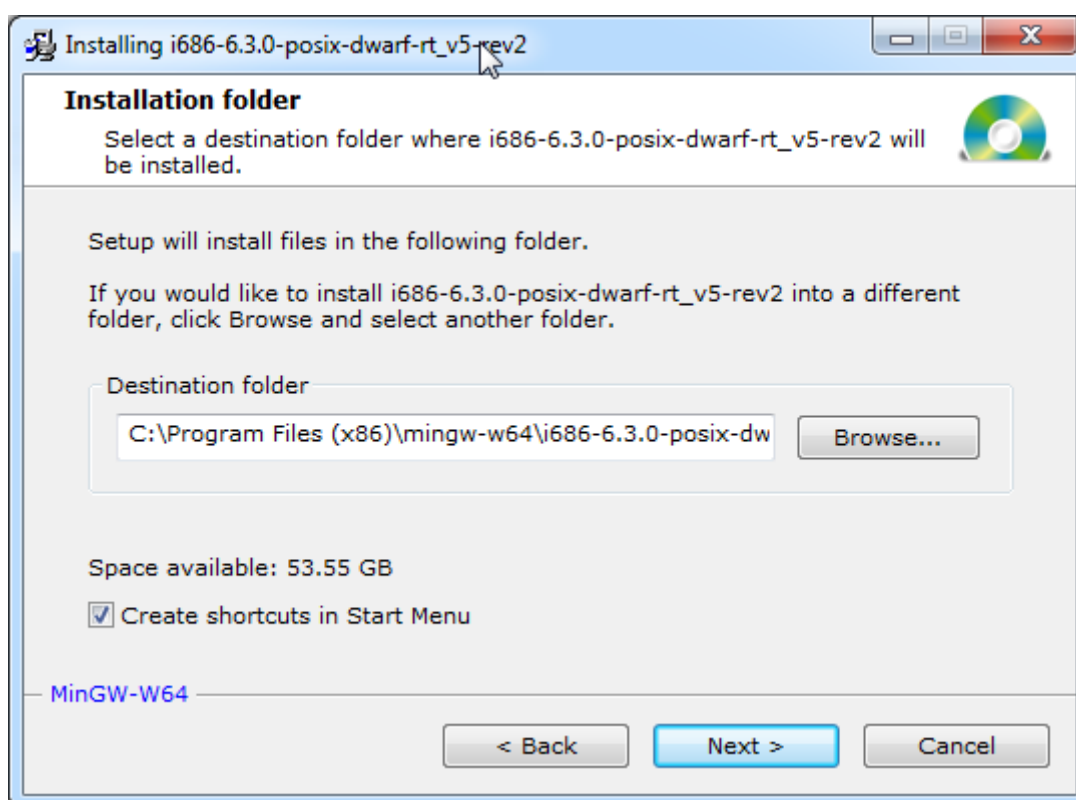
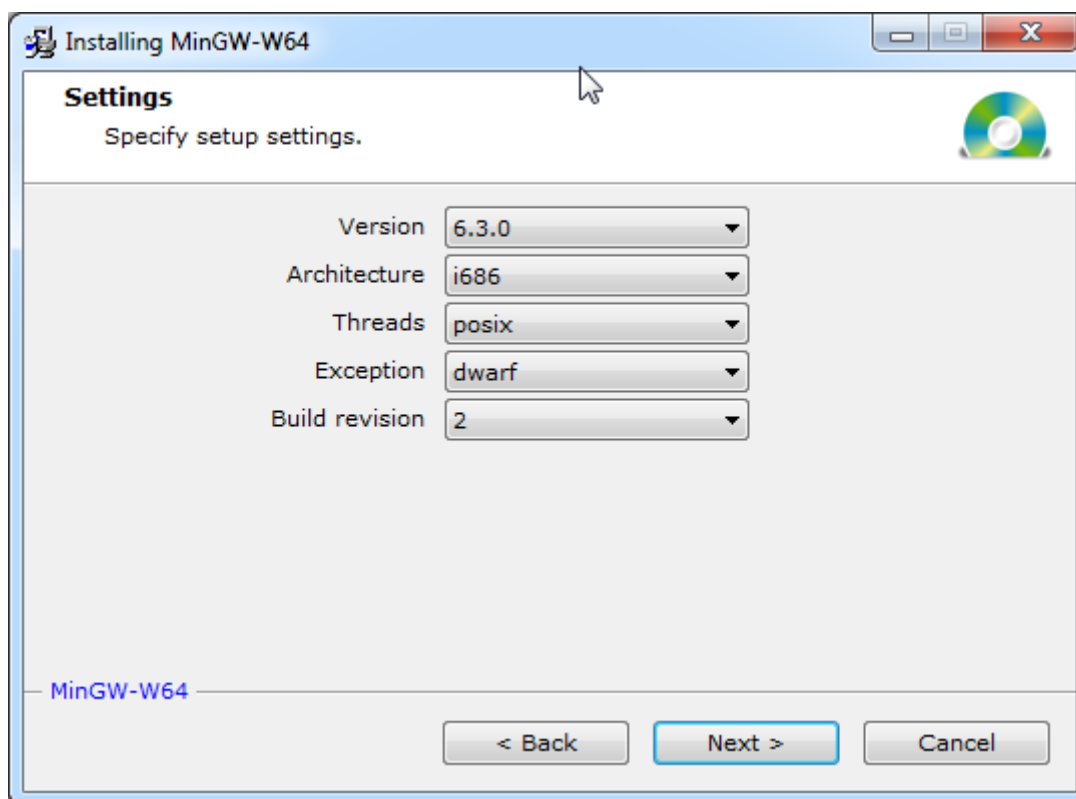
همانطور که اشاره شد برای اجرای کدهای C++ نیاز به یک کامپایلر داریم. کامپایلری که در این سری آموزشی می‌خواهیم از آن استفاده کنیم MinGW می‌باشد که می‌توانید آن را از لینک زیر دانلود کنید:

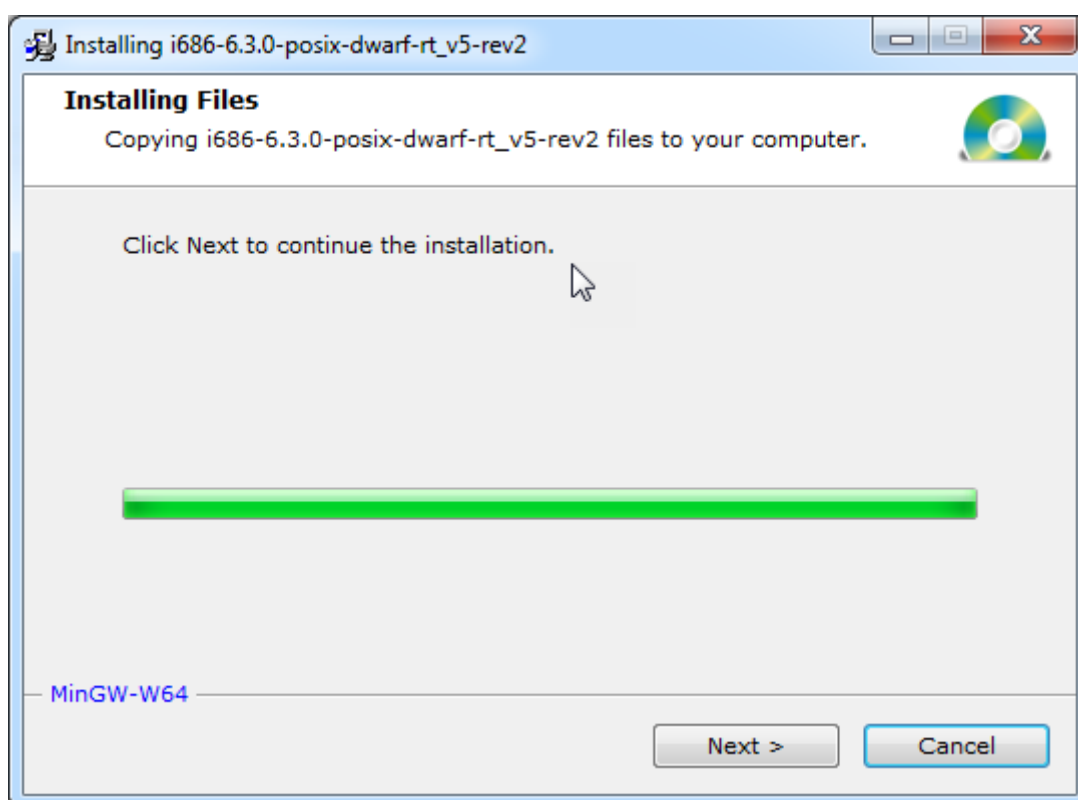
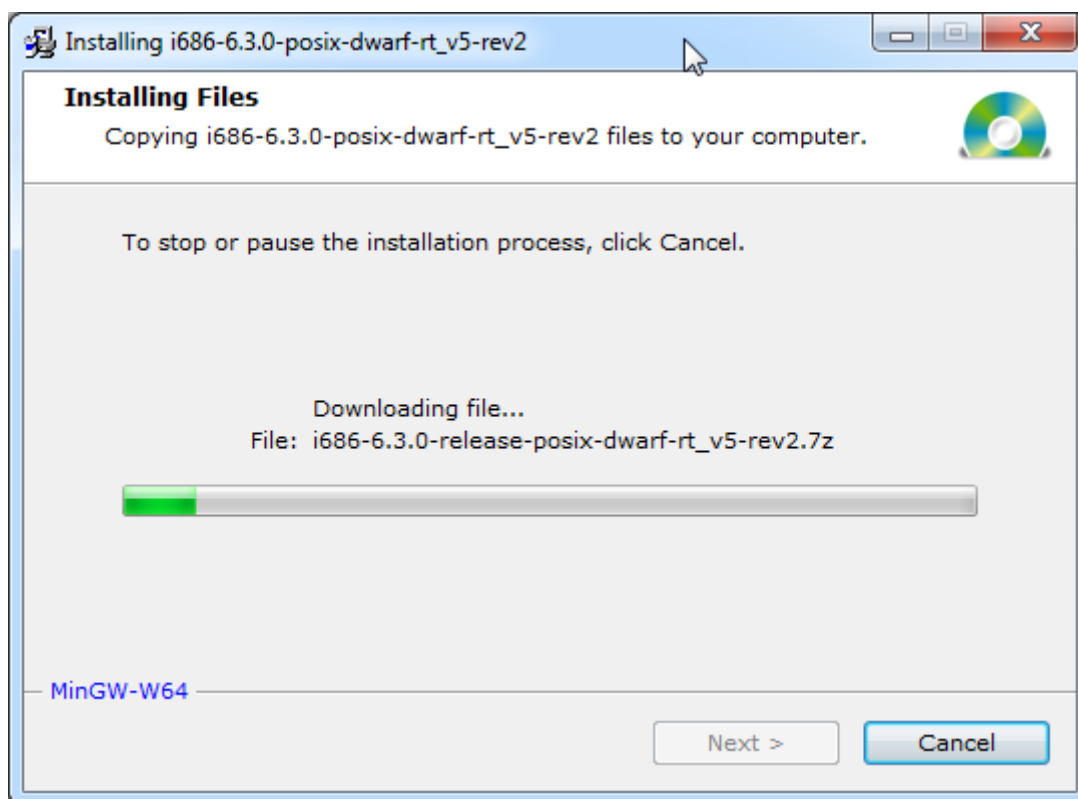
[dl.w3-farsi.com/Software/Cplusplus/mingw-w64-install.exe](http://dl.w3-farsi.com/Software/Cplusplus/mingw-w64-install.exe)

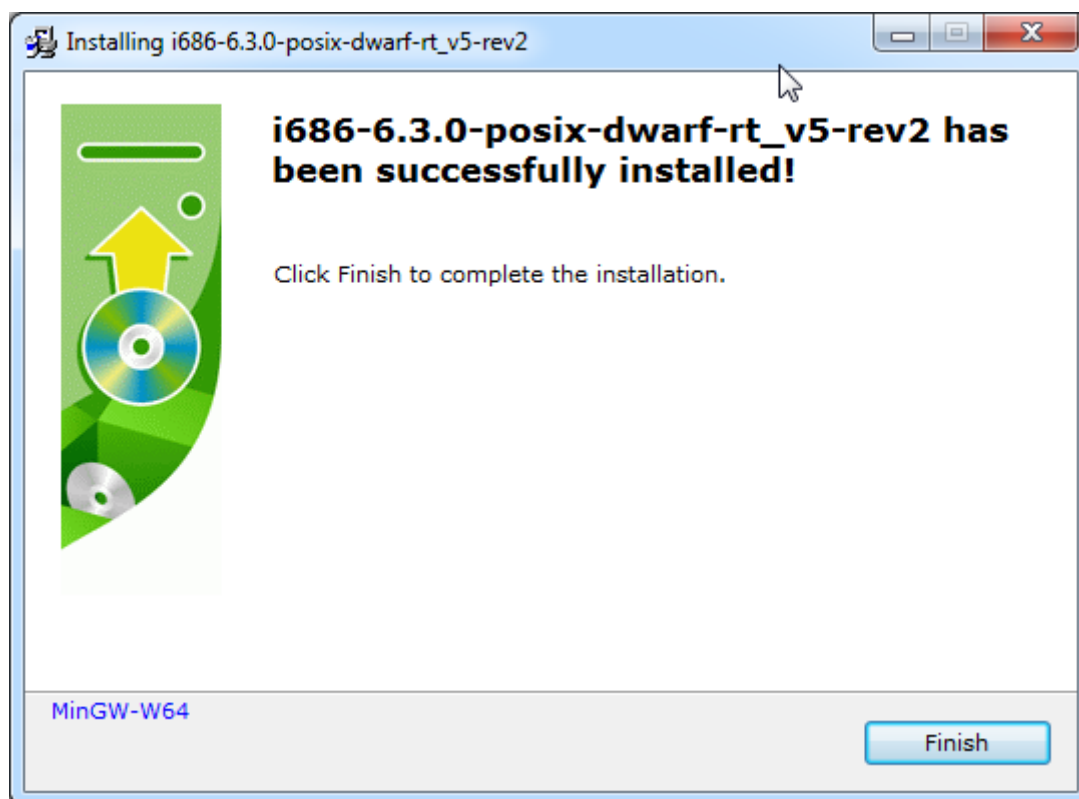
## نصب MinGW

بعد از دانلود فایل بالا بر روی آن دو بار کلیک کرده و مراحل زیر را برای نصب و پیکربندی آن طی کنید:







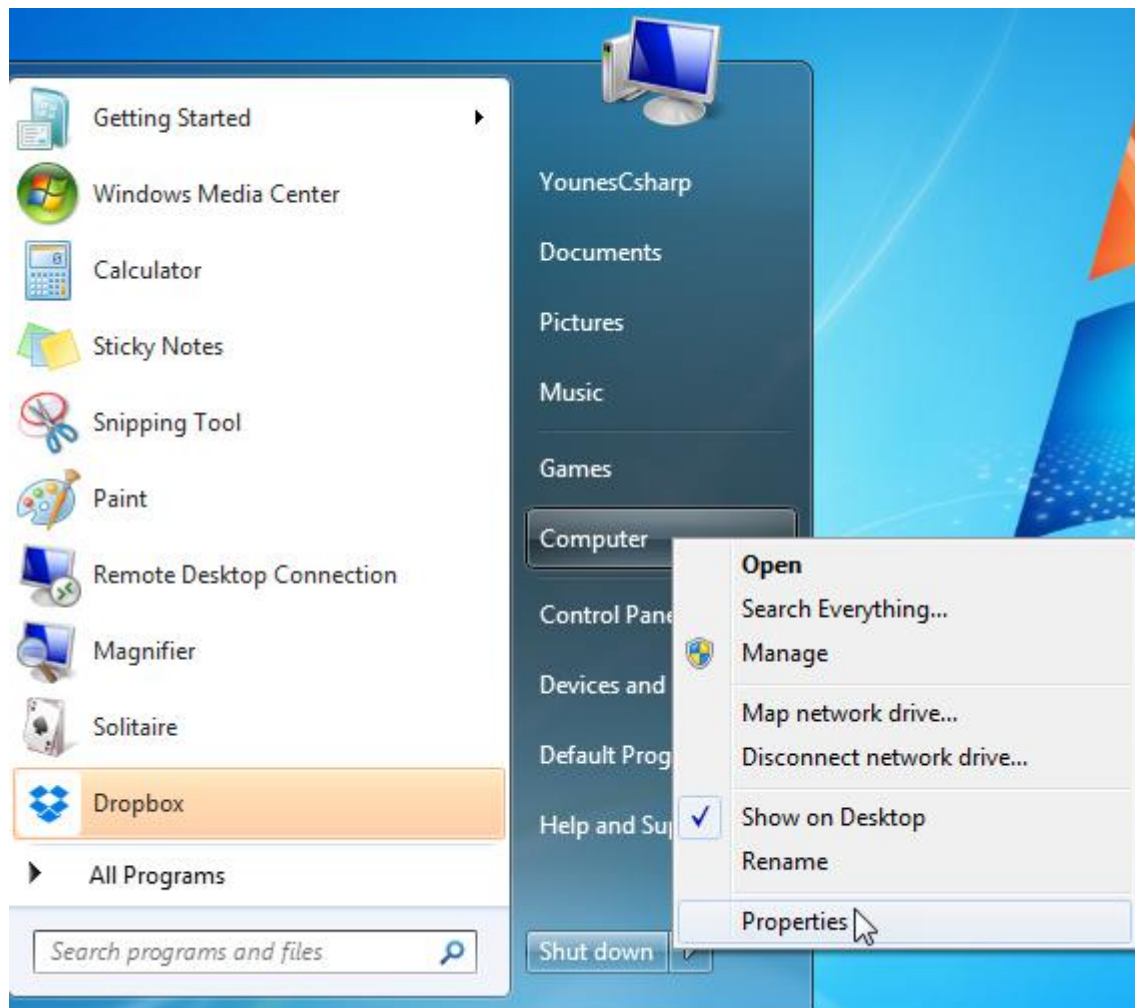


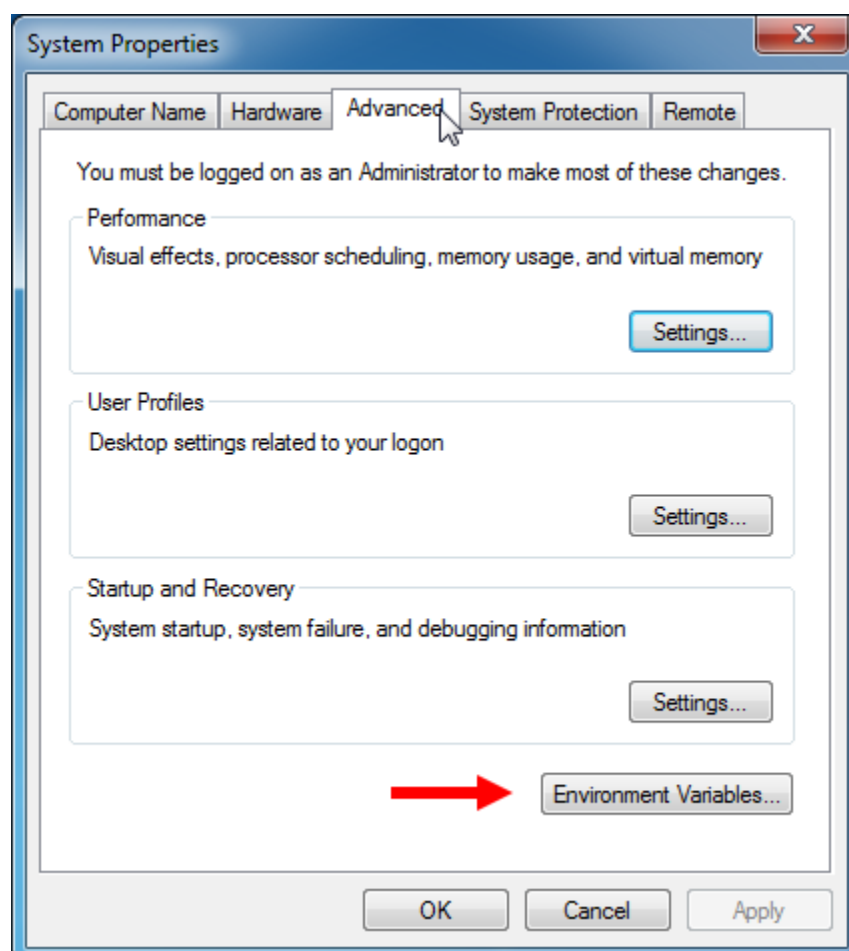
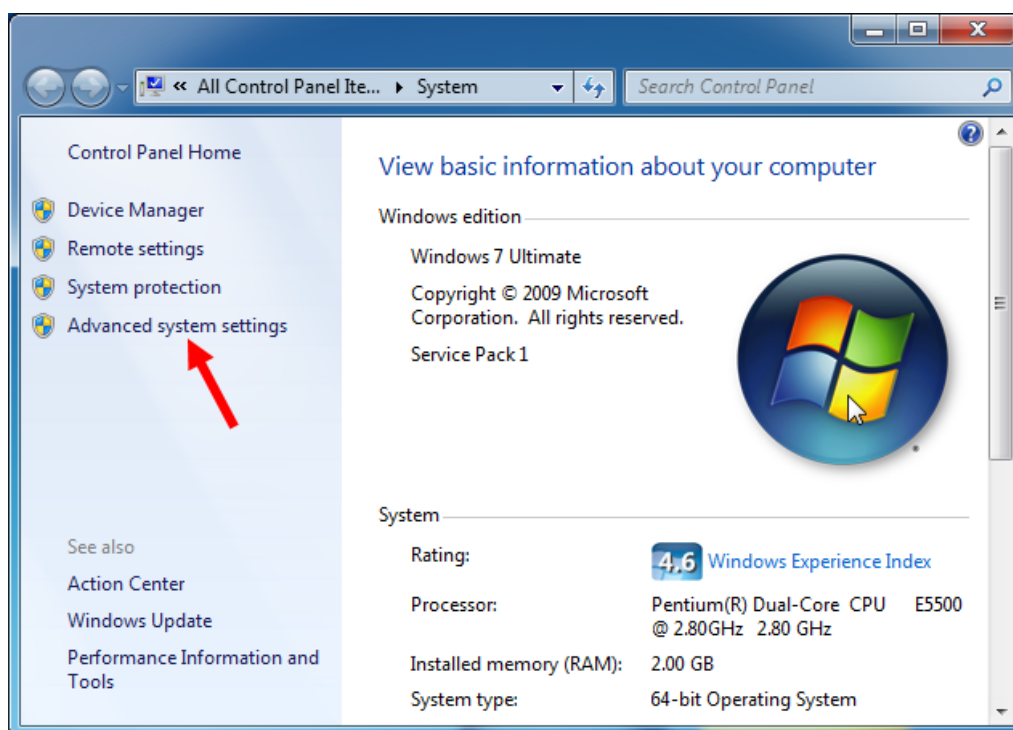
## پیکربندی MinGW

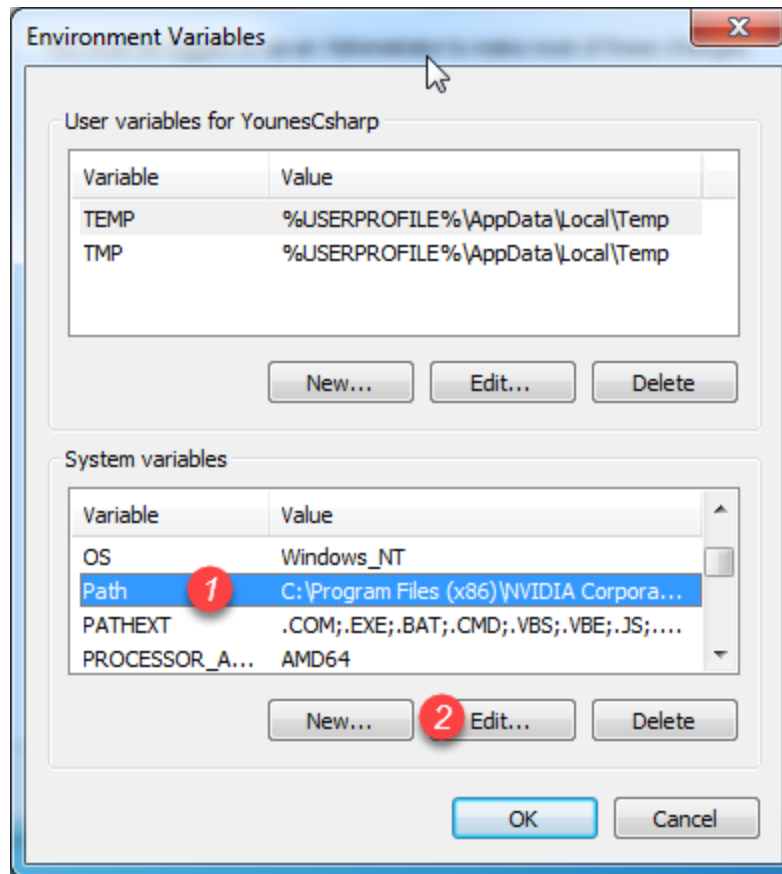
بعد از اتمام نصب نوبت به پیکربندی MinGW می‌رسد. در کل باید مسیر پوشه bin نرم افزار MinGW را در متغیر PATH ویندوز قرار دهید. پوشه bin نرم افزار MinGW ما در مسیر زیر قرار دارد و البته ممکن است برای شما این مسیر متفاوت باشد:

```
C:\Program Files (x86)\mingw-w64\i686-6.3.0-posix-dwarf-rt_v5-rev2\mingw32\bin
```

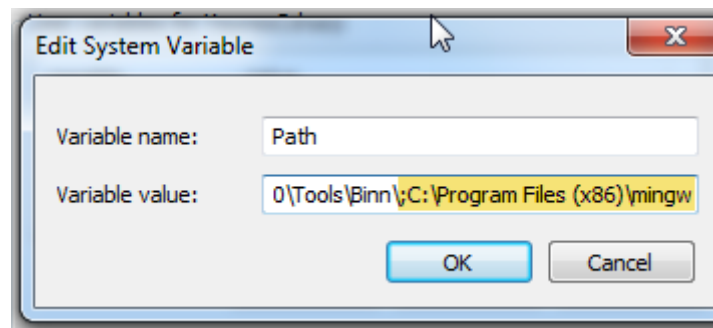
مسیر پوشه bin را کپی کرده و مراحل زیر را طی کنید:







همانطور که در شکل بالا مشاهده می‌کنید بعد از کلیک بر روی متغیر path سپس دکمه Edit پنجره‌ای به صورت زیر نشان داده می‌شود. نشانگر ماوس را به آخر جعبه متن برده و سپس یک علامت سمیکالن (;) گذاشته و مسیر پوشه bin خود را بعد از علامت سمیکالن کپی کنید :



در نهایت دکمه OK تمام پنجره‌های باز را بزنید. با این کار پیکربندی هم به اتمام می‌رسد. در درس بعد نحوه اجرای اولین برنامه با سی پلاس پلاس را به شما آموزش می‌دهیم.

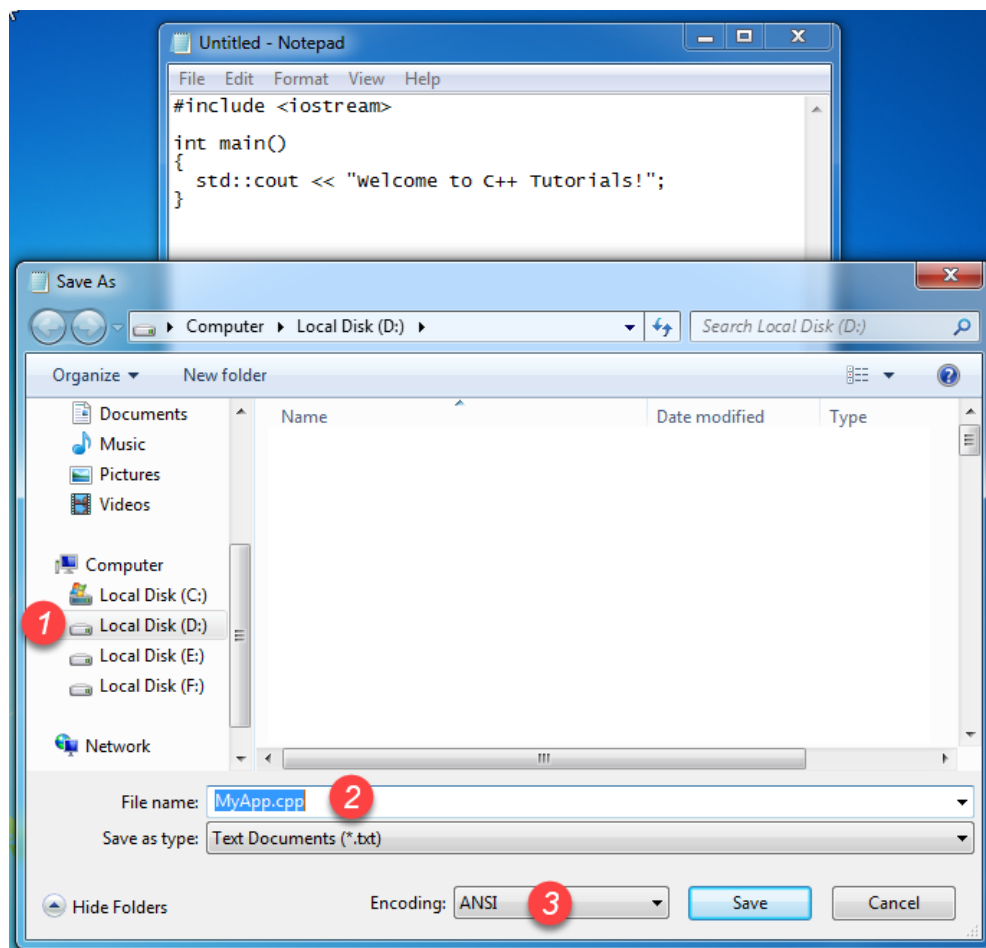
## ساخت یک برنامه ساده

اجازه بدهید یک برنامه بسیار ساده به زبان سی پلاس پلاس (C++) بنویسیم. این برنامه یک پیغام را در محیط کنسول نمایش می‌دهد. در این درس می‌خواهم ساختار و دستور زبان یک برنامه ساده C++ را توضیح دهم. هر چند که محیط‌های کدنویسی زیادی برای C++ وجود دارند ولی ما از ساده‌ترین روش برای کدنویسی استفاده می‌کنیم. برنامه Notepad ویندوز را باز کرده و کدهای زیر را در داخل آن بنویسید:

```
#include <iostream>

int main()
{
    std::cout << "Welcome to C++ Tutorials!";
}
```

حال فایل را با نام MyApp و پسوند .cpp به صورت زیر در درایو D ذخیره کنید:





## ساختار یک برنامه در C++

مثال بالا ساده‌ترین برنامه ای است که شما می‌توانید در C++ بنویسید . هدف در مثال بالا نمایش یک پیغام در صفحه نمایش است. هر زبان برنامه نویسی دارای قواعدی برای کدنویسی است. اجازه بدهید هر خط کد را در مثال بالا توضیح بدهیم. در خط اول فایل هدر یا سرایند آمده است. فایل‌های سرآیند کتابخانه استاندارد C++ می‌باشند و در این برنامه ما به فایل سرایند `iostream` نیاز داریم (در درس‌های آینده در مورد این فایل‌ها به طور مفصل توضیح می‌دهیم). خط ۳ متد `main()` یا متد اصلی نامیده می‌شود. هر متد شامل یک سری کد است که وقتی اجرا می‌شوند که متد را صدا بزنیم. درباره متد و نحوه صدا زدن آن در فصول بعدی توضیح خواهیم داد. متد `main()` نقطه آغاز اجرای برنامه است. این بدان معناست که ابتدا تمام کدهای داخل متد `main()` و سپس بقیه کدها اجرا می‌شود. درباره متد `main()` در فصول بعدی توضیح خواهیم داد. متد `main()` و سایر متدها دارای آکولاد و کدهایی در داخل آن‌ها می‌باشند و وقتی کدها اجرا می‌شوند که متدها را صدا بزنیم. هر خط کد در C++ به یک سمیکالن ( ; ) ختم می‌شود. اگر سمیکالن در آخر خط فراموش شود برنامه با خطا مواجه می‌شود. مثالی از یک خط کد در C++ به صورت زیر است:

```
std::cout << "Welcome to C++ Tutorials!";
```

این خط کد پیغام `Welcome to Visual C++ Tutorials!` را در صفحه نمایش نشان می‌دهد. از شیء `cout` برای چاپ یک رشته استفاده می‌شود. یک رشته گروهی از کاراکترها است، که به وسیله دابل کوتیشن (") محصور شده است . مانند `"Welcome to Visual C++ Tutorials!"`.

یک کاراکتر می‌تواند یک حرف، عدد، علامت یا ... باشد. در کل مثال بالا نحوه استفاده از شیء `cout` است که در داخل فضای نام `std` قرار دارد را نشان می‌دهد. توضیحات بیشتر در درس‌های آینده آمده است C++. فضای خالی و خطوط جدید را نادیده می‌گیرد. بنابراین شما می‌توانید همه برنامه را در یک خط بنویسید. اما اینکار خواندن و اشکال زدایی برنامه را مشکل می‌کند. یکی از خطاهای معمول در برنامه نویسی فراموش کردن سمیکالن در پایان هر خط کد است . به مثال زیر توجه کنید :

```
std::cout <<
"Welcome to C++ Tutorials!";
```

سی پلاس پلاس فضای خالی بالا را نادیده می‌گیرد و از کد بالا اشکال نمی‌گیرد. اما از کد زیر ایراد می‌گیرد:

```
std::cout << ;
"Welcome to C++ Tutorials!";
```

به سمیکالین آخر خط اول توجه کنید. برنامه با خطای نحوی مواجه می‌شود چون دو خط کد مربوط به یک برنامه هستند و شما فقط باید یک سمیکالین در آخر آن قرار دهید. همیشه به یاد داشته باشید که C++ به بزرگی و کوچکی حروف حساس است. یعنی به طور مثال MAN و man در سی پلاس پلاس با هم فرق دارند. رشته‌ها و توضیحات از این قاعده مستثنی هستند که در درس‌های آینده توضیح خواهیم داد. مثلاً کدهای زیر با خطا مواجه می‌شوند و اجرا نمی‌شوند:

```
std::cOUT << "Welcome to C++ Tutorials!";
STD::cout << "Welcome to C++ Tutorials!";
Std::Cout << "Welcome to C++ Tutorials!";
```

تغییر در بزرگی و کوچکی حروف از اجرای کدها جلوگیری می‌کند. اما کد زیر کاملاً بدون خطا است:

```
std::cout << "Welcome to C++ Tutorials!";
```

همیشه کدهای خود را در داخل آکولاد بنویسید.

```
{
    statement1;
}
```

این کار باعث می‌شود که کدنویسی شما بهتر به چشم بیاید و تشخیص خطاها راحت تر باشد.

## کامپایل و اجرای برنامه

برای کامپایل و اجرای برنامه، cmd ویندوز را باز کرده و سپس به درایو D یعنی جایی که فایل را ذخیره کرده‌اید بروید:

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\YounesCsharp>D:

D:\>
```

حال برای کامپایل برنامه کد زیر را نوشته و دکمه Enter را بزنید:

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\YounesCsharp>D:

D:\>g++ MyApp.cpp -o MyApp.exe
```

و در نهایت برای اجرای برنامه کد زیر را بنویسید:

```
Microsoft Windows [Version 6.1.7601]
```

```
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Users\YounesCsharp>D:
D:\>g++ MyApp.cpp -o MyApp.exe
D:\>MyApp.exe
Welcome to C++ Tutorials!
D:\>
```

مشاهده می‌کنید که پیغام Welcome to C++ Tutorials چاپ می‌شود.

## وارد کردن فضای نام در برنامه

در برنامه فوق ما یک فضای نام در برنامه مان با نام std داریم، اما سی پلاس پلاس دارای تعداد زیادی فضای نام می‌باشد. یکی از این فضاهای نامی، فضای نام std است که شیء cout که ما از آن در برنامه بالا استفاده کردیم در این فضای نام قرار دارد.

```
std::cout << "Welcome to C++ Tutorials!";
```

اینکه قبل از استفاده از هر کلاس ابتدا فضای نام آن را مانند کد بالا بنویسیم کمی خسته کننده است. خوشبختانه C++ به ما اجازه می‌دهد که برای جلوگیری از تکرار مکررات، فضاهای نامی را که قرار است در برنامه استفاده کنیم با استفاده از دستور using و کلمه namespace در ابتدای برنامه وارد نماییم:

```
using namespace NameofNameSpace;
```

دستور بالا نحوه وارد کردن یک فضای نام در برنامه را نشان می‌دهد. در نتیجه به جای آنکه به صورت زیر ابتدا نام فضای نام و سپس نام کلاس را بنویسیم:

```
std::cout << "Welcome to C++ Tutorials!";
```

می‌توانیم فضای نام را با دستوری که ذکر شد وارد برنامه کرده و کد بالا را به صورت خلاصه شده زیر بنویسیم:

```
cout << "Welcome to C++ Tutorials!";
```

دستورات using که باعث وارد شدن فضاهای نامی به برنامه می‌شوند عموماً در ابتدای برنامه و قبل از همه کدها نوشته می‌شوند، پس برنامه این درس را می‌توان به صورت زیر نوشت:

```
#include <iostream>
using namespace std;
```

```
int main()
{
    cout << "Welcome to C++ Tutorials!";
}
```

حال که با خصوصیات و ساختار اولیه C++ آشنا شدید در درس های آینده مطالب بیشتری از این زبان برنامه نویسی قدرتمند خواهید آموخت.

## توضیحات

وقتی که کدی تایپ می کنید شاید بخواهید که متنی جهت یادآوری وظیفه آن کد به آن اضافه کنید. در C++ (و بیشتر زبانهای برنامه نویسی) می توان این کار را با استفاده از توضیحات انجام داد. توضیحات متونی هستند که توسط کامپایلر نادیده گرفته می شوند و به عنوان بخشی از کد محسوب نمی شوند.

هدف اصلی از ایجاد توضیحات، بالا بردن خوانایی و تشخیص نقش کدهای نوشته شده توسط شما، برای دیگران است. فرض کنید که می خواهید در مورد یک کد خاص، توضیح بدهید، می توانید توضیحات را در بالای کد یا کنار آن بنویسید. از توضیحات برای مستند سازی برنامه هم استفاده می شود. در برنامه زیر نقش توضیحات نشان داده شده است:

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     // This line will print the message hello world
7     cout << "Hello World!";
8 }
```

در کد بالا، خط ۶ یک توضیح درباره خط ۷ است که به کاربر اعلام می کند که وظیفه خط ۷ چیست؟ با اجرای کد بالا فقط جمله Hello World چاپ شده و خط ۷ در خروجی نمایش داده نمی شود چون کامپایلر توضیحات را نادیده می گیرد. توضیحات بر دو نوع اند:

توضیحات تک خطی

```
// single line comment
```

توضیحات چند خطی

```
/* multi
   line
comment */
```

توضیحات تک خطی همانگونه که از نامش پیداست، برای توضیحاتی در حد یک خط به کار می‌روند. این توضیحات با علامت // شروع می‌شوند و هر نوشته ای که در سمت راست آن قرار بگیرد جز توضیحات به حساب می‌آید. این نوع توضیحات معمولاً در بالا یا کنار کد قرار می‌گیرند. اگر توضیح درباره یک کد به بیش از یک خط نیاز باشد از توضیحات چند خطی استفاده می‌شود. توضیحات چند خطی با /\* شروع و با \*/ پایان می‌یابند. هر نوشته ای که بین این دو علامت قرار بگیرد جز توضیحات محسوب می‌شود.

## کاراکترهای کنترلی

کاراکترهای کنترلی کاراکترهای ترکیبی هستند که با یک بک اسلش (\) شروع می‌شوند و به دنبال آن‌ها یک حرف یا عدد می‌آید و یک رشته را با فرمت خاص نمایش می‌دهند. برای مثال برای ایجاد یک خط جدید و قرار دادن رشته در آن می‌توان از کاراکتر کنترلی \n استفاده کرد:

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello\nWorld";
}
```

```
Hello
World
```

مشاهده کردید که کامپایلر بعد از مواجهه با کاراکتر کنترلی \n نشانگر ماوس را به خط بعد برده و بقیه رشته را در خط بعد نمایش می‌دهد. جدول زیر لیست کاراکترهای کنترلی و کارکرد آن‌ها را نشان می‌دهد:

عملکرد	کاراکتر کنترلی	عملکرد	کاراکتر کنترلی
Form Feed	\f	چاپ کوتیشن	\'
خط جدید	\n	چاپ دابل کوتیشن	\"
سر سطر رفتن	\r	چاپ بک اسلش	\\
حرکت به صورت افقی	\t	چاپ فضای خالی	\0
حرکت به صورت عمودی	\v	صدای بیپ	\a

چاپ کاراکتر یونیکد	\u	حرکت به عقب	\b
--------------------	----	-------------	----

ما برای استفاده از کاراکترهای کنترلی از بک اسلش (\) استفاده می‌کنیم. از آنجاییکه \ معنای خاصی به رشته‌ها می‌دهد برای چاپ بک اسلش (\) باید از (\\) استفاده کنیم:

```
cout << "We can print a \\ by using the \\\\ escape sequence.";
We can print a \ by using the \\ escape sequence.
```

یکی از موارد استفاده از \\، نشان دادن مسیر یک فایل در ویندوز است:

```
cout << "C:\\Program Files\\Some Directory\\SomeFile.txt";
C:\\Program Files\\Some Directory\\SomeFile.txt
```

از آنجاییکه از دابل کوتیشن (") برای نشان دادن رشته‌ها استفاده می‌کنیم برای چاپ آن از \" استفاده می‌کنیم:

```
cout << "I said, \"Motivate yourself!\".";
I said, "Motivate yourself!".
```

همچنین برای چاپ کوتیشن (') از \' استفاده می‌کنیم:

```
cout << "The programmer\'s heaven.";
The programmer's heaven.
```

برای ایجاد فاصله بین حروف یا کلمات از \t استفاده می‌شود:

```
cout << "Left\\tRight";
Left Right
```

هر تعداد کاراکتر که بعد از کاراکتر کنترلی \r بیایند به اول سطر منتقل و جایگزین کاراکترهای موجود می‌شوند:

```
cout << "Mitten\\rK";
Kitten
```

مثلاً در مثال بالا کاراکتر K بعد از کاراکتر کنترلی \r آمده است. کاراکتر کنترلی حرف K را به ابتدای سطر برده و جایگزین حرف M می‌کند. برای چاپ کاراکترهای یونیکد می‌توان از \u استفاده کرد. برای استفاده از \u، مقدار در مبنای ۱۶ کاراکتر را درست بعد از علامت \u قرار می‌دهیم. برای مثال اگر بخواهیم علامت (π) را چاپ کنیم باید بعد از علامت \u مقدار ۰۰ A9 را قرار دهیم مانند :

```
cout << "\u00A9";
```

```
T-
```

برای مشاهده لیست مقادیر مبنای ۱۶ برای کاراکترهای یونیکد به لینک زیر مراجعه نمایید :

<http://www.asci.cl/htmlcodes.htm>

اگر کامپایلر به یک کاراکتر کنترلی غیر مجاز برخورد کند، برنامه پیغام خطا می‌دهد. بیشترین خطا زمانی اتفاق می‌افتد که برنامه نویس برای چاپ اسلش ( \ ) از \ \ استفاده می‌کند. برای دریافت اطلاعات بیشتر در مورد کاراکترهای کنترلی به لینک زیر مراجعه کنید :

<https://msdn.microsoft.com/en-us/library/h21280bw.aspx>

## متغیر

متغیر مکانی از حافظه است که شما می‌توانید مقداری را در آن ذخیره کنید. می‌توان آن را به عنوان یک ظرف تصور کرد که داده‌های خود را در آن قرار داده‌اید. محتویات این ظرف می‌تواند پاک شود یا تغییر کند. هر متغیر دارای یک نام نیز هست. که از طریق آن می‌توان متغیر را از دیگر متغیرها تشخیص داد و به مقدار آن دسترسی پیدا کرد. همچنین دارای یک مقدار می‌باشد که می‌تواند توسط کاربر انتخاب شده باشد یا نتیجه یک محاسبه باشد. مقدار متغیر می‌تواند تهی نیز باشد. متغیر دارای نوع نیز هست بدین معنی که نوع آن با نوع داده ای که در آن ذخیره می‌شود یکی است. متغیر دارای عمر نیز هست که از روی آن می‌توان تشخیص داد که متغیر باید چقدر در طول برنامه مورد استفاده قرار گیرد. و در نهایت متغیر دارای محدوده استفاده نیز هست که به شما می‌گوید که متغیر در چه جای برنامه برای شما قابل دسترسی است. ما از متغیرها به عنوان یک انبار موقتی برای ذخیره داده استفاده می‌کنیم. هنگامی که یک برنامه ایجاد می‌کنیم احتیاج به یک مکان برای ذخیره داده، مقادیر یا داده‌هایی که توسط کاربر وارد می‌شوند داریم. ایم مکان همان متغیر است. برای این از کلمه متغیر استفاده می‌شود چون ما می‌توانیم بسته به نوع شرایط هر جا که لازم باشد مقدار آن را تغییر دهیم. متغیرها موقتی هستند و فقط موقعی مورد استفاده قرار می‌گیرند که برنامه در حال اجراست و وقتی شما برنامه را می‌بندید محتویات متغیرها نیز پاک می‌شود. قبلاً ذکر شد که به وسیله نام متغیر می‌توان به آن دسترسی پیدا کرد. برای نامگذاری متغیرها باید قوانین زیر را رعایت کرد :

- نام متغیر باید با یک از حروف الفبا (a-z or A-Z) شروع شود.
- نمی‌تواند شامل کاراکترهای غیرمجاز مانند \$, ^, ?, # باشد.

- نمی‌توان از کلمات رزرو شده در C++ برای نام متغیر استفاده کرد.
- نام متغیر نباید دارای فضای خالی (spaces) باشد.
- اسامی متغیرها نسبت به بزرگی و کوچکی حروف حساس هستند. در C++ دو حرف مانند `a` و `A` دو کاراکتر مختلف به حساب می‌آیند.

دو متغیر با نامهای `myNumber` و `MyNumber` دو متغیر مختلف محسوب می‌شوند چون یکی از آن‌ها با حرف کوچک `m` و دیگری با حرف بزرگ `M` شروع می‌شود. شما نمی‌توانید دو متغیر را که دقیق شبیه هم هستند را در یک `scope` (محدوده) تعریف کنید. `Scope` به معنای یک بلوک کد است که متغیر در آن قابل دسترسی و استفاده است. در مورد `Scope` در فصل‌های آینده بیشتر توضیح خواهیم داد. متغیر دارای نوع هست که نوع داده ای را که در خود ذخیره می‌کند را نشان می‌دهد. معمول‌ترین انواع داده `int`, `double`, `string`, `char`, `float` می‌باشند. برای مثال شما برای قرار دادن یک عدد صحیح در متغیر باید از نوع `int` استفاده کنید.

## انواع ساده

انواع ساده انواعی از داده‌ها هستند که شامل اعداد، کاراکترها و رشته‌ها و مقادیر بولی می‌باشند. به انواع ساده انواع اصلی نیز گفته می‌شود چون از آن‌ها برای ساخت انواع پیچیده تری مانند کلاس‌ها و ساختارها استفاده می‌شود. انواع ساده دارای مجموعه مشخصی از مقادیر هستند و محدوده خاصی از اعداد را در خود ذخیره می‌کنند. در C++ هفت نوع داده وجود دارد که در جدول زیر ذکر شده‌اند:

کلمه کلیدی	نوع
<code>bool</code>	<code>Boolean</code>
<code>char</code>	<code>Character</code>
<code>int</code>	<code>Integer</code>
<code>float</code>	<code>Floating point</code>
<code>double</code>	<code>Double floating point</code>
<code>void</code>	<code>Valueless</code>
<code>wchar_t</code>	<code>Wide character</code>



انواع بالا ( به جز void ) می‌توانند با عباراتی مثل signed ، long ، unsigned و short ترکیب شده و نوع‌های دیگری را به وجود آورند :

نوع	مقدار فضایی که از حافظه اشغال می‌کند	محدوده
char	1byte	-128 to 127 or 0 to 255
unsigned char	1byte	0 to 255
signed char	1byte	-128 to 127
int	4bytes	-2147483648 to 2147483647
unsigned int	4bytes	0 to 4294967295
signed int	4bytes	-2147483648 to 2147483647
short int	2bytes	-32768 to 32767
unsigned short int	2bytes	0 to 65,535
signed short int	2bytes	-32768 to 32767
long int	8bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
signed long int	8bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
unsigned long int	8bytes	0 to 18,446,744,073,709,551,615
float	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	8bytes	+/- 1.7e +/- 308 (~15 digits)
wchar_t	2 or 4 bytes	1 wide character

نوع char برای ذخیره کاراکترهای یونیکد استفاده می‌شود. کاراکترها باید داخل یک کوتیشن ساده قرار بگیرند مانند. ('a')

نوع bool فقط می‌تواند مقادیر درست (true) یا نادرست (false) را در خود ذخیره کند و بیشتر در برنامه‌هایی که دارای ساختار تصمیم‌گیری هستند مورد استفاده قرار می‌گیرد.

نوع `string` برای ذخیره گروهی از کاراکترها مانند یک پیغام استفاده می‌شود. مقادیر ذخیره شده در یک رشته باید داخل دابل کوتیشن قرار گیرند تا توسط کامپایلر به عنوان یک رشته در نظر گرفته شوند. مانند ("message")

## استفاده از متغیرها

در مثال زیر نحوه تعریف و مقدار دهی متغیرها نمایش داده شده است:

```
#include <iostream>
using namespace std;

int main()
{
    //Declare variables
    int num1;
    int num2;
    double num3;
    double num4;
    bool boolVal;
    char myChar;
    string message;

    //Assign values to variables
    num1 = 1;
    num2 = 2;
    num3 = 3.54;
    num4 = 4.12;
    boolVal = true;
    myChar = 'R';
    message = "Hello World!";

    //Show the values of the variables
    cout << "num1 = " << num1 << endl;
    cout << "num2 = " << num2 << endl;
    cout << "num3 = " << num3 << endl;
    cout << "num4 = " << num4 << endl;
    cout << "boolVal = " << boolVal << endl;
    cout << "myChar = " << myChar << endl;
    cout << "message = " << message << endl;
}
```

## تعریف متغیر

در کد بالا متغیرهایی با نوع و نام متفاوت تعریف شده‌اند. ابتدا باید نوع داده‌هایی را که این متغیرها قرار است در خود ذخیره کنند را مشخص کنیم و سپس یک نام برای آن‌ها در نظر بگیریم و در آخر سیمیکولن بگذاریم. همیشه به یاد داشته باشید که قبل از مقدار دهی و استفاده از متغیر باید آن را تعریف کرد. شاید برایتان این سؤال پیش آمده باشد که کاربرد endl چیست؟ endl برای ایجاد خط جدید مورد استفاده قرار گرفته است. یعنی نشانگر ماوس را همانند کاراکتر کنترلی \n به خط بعد می‌برد، در نتیجه خروجی کد بالا در خطوط جداگانه چاپ می‌شود .

```
//Declare variables
int num1;
int num2;
double num3;
double num4;
bool boolVal;
char myChar;
string message;
```

نحوه تعریف متغیر به صورت زیر است :

```
data_type identifier;
```

date\_type همان نوع داده است مانند int ، double و .... dentifier نیز نام متغیر است که به ما امکان استفاده و دسترسی به مقدار متغیر را می‌دهد . برای تعریف چند متغیر از یک نوع می‌توان به صورت زیر عمل کرد:

```
data_type identifier1, identifier2, ... indentifierN;
```

مثال

```
int num1, num2, num3, num4, num5;
string message1, message2, message3;
```

در مثال بالا ۵ متغیر از نوع صحیح و ۳ متغیر از نوع رشته تعریف شده است. توجه داشته باشید که بین متغیرها باید علامت کاما ( , ) باشد .

## نامگذاری متغیرها

- نام متغیر باید با یک حرف یا زیرخط و به دنبال آن حرف یا عدد شروع شود .
- نمی‌توان از کاراکترهای خاص مانند & , % , # یا عدد برای شروع نام متغیر استفاده کرد مانند 2numbers .

- نام متغیر نباید دارای فاصله باشد. برای نام های چند حرفی می توان به جای فاصله از علامت زیرخط یا \_ استفاده کرد.

نامهای مجاز:

```
num1 myNumber studentCount total first_name _minimum
num2 myChar average amountDue last_name _maximum
name counter sum isLeapYear color_of_car _age
```

نامهای غیر مجاز:

```
123 #numbers# #ofstudents 1abc2
123abc $money first name ty.np
my number this&that last name 1:00
```

اگر به نامهای مجاز در مثال بالا توجه کنید متوجه قراردادهای به کار رفته در نامگذاری آنها خواهید شد. یکی از روش های نامگذاری، نامگذاری کوهان شتری است. در این روش که برای متغیرهای دو کلمه ای به کار می رود، اولین حرف اولین کلمه با حرف کوچک و اولین حرف دومین کلمه با حرف بزرگ نمایش داده می شود مانند: myNumber. توجه کنید که اولین حرف کلمه Number با حرف بزرگ شروع شده است. مثال دیگر کلمه numberOfStudents است. اگر توجه کنید بعد از اولین کلمه حرف اول سایر کلمات با حروف بزرگ نمایش داده شده است.

## محدوده متغیر

متغیرهای ابتدای درس در داخل متد main() تعریف شده اند. در نتیجه این متغیرها فقط در داخل متد main() قابل دسترسی هستند. محدوده یک متغیر مشخص می کند که متغیر در کجای کد قابل دسترسی است. هنگامیکه برنامه به پایان متد main() می رسد متغیرها از محدوده خارج و بدون استفاده می شوند تا زمانی که برنامه در حال اجراست. محدوده متغیرها انواعی دارد که در درس های بعدی با آنها آشنا می شوید. تشخیص محدوده متغیر بسیار مهم است چون به وسیله آن می فهمید که در کجای کد می توان از متغیر استفاده کرد. باید یاد آور شد که دو متغیر در یک محدوده نمی توانند دارای نام یکسان باشند. مثلاً کد زیر در برنامه ایجاد خطا می کند:

```
int num1;
int num1;
```

از آنجاییکه C++ به بزرگی و کوچکی بودن حروف حساس است می توان از این خاصیت برای تعریف چند متغیر هم نام ولی با حروف متفاوت (از لحاظ بزرگی و کوچکی) برای تعریف چند متغیر از یک نوع استفاده کرد مانند:

```
int num1;
int Num1;
int NUM1;
```

## مقداردهی متغیرها

می‌توان فوراً بعد از تعریف متغیرها مقادیری را به آن‌ها اختصاص داد. این عمل را مقداردهی می‌نامند. در زیر نحوه مقدار دهی متغیرها نشان داده شده است:

```
data_type identifier = value;
```

به عنوان مثال:

```
int myNumber = 7;
```

همچنین می‌توان چندین متغیر را فقط با گذاشتن کاما بین آن‌ها به سادگی مقدار دهی کرد:

```
data_type variable1 = value1, variable2 = value2, ... variableN, valueN;
int num1 = 1, num2 = 2, num3 = 3;
```

تعریف متغیر با مقدار دهی متغیرها متفاوت است. تعریف متغیر یعنی انتخاب نوع و نام برای متغیر ولی مقدار دهی یعنی اختصاص یک مقدار به متغیر.

## اختصاص مقدار به متغیر

در زیر نحوه اختصاص مقادیر به متغیرها نشان داده شده است:

```
num1 = 1;
num2 = 2;
num3 = 3.54;
num4 = 4.12;
boolVal = true;
myChar = 'R';
message = "Hello World!";
```

به این نکته توجه کنید که شما به مغیری که هنوز تعریف نشده نمی‌توانید مقدار بدهید. شما فقط می‌توانید از متغیرهایی استفاده کنید که هم تعریف و هم مقدار دهی شده باشند. مثلاً متغیرهای بالا همه قابل استفاده هستند. در این مثال num1 و num2 هر دو

تعریف شده‌اند و مقادیری از نوع صحیح به آن‌ها اختصاص داده شده است. اگر نوع داده با نوع متغیر یکی نباشد برنامه پیغام خطا می‌دهد.

## ثابت

ثابت‌ها انواعی هستند که مقدار آن‌ها در طول برنامه تغییر نمی‌کند. ثابت‌ها حتماً باید مقدار دهی اولیه شوند و اگر مقدار دهی آن‌ها فراموش شود در برنامه خطا به وجود می‌آید. بعد از این که به ثابت‌ها مقدار اولیه اختصاص داده شد هرگز در زمان اجرای برنامه نمی‌توان آن را تغییر داد. برای تعریف ثابت‌ها باید از کلمه کلیدی `const` و `#define` استفاده کرد. معمولاً نام ثابت‌ها را طبق قرارداد با حروف بزرگ می‌نویسند تا تشخیص آن‌ها در برنامه راحت باشد. نحوه تعریف ثابت در زیر آمده است:

```
const data_type identifier = initial_value;
```

در کد بالا ابتدا کلمه کلیدی `const` و سپس نوع ثابت و بعد نام ثابت را با حروف بزرگ می‌نویسیم. و در نهایت یک مقدار را به آن اختصاص می‌دهیم و علامت سمیکالن می‌گذاریم.

```
#define data_type identifier initial_value
```

در روش بالا فقط `#define` را نوشته و سپس نام ثابت و بعد مقداری که قرار است دریافت کند. به این نکته توجه کنید که در روش بالا نه علامت سمیکالن وجود دارد و نه علامت مساوی. مثال:

```
#include <iostream>
using namespace std;

int main()
{
    const int NUMBER = 1;

    NUMBER = 20; //ERROR, Cant modify a constant

    cout << NUMBER;
}
```

```
#include <iostream>
using namespace std;

int main()
{
    #define NUMBER 1

    NUMBER = 20; //ERROR, Cant modify a constant

    cout << NUMBER;
```

```
}
```

در این مثال می بینید که مقدار دادن به یک ثابت ، که قبلاً مقدار دهی شده برنامه را با خطا مواجه می کند. نکته دیگری که نباید فراموش شود این است که نباید مقدار ثابت را با مقدار دیگر متغیرهای تعریف شده در برنامه برابر قرار داد. مثال :

```
int someVariable;
const int MY_CONST = someVariable;
```

ممکن است این سؤال برایتان پیش آمده باشد که دلیل استفاده از ثابت ها چیست؟ اگر مطمئن هستید که مقداری در برنامه وجود دارند که هرگز در طول برنامه تغییر نمی کنند بهتر است که آن ها را به صورت ثابت تعریف کنید. این کار هر چند کوچک کیفیت برنامه شما را بالا می برد.

## عبارات و عملگرها

ابتدا با دو کلمه آشنا شوید :

- عملگر : نمادهایی هستند که اعمال خاص انجام می دهند.
- عملوند : مقداری که عملگرها بر روی آن ها عملی انجام می دهند.

مثلاً  $X+Y$  : یک عبارت است که در آن  $X$  و  $Y$  عملوند و علامت  $+$  عملگر به حساب می آیند.

زبانهای برنامه نویسی جدید دارای عملگرهایی هستند که از اجزاء معمول زبان به حساب می آیند  $C++$ . دارای عملگرهای مختلفی از جمله عملگرهای ریاضی، تخصیصی، مقایسه ای، منطقی و بیتی می باشد. از عملگرهای ساده ریاضی می توان به عملگر جمع و

تفریق اشاره کرد. سه نوع عملگر در  $C++$  وجود دارد :

- یگانی - (Unary) به یک عملوند نیاز دارد
- دودویی - (Binary) به دو عملوند نیاز دارد
- سه تایی - (Ternary) به سه عملوند نیاز دارد

انواع مختلف عملگر که در این بخش مورد بحث قرار می گیرند عبارت اند از :

- عملگرهای ریاضی
- عملگرهای تخصیصی

- عملگرهای مقایسه ای
- عملگرهای منطقی
- عملگرهای بیتی
- عملگرهای ریاضی

## عملگرهای ریاضی

C++ از عملگرهای ریاضی برای انجام محاسبات استفاده می‌کند. جدول زیر عملگرهای ریاضی سی پلاس پلاس را نشان می‌دهد:

عملگر	دسته	مثال	نتیجه
+	Binary	<code>var1 = var2 + var3;</code>	Var1 برابر است با حاصل جمع var2 و var3
-	Binary	<code>var1 = var2 - var3;</code>	Var1 برابر است با حاصل تفریق var2 و var3
*	Binary	<code>var1 = var2 * var3;</code>	Var1 برابر است با حاصل ضرب var2 در var3
/	Binary	<code>var1 = var2 / var3;</code>	Var1 برابر است با حاصل تقسیم var2 بر var3
%	Binary	<code>var1 = var2 % var3;</code>	Var1 برابر است با باقیمانده تقسیم var2 و var3
+	Unary	<code>var1 = +var2;</code>	Var1 برابر است با مقدار var2
-	Unary	<code>var1 = -var2;</code>	Var1 برابر است با مقدار var2 ضربدر ۱ -

مثال بالا در از نوع عددی استفاده شده است. اما استفاده از عملگرهای ریاضی برای نوع رشته ای نتیجه متفاوتی دارد. همچنین در جمع دو کاراکتر کامپایلر معادل عددی آن‌ها را نشان می‌دهد. دیگر عملگرهای C++ عملگرهای کاهش و افزایش هستند. این عملگرها مقدار ۱ را از متغیرها کم یا به آن‌ها اضافه می‌کنند. از این متغیرها اغلب در حلقه‌ها استفاده می‌شود:

عملگر	دسته	مثال	نتیجه
++	Unary	<code>var1 = ++var2;</code>	مقدار var1 برابر است با var2 بعلاوه ۱
--	Unary	<code>var1 = --var2;</code>	مقدار var1 برابر است با var2 منهای ۱



var2 با مقدار برابر است به متغیر var2 یک واحد اضافه می‌شود	var1 = var2++;	Unary	++
var2 با مقدار برابر است از متغیر var2 یک واحد کم می‌شود	var1 = var2--;	Unary	-

به این نکته توجه داشته باشید که محل قرار گیری عملگر در نتیجه محاسبات تأثیر دارد. اگر عملگر قبل از متغیر var2 بیاید افزایش یا کاهش var1 اتفاق می‌افتد. چنانچه عملگرها بعد از متغیر var2 قرار بگیرند ابتدا var1 برابر var2 می‌شود و سپس متغیر var2 افزایش یا کاهش می‌یابد. به مثال‌های زیر توجه کنید:

```
#include <iostream>
using namespace std;

int main()
{
    int x = 0;
    int y = 1;

    x = ++y;

    cout << "x= {0}" << x << endl;
    cout << "y= {0}" << y << endl;
}
```

```
x=2
y=2
```

```
#include <iostream>
using namespace std;

int main()
{
    int x = 0;
    int y = 1;

    x = --y;

    cout << "x= {0}" << x << endl;
    cout << "y= {0}" << y << endl;
}
```

```
x=0
y=0
```

همانطور که در دو مثال بالا مشاهده می‌کنید، درج عملگرهای ++ و -- قبل از عملوند y باعث می‌شود که ابتدا یک واحد از y کم و یا یک واحد به y اضافه شود و سپس نتیجه در عملوند x قرار بگیرد. حال به دو مثال زیر توجه کنید:

```
#include <iostream>
using namespace std;

int main()
{
    int x = 0;
    int y = 1;

    x = y--;

    cout << "x= {0}" << x << endl;
    cout << "y= {0}" << y << endl;
}
```

```
x=1
y=0
```

```
#include <iostream>
using namespace std;

int main()
{
    int x = 0;
    int y = 1;

    x = y++;

    cout << "x= {0}" << x << endl;
    cout << "y= {0}" << y << endl;
}
```

```
x=1
y=2
```

همانطور که در دو مثال بالا مشاهده می‌کنید، درج عملگرهای ++ و -- بعد از عملوند y باعث می‌شود که ابتدا مقدار y در داخل متغیر x قرار بگیرد و سپس یک واحد از y کم و یا یک واحد به آن اضافه شود. حال می‌توانیم با ایجاد یک برنامه نحوه عملکرد عملگرهای ریاضی در C++ را یاد بگیریم:

```
#include <iostream>
using namespace std;

int main()
{
    //Variable declarations
    int num1, num2;
    string msg1, msg2;
```

```
//Assign test values
num1 = 6;
num2 = 3;

//Demonstrate use of mathematical operators
cout << "The sum of num1 and num2 is " << (num1 + num2) << endl;
cout << "The difference of num1 and num2 is " << (num1 - num2) << endl;
cout << "The product of num1 and num2 is " << (num1 * num2) << endl;
cout << "The quotient of num1 and num2 is " << (num1 / num2) << endl;
cout << "The remainder of num1 and num2 is " << (num1 % num2) << endl;

msg1 = "Hello ";
msg2 = "World!";
cout << msg1 + msg2;
}
```

```
The sum of 6 and 3 is 9.
The difference of 6 and 3 is 3.
The product of 6 and 3 is 18.
The quotient of 6 and 3 is 2.
The remainder of 6 divided by 3 is 0
Hello World!
```

برنامه بالا نتیجه هر عبارت را نشان می‌دهد. در این برنامه از `endl` برای نشان دادن نتایج در سطریهای متفاوت استفاده شده است. `C++` خط جدید و فاصله و فضای خالی را نادیده می‌گیرد. در خط ۲۲ مشاهده می‌کنید که دو رشته به وسیله عملگر `+` به هم متصل شده‌اند. نتیجه استفاده از عملگر `+` برای چسباندن دو کلمه `"Hello "` و `"World!"` رشته `"Hello World!"` خواهد بود. به فاصله‌های خالی بعد از اولین کلمه توجه کنید اگر آن‌ها را حذف کنید از خروجی برنامه نیز حذف می‌شوند.

## عملگرهای تخصیصی

نوع دیگر از عملگرهای `C++` عملگرهای جایگزینی نام دارند. این عملگرها مقدار متغیر سمت راست خود را در متغیر سمت چپ قرار می‌دهند. جدول زیر انواع عملگرهای تخصیصی در سی شارپ را نشان می‌دهد:

عملگر	مثال	نتیجه
=	<code>var1 = var2;</code>	مقدار <code>var1</code> برابر است با مقدار <code>var2</code>
+=	<code>var1 += var2;</code>	مقدار <code>var1</code> برابر است با حاصل جمع <code>var1</code> و <code>var2</code>
-=	<code>var1 -= var2;</code>	مقدار <code>var1</code> برابر است با حاصل تفریق <code>var1</code> و <code>var2</code>
*=	<code>var1 *= var2;</code>	مقدار <code>var1</code> برابر است با حاصل ضرب <code>var1</code> در <code>var2</code>

مقدار var1 برابر است با حاصل تقسیم var1 بر var2	var1 /= var2;	/=
مقدار var1 برابر است با باقیمانده تقسیم var1 بر var2	var1 %= var2;	%=

استفاده از این نوع عملگرها در واقع یک نوع خلاصه نویسی در کد است. مثلاً شکل اصلی کد `var1 += var2` به صورت `var1 = var1 + var2` می باشد. این حالت کدنویسی زمانی کارایی خود را نشان می دهد که نام متغیرها طولانی باشد. برنامه زیر چگونگی استفاده از عملگرهای تخصیصی و تأثیر آنها را بر متغیرها نشان می دهد.

```
#include <iostream>
using namespace std;

int main()
{
    int number;

    cout << "Assigning 10 to number..." << endl;
    number = 10;
    cout << "Number = " << number << endl;

    cout << "Adding 10 to number..." << endl;
    number += 10;
    cout << "Number = " << number << endl;

    cout << "Subtracting 10 from number..." << endl;
    number -= 10;
    cout << "Number = " << number << endl;
}
```

```
Assigning 10 to number...
Number = 10
Adding 10 to number...
Number = 20
Subtracting 10 from number...
Number = 10
```

در برنامه از ۳ عملگر تخصیصی استفاده شده است. ابتدا یک متغیر و مقدار 10 با استفاده از عملگر `=` به آن اختصاص داده شده است. سپس به آن با استفاده از عملگر `+=` مقدار ۱۰ اضافه و در آخر به وسیله عملگر `-=` عدد ۱۰ از آن کم شده است.

## عملگرهای مقایسه ای

از عملگرهای مقایسه ای برای مقایسه مقادیر استفاده می شود. نتیجه این مقادیر یک مقدار بولی (منطقی) است. این عملگرها اگر نتیجه مقایسه دو مقدار درست باشد مقدار ۱ و اگر نتیجه مقایسه اشتباه باشد مقدار ۰ را نشان می دهند. این عملگرها به طور

معمول در دستورات شرطی به کار می‌روند به این ترتیب که باعث ادامه یا توقف دستور شرطی می‌شوند. جدول زیر عملگرهای مقایسه ای در C++ را نشان می‌دهد:

عملگر	دسته	مثال	نتیجه
==	Binary	var1 = var2 == var3	var1 در صورتی ۱ است که مقدار var2 با مقدار var3 برابر باشد در غیر اینصورت ۰ است
!=	Binary	var1 = var2 != var3	var1 در صورتی ۱ است که مقدار var2 با مقدار var3 برابر نباشد در غیر اینصورت ۰ است
<	Binary	var1 = var2 < var3	var1 در صورتی ۱ است که مقدار var2 کوچک‌تر از var3 مقدار باشد در غیر اینصورت ۰ است
>	Binary	var1 = var2 > var3	var1 در صورتی ۱ است که مقدار var2 بزرگ‌تر از مقدار var3 باشد در غیر اینصورت ۰ است
<=	Binary	var1 = var2 <= var3	var1 در صورتی ۱ است که مقدار var2 کوچک‌تر یا مساوی مقدار var3 باشد در غیر اینصورت ۰ است
>=	Binary	var1 = var2 >= var3	var1 در صورتی ۱ است که مقدار var2 بزرگ‌تر یا مساوی var3 مقدار باشد در غیر اینصورت ۰ است

برنامه زیر نحوه عملکرد این عملگرها را نشان می‌دهد:

```
#include <iostream>
using namespace std;

int main()
{
    int num1 = 10;
    int num2 = 5;

    cout << num1 << " == " << num2 << " : " << (num1 == num2) << endl;
    cout << num1 << " != " << num2 << " : " << (num1 != num2) << endl;
    cout << num1 << " < " << num2 << " : " << (num1 < num2) << endl;
    cout << num1 << " > " << num2 << " : " << (num1 > num2) << endl;
    cout << num1 << " <= " << num2 << " : " << (num1 <= num2) << endl;
    cout << num1 << " >= " << num2 << " : " << (num1 >= num2) << endl;
}
```

```
10 == 5 : 0
```

```
10 != 5 : 1
10 < 5 : 0
10 > 5 : 1
10 <= 5 : 0
10 >= 5 : 1
```

در مثال بالا ابتدا دو متغیر را که می‌خواهیم با هم مقایسه کنیم را ایجاد کرده و به آن‌ها مقادیری اختصاص می‌دهیم. سپس با استفاده از یک عملگر مقایسه ای آن‌ها را با هم مقایسه کرده و نتیجه را چاپ می‌کنیم. به این نکته توجه کنید که هنگام مقایسه دو متغیر از عملگر == به جای عملگر = باید استفاده شود. عملگر = عملگر تخصیصی است و در عبارتی مانند  $x = y$  مقدار  $y$  را در  $x$  اختصاص می‌دهد. عملگر == عملگر مقایسه ای است که دو مقدار را با هم مقایسه می‌کند مانند  $x == y$  و اینطور خوانده می‌شود  $x$  برابر است با  $y$ .

## عملگرهای منطقی

عملگرهای منطقی بر روی عبارات منطقی عمل می‌کنند و نتیجه آن‌ها نیز یک مقدار بولی است. از این عملگرها اغلب برای شرطهای پیچیده استفاده می‌شود. همانطور که قبلاً یاد گرفتید مقادیر بولی می‌توانند `true` یا `false` باشند. فرض کنید که `var2` و `var3` دو مقدار بولی هستند.

عملگر	نام	دسته	مثال
<code>&amp;&amp;</code>	منطقی AND	Binary	<code>var1 = var2 &amp;&amp; var3;</code>
<code>  </code>	منطقی OR	Binary	<code>var1 = var2    var3;</code>
<code>!</code>	منطقی NOT	Unary	<code>var1 = !var1;</code>

## عملگر منطقی (&&) AND

اگر مقادیر دو طرف عملگر AND ، `true` باشند عملگر AND مقدار `true` را بر می‌گرداند. در غیر اینصورت اگر یکی از مقادیر یا هر دوی آن‌ها `false` باشند مقدار `false` را بر می‌گرداند. در زیر جدول درستی عملگر AND نشان داده شده است:

X	Y	X && Y
true	true	true
true	false	false

false	true	false
false	false	false

برای درک بهتر تأثیر عملگر AND یاد آوری می‌کنم که این عملگر فقط در صورتی مقدار true را نشان می‌دهد که هر دو عملوند مقدارشان true باشد. در غیر این صورت نتیجه تمام ترکیب‌های بعدی false خواهد شد. استفاده از عملگر AND مانند استفاده از عملگرهای مقایسه ای است. به عنوان مثال نتیجه عبارت زیر درست (true) است اگر سن (age) بزرگ‌تر از ۱۸ و salary کوچک‌تر از ۱۰۰۰ باشد.

```
result = (age > 18) && (salary < 1000);
```

عملگر AND زمانی کارآمد است که ما با محدود خاصی از اعداد سرو کار داریم. مثلاً عبارت  $10 \leq x \leq 100$  بدین معنی است که x می‌تواند مقداری شامل اعداد ۱۰ تا ۱۰۰ را بگیرد. حال برای انتخاب اعداد خارج از این محدوده می‌توان از عملگر منطقی AND به صورت زیر استفاده کرد.

```
inRange = (number <= 10) && (number >= 100);
```

## عملگر منطقی (||) OR

اگر یکی یا هر دو مقدار دو طرف عملگر OR، درست (true) باشد، عملگر OR مقدار true را بر می‌گرداند. جدول درستی عملگر OR در زیر نشان داده شده است :

X	Y	X    Y
true	true	true
true	false	true
false	true	true
false	false	false

در جدول بالا مشاهده می‌کنید که عملگر OR در صورتی مقدار false را بر می‌گرداند که مقادیر دو طرف آن false باشند. کد زیر را در نظر بگیرید. نتیجه این کد در صورتی درست (true) است که رتبه نهایی دانش آموز (finalGrade) بزرگ‌تر از ۷۵ یا یا نمره نهایی امتحان آن ۱۰۰ باشد.

```
isPassed = (finalGrade >= 75) || (finalExam == 100);
```

برخلاف دو اپراتور OR و AND عملگر منطقی NOT یک عملگر یگانی است و فقط به یک عملوند نیاز دارد. این عملگر یک مقدار یا اصطلاح بولی را نفی می‌کند. مثلاً اگر عبارت یا مقدار true باشد آنرا false و اگر false باشد آنرا true می‌کند. جدول زیر عملکرد اپراتور NOT را نشان می‌دهد:

X	!X
true	false
false	true

نتیجه کد زیر در صورتی درست است که (age سن) بزرگ‌تر یا مساوی ۱۸ نباشد.

```
isMinor = !(age >= 18);
```

شما سیستم باینری و نحوه تبدیل اعداد دهدهی به باینری را از لینک زیر یاد بگیرید :

<http://www.w3-farsi.com/?p=5698>

در سیستم باینری (دودویی) که کامپیوتر از آن استفاده می‌کند وضعیت هر چیز یا خاموش است یا روشن. برای نشان دادن حالت روشن از عدد ۱ و برای نشان دادن حالت خاموش از عدد ۰ استفاده می‌شود. بنابراین اعداد باینری فقط می‌توانند صفر یا یک باشند. اعداد باینری را اعداد در مبنای ۲ و اعداد اعشاری را اعداد در مبنای ۱۰ می‌گویند. یک بیت نشان دهنده یک رقم باینری است و هر بیت نشان دهنده ۸ بیت است. به عنوان مثال برای یک داده از نوع `int` به ۳۲ بیت یا ۴ بایت فضا برای ذخیره آن نیاز داریم، این بدین معناست که اعداد از ۳۲ رقم ۰ و ۱ برای ذخیره استفاده می‌کنند. برای مثال عدد ۱۰۰ وقتی به عنوان یک متغیر از نوع `int` ذخیره می‌شود در کامپیوتر به صورت زیر خوانده می‌شود:

[illegible]

عدد ۱۰۰ در مبنای ده معادل عدد ۱۱۰۰۱۰۰ در مبنای ۲ است. در اینجا ۷ رقم سمت راست نشان دهنده عدد ۱۰۰ در مبنای ۲ است و مابقی صفرهای سمت راست برای پر کردن بیت‌هایی است که عدد از نوع `int` نیاز دارد. به این نکته توجه کنید که اعداد باینری از سمت راست به چپ خوانده می‌شوند. عملگرهای بیتی `C++` در جدول زیر نشان داده شده‌اند:



عملگر	نام	دسته	مثال
&	بیتی AND	Binary	<code>x = y &amp; z;</code>
	بیتی OR	Binary	<code>x = y   z;</code>
^	بیتی XOR	Binary	<code>x = y ^ z;</code>
~	بیتی NOT	Unary	<code>x = ~y;</code>
&=	بیتی AND Assignment	Binary	<code>x &amp;= y;</code>
=	بیتی OR Assignment	Binary	<code>x  = y;</code>
^=	بیتی XOR Assignment	Binary	<code>x ^= y;</code>

## عملگر بیتی AND(&)

عملگر بیتی AND کاری شبیه عملگر منطقی AND انجام می‌دهد با این تفاوت که این عملگر بر روی بیت‌ها کار می‌کند. اگر مقادیر دو طرف آن ۱ باشد مقدار ۱ را بر می‌گرداند و اگر یکی یا هر دو طرف آن صفر باشد مقدار صفر را بر می‌گرداند. جدول درستی عملگر بیتی AND در زیر آمده است:

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

در زیر نحوه استفاده از عملگر بیتی AND آمده است:

```
int result = 5 & 3;
cout << result;
```

1

[illegible]

### عملگر پیتی OR (|)

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

```
int result = 7 | 9;

cout << result;
```

15

```

7: 00000000000000000000000000000000111
9: 000000000000000000000000000000001001
-----
15: 000000000000000000000000000000001111

```

با استفاده از جدول درستی عملگر بیتی OR می‌توان نتیجه استفاده از این عملگر را تشخیص داد. عدد ۱۱۱۱ باینری معادل عدد ۱۵ صحیح است.

## عملگریختی (XOR (^))

جدول درستی این عملگر در زیر آمده است:

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

در صورتیکه عملوندهای دو طرف این عملگر هر دو صفر یا هر دو یک باشند نتیجه صفر در غیر اینصورت نتیجه یک می‌شود. در مثال زیر تأثیر عملگر بیتی XOR را بر روی دو مقدار مشاهده می‌کنید:

```
int result = 5 ^ 7;

cout << result;
```

2

در زیر معادل پاییری اعداد بالا (۵ و ۷) نشان داده شده است.

[illegible]

با نگاه کردن به جدول درستی عملگر بیتی XOR ، می توان فهمید که چرا نتیجه عدد ۲ می شود.

## عملگر پیتی (~) NOT

این عملگر یک عملگر یگانی است و فقط به یک عملوند نیاز دارد. در زیر جدول درستی این عملگر آمده است:

X	NOT X
1	0

عملگر بتی NOT مقادیر بیت‌ها را معکوس می‌کند. در زیر چگونگی استفاده از این عملگر آمده است:

-8

```
7: 0000000000000000000000000000000000111  
-----  
-8: 1111111111111111111111111111111111000
```

این نوع عملگرها به شما اجازه می‌دهند که بیت‌ها را به سمت چپ یا راست جا به جا کنید. دو نوع عملگر بیتی تغییر مکان وجود دارد که هر کدام دو عملوند قبول می‌کنند. عملوند سمت چپ این عملگرها حالت باینری یک مقدار و عملوند سمت راست تعداد جابه جایی بیت‌ها را نشان می‌دهد.

این عملگر بیت‌های عملوند سمت چپ را به تعداد  $n$  مکان مشخص شده توسط عملوند سمت راست، به سمت چپ منتقل می‌کند.

به عنوان مثال:

40

در مثال بالا ما بیت‌های مقدار ۱۰ را دو مکان به سمت چپ منتقل کرده‌ایم، حال بیا ببینیم تأثیر این انتقال را بررسی کنیم:

اگر ما حق تقدم عملگرها را رعایت نکنیم و عبارت بالا را از سمت چپ به راست انجام دهیم نتیجه ۹ خواهد شد ( $3+2=5$  سپس  $5+4=9$ ) اما کامپایلر با توجه به تقدم عملگرها محاسبات را انجام می‌دهد. برای مثال عمل ضرب و تقسیم نسبت به

جمع و تفریق تقدم دارند. بنابراین در مثال فوق ابتدا عدد ۲ ضربدر ۳ و سپس نتیجه آن ها تقسیم بر ۱ می شود که نتیجه ۶ به دست می آید. در آخر عدد ۶ با ۱ جمع می شود و عدد ۷ حاصل می شود. در جدول زیر تقدم عملگرهای C++ از بالا به پایین آمده است:

Level	Precedence group	Operator	Grouping
1	Scope	::	Left-to-right
2	Postfix (unary)	++ -	Left-to-right
		()	
		[]	
		. ->	
3	Prefix (unary)	++ -	Right-to-left
		~ !	
		+ -	
		& *	
		new delete	
		sizeof	
		(type)	
4	Pointer-to-member	.* ->*	Left-to-right
5	Aritdmetic: scaling	* / %	Left-to-right
6	Aritdmetic: addition	+ -	Left-to-right
7	Bitwise shift	<< >>	Left-to-right
8	Relational	< > <= >=	Left-to-right
9	Equality	== !=	Left-to-right
10	And	&	Left-to-right
11	Exclusive or	^	Left-to-right
12	Inclusive or		Left-to-right
13	Conjunction	&&	Left-to-right
14	Disjunction		Left-to-right
15	Assignment-level expressions	= *= /= %= += -= >>= <<= &= ^=  =	Right-to-left

		?:	
16	Sequencing	,	Left-to-right

ابتدا عملگرهای با بالاترین و سپس عملگرهای با پایین‌ترین حق تقدم در محاسبات تأثیر می‌گذارند. به این نکته توجه کنید که تقدم عملگرها ++و - به مکان قرارگیری آن‌ها بستگی دارد (در سمت چپ یا راست عملوند باشند). به عنوان مثال:

```
int number = 3;
number1 = 3 + ++number; //results to 7
number2 = 3 + number++; //results to 6
```

در عبارت اول ابتدا به مقدار number یک واحد اضافه شده و ۴ می‌شود و سپس مقدار جدید با عدد ۳ جمع می‌شود و در نهایت عدد ۷ به دست می‌آید. در عبارت دوم مقدار عددی ۳ به مقدار number اضافه می‌شود و عدد ۶ به دست می‌آید. سپس این مقدار در متغیر number2 قرار می‌گیرد. و در نهایت مقدار number به ۴ افزایش می‌یابد. برای ایجاد خوانایی در تقدم عملگرها و انجام محاسباتی که در آن‌ها از عملگرهای زیادی استفاده می‌شود از پرانتز استفاده می‌کنیم:

```
number = ( 1 + 2 ) * ( 3 / 4 ) % ( 5 - ( 6 * 7 ) );
```

در مثال بالا ابتدا هر کدام از عباراتی که داخل پرانتز هستند مورد محاسبه قرار می‌گیرند. به نکته ای در مورد عبارتی که در داخل پرانتز سوم قرار دارد توجه کنید. در این عبارت ابتدا مقدار داخلی‌ترین پرانتز مورد محاسبه قرار می‌گیرد یعنی مقدار ۶ ضربدر ۷ شده و سپس از ۵ کم می‌شود. اگر دو یا چند عملگر با حق تقدم یکسان موجود باشد ابتدا باید هر کدام از عملگرها را که در ابتدای عبارت می‌آیند مورد ارزیابی قرار دهید. به عنوان مثال:

```
number = 3 * 2 + 8 / 4;
```

هر دو عملگر \* و / دارای حق تقدم یکسانی هستند. بنابر این شما باید از چپ به راست آن‌ها را در محاسبات تأثیر دهید. یعنی ابتدا ۳ را ضربدر ۲ می‌کنید و سپس عدد ۸ را بر ۴ تقسیم می‌کنید. در نهایت نتیجه دو عبارت را جمع کرده و در متغیر number قرار می‌دهید.

## گرفتن ورودی از کاربر

سی پلاس پلاس دارای تعدادی شیء و متد برای گرفتن ورودی از کاربر می‌باشد. حال می‌خواهیم درباره cin یکی دیگر از اشیاء کلاس Istraem بحث کنیم که یک مقدار را از کاربر دریافت می‌کند. کار cin این است که تمام کاراکترهایی را که شما در محیط کنسول تایپ می‌کنید تا زمانی که دکمه Enter را می‌زنید می‌خواند. به برنامه زیر توجه کنید:

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
    string name;
    int age;
    double height;

    cout << "Enter your name: ";
    cin >> name;
    cout << "Enter your age: ";
    cin >> age;
    cout << "Enter your height: ";
    cin >> height;

    //Print a blank line
    cout << endl;

    //Show the details you typed
    cout << "Name is " << name << endl;
    cout << "Age is " << age << endl;
    cout << "Height is " << height << endl;
}
```

```
Enter your name: John
Enter your age: 18
Enter your height: 160.5

Name is John.
Age is 18.
Height is 160.5.
```

ابتدا ۳ متغیر را برای ذخیره داده در برنامه تعریف می‌کنیم (خطوط ۸ و ۹ و ۱۰). برنامه از کاربر می‌خواهد که نام خود را وارد کند (خط ۱۲). در خط ۱۳ شما به عنوان کاربر نام خود را وارد می‌کنید. مقدار متغیر نام، برابر مقداری است که توسط `cin` خوانده می‌شود. از آنجاییکه نام از نوع رشته است باید کتابخانه مربوط به رشته‌ها را در ابتدای برنامه وارد کنیم:

```
#include <string>
```

سپس برنامه از ما سن را سؤال می‌کند (خط ۱۴). آن را در خط ۱۵ وارد کرده و در نهایت در خط ۱۶ و ۱۷ هم قد را وارد می‌کنیم.

## ساختارهای تصمیم



تقریباً همه زبانهای برنامه نویسی به شما اجازه اجرای کد را در شرایط مطمئن می‌دهند. حال تصور کنید که یک برنامه دارای ساختار تصمیم‌گیری نباشد و همه کدها را اجرا کند. این حالت شاید فقط برای چاپ یک پیغام در صفحه مناسب باشد ولی فرض کنید که شما بخواهید اگر مقدار یک متغیر با یک عدد برابر باشد سپس یک پیغام چاپ شود آن وقت با مشکل مواجه خواهید شد C++ . راه‌های مختلفی برای رفع این نوع مشکلات ارائه می‌دهد. در این بخش با مطالب زیر آشنا خواهید شد :

- دستور if
- دستور if...else
- عملگر سه تایی
- دستور if چندگانه
- دستور if تو در تو
- عملگرهای منطقی
- دستور switch

## دستور if

می‌توان با استفاده از دستور if و یک شرط خاص که باعث ایجاد یک کد می‌شود یک منطق به برنامه خود اضافه کنید. دستور if ساده‌ترین دستور شرطی است که برنامه می‌گوید اگر شرطی برقرار است کد معینی را انجام بده. ساختار دستور if به صورت زیر است :

```
if (condition)
    code to execute;
```

قبل از اجرای دستور if ابتدا شرط بررسی می‌شود. اگر شرط برقرار باشد یعنی درست باشد سپس کد اجرا می‌شود. شرط یک عبارت مقایسه‌ای است. می‌توان از عملگرهای مقایسه‌ای برای تست درست یا اشتباه بودن شرط استفاده کرد. اجازه بدهید که نگاهی به نحوه استفاده از دستور if در داخل برنامه بیندازیم . برنامه زیر پیغام Hello World را اگر مقدار number کمتر از ۱۰ و Goodbye World را اگر مقدار number از ۱۰ بزرگ‌تر باشد در صفحه نمایش می‌دهد.

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      //Declare a variable and set it a value less than 10
```

```

7      int number = 5;
8
9      //If the value of number is less than 10
10     if (number < 10)
11         cout << "Hello World." << endl;
12
13     //Change the value of a number to a value which is greater than 10
14     number = 15;
15
16     //If the value of number is greater than 10
17     if (number > 10)
18         cout << "Goodbye World.";
19 }

```

```

Hello World.
Goodbye World.

```

در خط ۷ یک متغیر با نام `number` تعریف و مقدار ۵ به آن اختصاص داده شده است. وقتی به اولین دستور `if` در خط ۱۰ می‌رسیم برنامه تشخیص می‌دهد که مقدار `number` از ۱۰ کمتر است یعنی ۵ کوچک‌تر از ۱۰ است.

منطقی است که نتیجه مقایسه درست می‌باشد بنابراین دستور `if` دستور را اجرا می‌کند (خط ۱۱) و پیغام `Hello World` چاپ می‌شود. حال مقدار `number` را به ۱۵ تغییر می‌دهیم (خط ۱۴). وقتی به دومین دستور `if` در خط ۱۷ می‌رسیم برنامه مقدار `number` را با ۱۰ مقایسه می‌کند و چون مقدار `number` یعنی ۱۵ از ۱۰ بزرگ‌تر است برنامه پیغام `Goodbye World` را چاپ می‌کند (خط ۱۸). به این نکته توجه کنید که دستور `if` را می‌توان در یک خط نوشت:

```
if (number > 10) cout << "Goodbye World.";
```

شما می‌توانید چندین دستور را در داخل دستور `if` بنویسید. کافایت که از یک آکولاد برای نشان دادن ابتدا و انتهای دستورات استفاده کنید. همه دستورات داخل بین آکولاد جز بدنه دستور `if` هستند. نحوه تعریف چند دستور در داخل بدنه `if` به صورت زیر است:

```

if (condition)
{
    statement1;
    statement2;
    .
    .
    .
    statementN;
}

```

این هم یک مثال ساده:

```

if (x > 10)
{
    cout << "x is greater than 10." << endl;
}

```

```
    cout << "This is still part of the if statement.";
}
```

در مثال بالا اگر مقدار  $x$  از ۱۰ بزرگتر باشد دو پیغام چاپ می‌شود. حال اگر به عنوان مثال آکولاد را حذف کنیم و مقدار  $x$  از ۱۰ بزرگتر نباشد مانند کد زیر:

```
if (x > 10)
cout << "x is greater than 10." << endl;
cout << "This is still part of the if statement. (Really?)";
```

کد بالا در صورتی بهتر خوانده می‌شود که بین دستورات فاصله بگذاریم.

```
if (x > 10)
cout << "x is greater than 10." << endl;

cout << "This is still part of the if statement. (Really?)";
```

می‌بیند که دستور دوم (خط ۳) در مثال بالا جز دستور `if` نیست. اینجاست که چون ما فرض را بر این گذاشته‌ایم که مقدار  $x$  از ۱۰ کوچکتر است پس خط `"This is still part of the if statement. (Really?)"` چاپ می‌شود. در نتیجه اهمیت وجود آکولاد مشخص می‌شود. به عنوان تمرین همیشه حتی اگر فقط یک دستور در بدنه `if` داشتید برای آن یک آکولاد بگذارید. فراموش نکنید که از قلم انداختن یک آکولاد باعث به وجود آمدن خطا شده و یافتن آن را سخت می‌کند. مثالی دیگر در مورد دستور: `if`

```
#include <iostream>
using namespace std;

int main()
{
    int firstNumber;
    int secondNumber;

    cout << "Enter a number: ";
    cin >> firstNumber;

    cout << "Enter another number: ";
    cin >> secondNumber;

    if (firstNumber == secondNumber)
    {
        cout << firstNumber << " == " << secondNumber << endl;
    }
    if (firstNumber != secondNumber)
    {
        cout << firstNumber << " != " << secondNumber << endl;
    }
    if (firstNumber < secondNumber)
    {
```

```

    cout << firstNumber << " < " << secondNumber << endl;
}
if (firstNumber > secondNumber)
{
    cout << firstNumber << " > " << secondNumber << endl;
}
if (firstNumber <= secondNumber)
{
    cout << firstNumber << "<= " << secondNumber << endl;
}
if (firstNumber >= secondNumber)
{
    cout << firstNumber << ">= " << secondNumber << endl;
}
}

```

```

Enter a number: 2
Enter another number: 5
2 != 5
2 < 5
2 <= 5
Enter a number: 10
Enter another number: 3
10 != 3
10 > 3
10 >= 3
Enter a number: 5
Enter another number: 5
5 == 5
5 <= 5
5 >= 5

```

ما از عملگرهای مقایسه ای در دستور `if` استفاده کرده ایم. ابتدا دو عدد که قرار است با هم مقایسه شوند را به عنوان ورودی از کاربر می گیریم. اعداد با هم مقایسه می شوند و اگر شرط درست بود پیغامی چاپ می شود. به این نکته توجه داشته باشید که شرطها مقادیر بولی هستند، بنابراین شما می توانید نتیجه یک عبارت را در داخل یک متغیر بولی ذخیره کنید و سپس از متغیر به عنوان شرط در دستور `if` استفاده کنید. اگر مقدار `year` برابر ۲۰۰۰ باشد سپس حاصل عبارت در متغیر `isNewMillenium` ذخیره می شود. می توان از متغیر برای تشخیص کد اجرایی بدنه دستور `if` استفاده کرد خواه مقدار متغیر درست باشد یا نادرست.

```

bool isNewMillenium = year == 2000;

if (isNewMillenium)
{
    cout << "Happy New Millenium!";
}

```

## دستور if...else

دستور if فقط برای اجرای یک حالت خاص به کار می‌رود یعنی اگر حالتی برقرار بود کار خاصی انجام شود. اما زمانی که شما بخواهید اگر شرط خاصی برقرار شد یک دستور و اگر برقرار نبود دستور دیگر اجرا شود باید از دستور if else استفاده کنید. ساختار دستور if else در زیر آمده است:

```
if (condition)
{
    code to execute if condition is true;
}
else
{
    code to execute if condition is false;
}
```

از کلمه کلیدی else نمی‌توان به تنهایی استفاده کرد بلکه حتماً باید با if به کار برده شود. اگر فقط یک کد اجرایی در داخل بدنه if و بدنه else دارید استفاده از آکولاد اختیاری است. کد داخل بلوک else فقط در صورتی اجرا می‌شود که شرط داخل دستور if نادرست باشد. در زیر نحوه استفاده از دستور if...else آمده است.

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int number = 5;
7
8      //Test the condition
9      if (number < 10)
10     {
11         cout << "The number is less than 10." << endl;
12     }
13     else
14     {
15         cout << "The number is either greater than or equal to 10." << endl;
16     }
17
18     //Modify value of number
19     number = 15;
20
21     //Repeat the test to yield a different result
22     if (number < 10)
23     {
24         cout << "The number is less than 10." << endl;
25     }
26     else
27     {
28         cout << "The number is either greater than or equal to 10." << endl;
29     }
30 }
```

در خط ۶ یک متغیر به نام `number` تعریف کرده ایم و در خط ۹ تست می کنیم که آیا مقدار متغیر `number` از ۱۰ کمتر است یا نه و چون کمتر است در نتیجه کد داخل بلوک `if` اجرا می شود (خط ۱۱) و اگر مقدار `number` را تغییر دهیم و به مقداری بزرگتر از ۱۰ تغییر دهیم (خط ۱۹)، شرط نادرست می شود (خط ۲۲) و کد داخل بلوک `else` اجرا می شود (خط ۲۸).

## عملگر شرطی

عملگر شرطی (`?:`) در `C++` مانند دستور شرطی `if...else` عمل می کند. در زیر نحوه استفاده از این عملگر آمده است:

```
<condition> ? <result if true> : <result if false>
```

عملگر شرطی تنها عملگر سه تایی `C++` است که نیاز به سه عملوند دارد، شرط، یک مقدار زمانی که شرط درست باشد و یک مقدار زمانی که شرط نادرست باشد. اجازه بدهید که نحوه استفاده از این عملگر را در داخل برنامه مورد بررسی قرار دهیم.

```
#include <iostream>
#include <String>
using namespace std;

int main()
{
    string pet1 = "puppy";
    string pet2 = "kitten";
    string type1;
    string type2;

    type1 = (pet1 == "puppy") ? "dog" : "cat";
    type2 = (pet2 == "kitten") ? "cat" : "dog";

    cout << type1 << endl;
    cout << type2;
}
```

```
dog
cat
```

برنامه بالا نحوه استفاده از این عملگر شرطی را نشان می دهد. خط یک به صورت زیر ترجمه می شود: اگر مقدار `pet1` برابر با `puppy` سپس مقدار `dog` را در `type1` قرار بده در غیر این صورت مقدار `cat` را `type1` قرار بده. خط دو به صورت زیر ترجمه می شود: اگر مقدار `pet2` برابر با `kitten` سپس مقدار `cat` را در `type2` قرار بده در غیر این صورت مقدار `dog`. حال برنامه بالا را با استفاده از دستور `if else` می نویسیم:

```
if (pet1 == "puppy")
    type1 = "dog";
else
```

```
type1 = "cat";
```

هنگامی که چندین دستور در داخل یک بلوک if یا else دارید از عملگر شرطی استفاده نکنید چون خوانایی برنامه را پایین می‌آورد.

## دستور if چندگانه

اگر بخواهید چند شرط را بررسی کنید چکار می‌کنید؟ می‌توانید از چندین دستور if استفاده کنید و بهتر است که این دستورات if را به صورت زیر بنویسید:

```
if (condition)
{
    code to execute;
}
else
{
    if (condition)
    {
        code to execute;
    }
    else
    {
        if (condition)
        {
            code to execute;
        }
        else
        {
            code to execute;
        }
    }
}
```

خواندن کد بالا سخت است. بهتر است دستورات را به صورت تو رفتگی در داخل بلوک else بنویسید. می‌توانید کد بالا را ساده تر کنید:

```
if (condition)
{
    code to execute;
}
else if (condition)
{
    code to execute;
}
else if (condition)
{
    code to execute;
}
```

```
else
{
    code to execute;
}
```

حال که نحوه استفاده از دستور `else if` را یاد گرفتید باید بدانید که مانند `else if` ، `else if` نیز به دستور `if` وابسته است. دستور `else if` وقتی اجرا می شود که اولین دستور `if` اشتباه باشد. حال اگر `else if` اشتباه باشد دستور `else if` بعدی اجرا می شود. و اگر آن نیز اجرا نشود در نهایت دستور `else` اجرا می شود. برنامه زیر نحوه استفاده از دستور `if else` را نشان می دهد:

```
#include <iostream>
#include <String>
using namespace std;

int main()
{
    int choice;

    cout << "What's your favorite color?" << endl;
    cout << "[1] Black" << endl;
    cout << "[2] White" << endl;
    cout << "[3] Blue" << endl;
    cout << "[4] Red" << endl;
    cout << "[5] Yellow" << endl;

    cout << "Enter your choice: ";
    cin >> choice;

    if (choice == 1)
    {
        cout << "You might like my black t-shirt." << endl;
    }
    else if (choice == 2)
    {
        cout << "You might be a clean and tidy person." << endl;
    }
    else if (choice == 3)
    {
        cout << "You might be sad today." << endl;
    }
    else if (choice == 4)
    {
        cout << "You might be inlove right now." << endl;
    }
    else if (choice == 5)
    {
        cout << "Lemon might be your favorite fruit." << endl;
    }
    else
    {
        cout << "Sorry, your favorite color is not in the choices above." << endl;
    }
}
```



```
}
```

```
What's your favorite color?
```

```
[1] Black
```

```
[2] White
```

```
[3] Blue
```

```
[4] Red
```

```
[5] Yellow
```

```
Enter your choice: 1
```

```
You might like my black t-shirt.
```

```
What's your favorite color?
```

```
[1] Black
```

```
[2] White
```

```
[3] Blue
```

```
[4] Red
```

```
[5] Yellow
```

```
Enter your choice: 999
```

```
Sorry, your favorite color is not in the choices above.
```

خروجی برنامه بالا به متغیر choice وابسته است. بسته به اینکه شما چه چیزی انتخاب می‌کنید پیغام‌های مختلفی چاپ می‌شود. اگر عددی که شما تایپ می‌کنید در داخل حالت‌های انتخاب نباشد کد مربوط به بلوک else اجرا می‌شود.

## دستور if تو در تو

می‌توان از دستور if تو در تو در C++ استفاده کرد. یک دستور ساده if در داخل دستور if دیگر.

```
if (condition)
{
    code to execute;

    if (condition)
    {
        code to execute;
    }
    else if (condition)
    {
        if (condition)
        {
            code to execute;
        }
    }
}
else
{
    if (condition)
    {
        code to execute;
    }
}
```

اجازه بدهید که نحوه استفاده از دستور if تو در تو را نشان دهیم:

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main()
6  {
7      int age;
8      string gender;
9
10     cout << "Enter your age: ";
11     cin >> age;
12
13     cout << "Enter your gender (male/female): ";
14     cin >> gender;
15
16     if (age > 12)
17     {
18         if (age < 20)
19         {
20             if (gender == "male")
21             {
22                 cout << "You are a teenage boy." << endl;
23             }
24             else
25             {
26                 cout << "You are a teenage girl." << endl;
27             }
28         }
29         else
30         {
31             cout << "You are already an adult." << endl;
32         }
33     }
34     else
35     {
36         cout << "You are still too young." << endl;
37     }
38 }
```

```
Enter your age: 18
Enter your gender: male
You are a teenage boy.
Enter your age: 12
Enter your gender: female
You are still too young.
```

اجازه بدهید که برنامه را کالبد شکافی کنیم. ابتدا برنامه از شما درباره سنتان سؤال می‌کند (خط ۱۰). در خط ۱۳ درباره جنستان از شما سؤال می‌کند. سپس به اولین دستور `if` می‌رسد (خط ۱۶). در این قسمت اگر سن شما بیشتر از ۱۲ سال باشد برنامه وارد بدنه دستور `if` می‌شود در غیر اینصورت وارد بلوک `else` (خط ۳۴) مربوط به همین دستور `if` می‌شود.

حال فرض کنیم که سن شما بیشتر از ۱۲ سال است و شما وارد بدنه اولین `if` شده‌اید. در بدنه اولین `if` دو دستور `if` دیگر را مشاهده می‌کنید. اگر سن کمتر ۲۰ باشد شما وارد بدنه `if` دوم می‌شوید و اگر نباشد به قسمت `else` متناظر با آن می‌روید (خط

(۲۹). دوباره فرض می‌کنیم که سن شما کمتر از ۲۰ باشد، در اینصورت وارد بدنه if دوم شده و با یک if دیگر مواجه می‌شوید (خط ۲۰). در اینجا جنسیت شما مورد بررسی قرار می‌گیرد که اگر برابر "male" باشد، کدهای داخل بدنه سومین if اجرا می‌شود در غیر اینصورت قسمت else مربوط به این if اجرا می‌شود (خط ۲۴). پیشنهاد می‌شود که از if تو در تو در برنامه کمتر استفاده کنید چون خوانایی برنامه را پایین می‌آورد.

## استفاده از عملگرهای منطقی

عملگرهای منطقی به شما اجازه می‌دهند که چندین شرط را با هم ترکیب کنید. این عملگرها حداقل دو شرط را در گیر می‌کنند و در آخر یک مقدار بولی را بر می‌گردانند. در جدول زیر برخی از عملگرهای منطقی آمده است :

عملگر	تلفظ	مثال	تأثیر
&&	And	$z = (x > 2) \&\& (y < 10)$	مقدار Z در صورتی true است که هر دو شرط دو طرف عملگر مقدارشان true باشد. اگر فقط مقدار یکی از شروط false باشد مقدار false ، z خواهد شد .
	Or	$z = (x > 2)    (y < 10)$	مقدار Z در صورتی true است که یکی از دو شرط دو طرف عملگر مقدارشان true باشد. اگر هر دو شرط مقدارشان false باشد مقدار false ، z خواهد شد .
!	Not	$z = !(x > 2)$	مقدار Z در صورتی true است که مقدار شرط false باشد و در صورتی false است که مقدار شرط true باشد .

به عنوان مثال جمله  $z = (x > 2) \&\& (y < 10)$  را به این صورت بخوانید : "در صورتی مقدار z برابر true است که مقدار x بزرگ‌تر از ۲ و مقدار y کوچک‌تر از ۱۰ باشد در غیر اینصورت false است". این جمله بدین معناست که برای اینکه مقدار کل دستور true باشد باید مقدار همه شروط true باشد. عملگر منطقی (||) تأثیر متفاوتی نسبت به عملگر منطقی AND (&&) دارد. نتیجه عملگر منطقی OR برابر true است اگر فقط مقدار یکی از شروط true باشد. و اگر مقدار هیچ یک از شروط

true نباشد نتیجه false خواهد شد. می‌توان عملگرهای منطقی AND و OR را با هم ترکیب کرده و در یک عبارت به کار برد مانند:

```
if ((x == 1) && ((y > 3) || z < 10)) {  
    //do something here  
}
```

در اینجا استفاده از پرانتز مهم است چون از آن در گروه بندی شرطها استفاده می‌کنیم. در اینجا ابتدا عبارت (z < 10) || (y > 3) را بررسی قرار می‌گیرد. (به علت تقدم عملگرها) سپس نتیجه آن بوسیله عملگر AND با نتیجه (x == 1) مقایسه می‌شود. حال بیا ببینیم نحوه استفاده از عملگرهای منطقی در برنامه را مورد بررسی قرار دهیم:

```
#include <iostream>  
#include <String>  
using namespace std;  
  
int main()  
{  
    int age;  
    string gender;  
  
    cout << "Enter your age: ";  
    cin >> age;  
  
    cout << "Enter your gender (male/female): ";  
    cin >> gender;  
  
    if (age > 12 && age < 20)  
    {  
        if (gender == "male")  
        {  
            cout << "You are a teenage boy." << endl;  
        }  
        else  
        {  
            cout << "You are a teenage girl." << endl;  
        }  
    }  
    else  
    {  
        cout << "You are still too young." << endl;  
    }  
}
```

```
Enter your age: 18  
Enter your gender (male/female): female  
You are a teenage girl.  
Enter you age: 10  
Enter your gender (male/female): male  
You are not a teenager.
```

برنامه بالا نحوه استفاده از عملگر منطقی AND را نشان می‌دهد (خط ۱۶). وقتی به دستور if می‌رسید (خط ۱۶) برنامه سن شما را چک می‌کند. اگر سن شما بزرگ‌تر از ۱۲ و کوچک‌تر از ۲۰ باشد (سنن بین ۱۲ و ۲۰ باشد) یعنی مقدار هر دو true باشد سپس کدهای داخل بلوک if اجرا می‌شوند. اگر نتیجه یکی از شروط false باشد کدهای داخل بلوک else اجرا می‌شود. عملگر AND عملوند سمت چپ را مورد بررسی قرار می‌دهد. اگر مقدار آن false باشد دیگر عملوند سمت راست را بررسی نمی‌کند و مقدار false را بر می‌گرداند. بر عکس عملگر || عملوند سمت چپ را مورد بررسی قرار می‌دهد و اگر مقدار آن true باشد سپس عملوند سمت راست را نادیده می‌گیرد و مقدار true را بر می‌گرداند.

```
if (x == 2 & y == 3)
{
    //Some code here
}

if (x == 2 | y == 3)
{
    //Some code here
}
```

نکته مهم اینجاست که شما می‌توانید از عملگرهای & و | به عنوان عملگر بیتی استفاده کنید. تفاوت جزئی این عملگرها وقتی که به عنوان عملگر بیتی به کار می‌روند این است که دو عملوند را بدون در نظر گرفتن مقدار عملوند سمت چپ مورد بررسی قرار می‌دهند. به عنوان مثال حتی اگر مقدار عملوند سمت چپ false باشد عملوند سمت چپ به وسیله عملگر بیتی (&) AND ارزیابی می‌شود. اگر شرطها را در برنامه ترکیب کنید استفاده از عملگرهای منطقی (&&) AND و (||) OR به جای عملگرهای بیتی (&) AND و (|) OR بهتر خواهد بود. یکی دیگر از عملگرهای منطقی عملگر (!) NOT است که نتیجه یک عبارت را خنثی یا منفی می‌کند. به مثال زیر توجه کنید:

```
if (!(x == 2))
{
    cout << "x is not equal to 2.";
}
```

اگر نتیجه عبارت x == 2 برابر false باشد عملگر ! آن را True می‌کند.

## دستور Switch

در C++ ساختاری به نام switch وجود دارد که به شما اجازه می‌دهد که با توجه به مقدار ثابت یک متغیر چندین انتخاب داشته باشید. دستور switch معادل دستور if تو در تو است با این تفاوت که در دستور switch متغیر فقط مقادیر ثابتی از اعداد،

رشته‌ها و یا کاراکترها را قبول می‌کند. مقادیر ثابت مقداری هستند که قابل تغییر نیستند. در زیر نحوه استفاده از دستور switch آمده است:

```
switch (testVar)
{
case compareVa11:
    code to execute if testVar == compareVa11;
    break;
case compareVa12:
    code to execute if testVar == compareVa12;
    break;
.
.
.
case compareVa1N:
    code to execute if testVer == compareVa1N;
    break;
default:
    code to execute if none of the values above match the testVar;
    break;
}
```

ابتدا یک مقدار در متغیر switch که در مثال بالا testVar است قرار می‌دهید. این مقدار با هر یک از عبارت‌های case داخل بلوک switch مقایسه می‌شود. اگر مقدار متغیر با هر یک از مقادیر موجود در دستورات case برابر بود کد مربوط به آن case اجرا خواهد شد. به این نکته توجه کنید که حتی اگر تعداد خط کدهای داخل دستور case از یکی بیشتر باشد نباید از آکولاد استفاده کنیم. آخر هر دستور case با کلمه کلیدی break تشخیص داده می‌شود که باعث می‌شود برنامه از دستور switch خارج شده و دستورات بعد از آن اجرا شوند. اگر این کلمه کلیدی از قلم بیوفتد برنامه با خطا مواجه می‌شود. دستور switch یک بخش default دارد. این دستور در صورتی اجرا می‌شود که مقدار متغیر با هیچ یک از مقادیر دستورات case برابر نباشد. دستور default اختیاری است و اگر از بدنه switch حذف شود هیچ اتفاقی نمی‌افتد. مکان این دستور هم مهم نیست اما بر طبق تعریف آن را در پایان دستورات می‌نویسند. به مثالی در مورد دستور switch توجه کنید:

```
#include <iostream>
#include <String>
using namespace std;

int main()
{
    int choice;

    cout << "What's your favorite pet?" << endl;
    cout << "[1] Dog" << endl;
    cout << "[2] Cat" << endl;
    cout << "[3] Rabbit" << endl;
    cout << "[4] Turtle" << endl;
    cout << "[5] Fish" << endl;
    cout << "[6] Not in the choices" << endl;
```

```

cout << "Enter your choice: " << endl;

cin >> choice;

switch (choice)
{
case 1:
    cout << "Your favorite pet is Dog." << endl;
    break;
case 2:
    cout << "Your favorite pet is Cat." << endl;
    break;
case 3:
    cout << "Your favorite pet is Rabbit." << endl;
    break;
case 4:
    cout << "Your favorite pet is Turtle." << endl;
    break;
case 5:
    cout << "Your favorite pet is Fish." << endl;
    break;
case 6:
    cout << "Your favorite pet is not in the choices." << endl;
    break;
default:
    cout << "You don't have a favorite pet." << endl;
    break;
}
}

```

```

What's your favorite pet?
[1] Dog
[2] Cat
[3] Rabbit
[4] Turtle
[5] Fish
[6] Not in the choices

```

```

Enter your choice: 2
Your favorite pet is Cat.
What's your favorite pet?
[1] Dog
[2] Cat
[3] Rabbit
[4] Turtle
[5] Fish
[6] Not in the choices

```

```

Enter your choice: 99
You don't have a favorite pet.

```

برنامه بالا به شما اجازه انتخاب حیوان مورد علاقه‌تان را می‌دهد. به اسم هر حیوان یک عدد نسبت داده شده است. شما عدد را وارد می‌کنید و این عدد در دستور switch با مقادیر case مقایسه می‌شود و با هر کدام از آن مقادیر که برابر بود پیغام مناسب نمایش داده خواهد شد. اگر هم با هیچ کدام از مقادیر case ها برابر نبود دستور default اجرا می‌شود. یکی دیگر از ویژگی‌های

دستور switch این است که شما می‌توانید از دو یا چند case برای نشان داده یک مجموعه کد استفاده کنید. در مثال زیر اگر مقدار number عدد ۱، ۲ یا ۳ باشد یک کد اجرا می‌شود. توجه کنید که case ها باید پشت سر هم نوشته شوند.

```
switch (number)
{
case 1:
case 2:
case 3:
    cout << "This code is shared by three values." << endl;
    break;
}
```

همانطور که قبلاً ذکر شد دستور switch معادل دستور if تو در تو است. برنامه بالا را به صورت زیر نیز می‌توان نوشت:

```
if (choice == 1)
cout << "Your favorite pet is Dog." << endl;
else if (choice == 2)
cout << "Your favorite pet is Cat." << endl;
else if (choice == 3)
cout << "Your favorite pet is Rabbit." << endl;
else if (choice == 4)
cout << "Your favorite pet is Turtle." << endl;
else if (choice == 5)
cout << "Your favorite pet is Fish." << endl;
else if (choice == 6)
cout << "Your favorite pet is not in the choices." << endl;
else
cout << "You don't have a favorite pet." << endl;
```

کد بالا دقیقاً نتیجه‌ای مانند دستور switch دارد. دستور default معادل دستور else می‌باشد. حال از بین این دو دستور (if else و switch) کدامیک را انتخاب کنیم. از دستور switch موقعی استفاده می‌کنیم که مقداری که می‌خواهیم با دیگر مقادیر مقایسه شود ثابت باشد. مثلاً در مثال زیر هیچگاه از switch استفاده نکنید.

```
int myNumber = 5;
int x = 5;

switch (myNumber)
{
case x:
    cout << "Error, you can't use variables as a value to be compared in a case statement.";
    break;
}
```

مشاهده می‌کنید که با اینکه مقدار x عدد ۵ است و به طور واضح با متغیر myNumber مقایسه شده است برنامه خطا می‌دهد چون x یک ثابت نیست بلکه یک متغیر است یا به زبان ساده‌تر، قابلیت تغییر را دارد. اگر بخواهید از x استفاده کنید و برنامه خطا ندهد باید از کلمه کلیدی const به صورت زیر استفاده کنید.



```
int myNumber = 5;
const int x = 5;

switch (myNumber)
{
case x:
    cout << "Error has been fixed!" << endl;
    break;
}
```

از کلمه کلیدی `const` برای ایجاد ثابت‌ها استفاده می‌شود. توجه کنید که بعد از تعریف یک ثابت نمی‌توان مقدار آن را در طول برنامه تغییر داد. به یاد داشته باشید که باید ثابت‌ها را حتماً مقداردهی کنید. دستور `switch` یک مقدار را با مقادیر `case` ها مقایسه می‌کند و شما لازم نیست که به شکل زیر مقادیر را با هم مقایسه کنید:

```
switch (myNumber)
{
case x > myNumber:
    cout << "switch statements can't test if a value is less than or greater than the other value.";
    break;
}
```

## تکرار

ساختارهای تکرار به شما اجازه می‌دهند که یک یا چند دستور کد را تا زمانی که یک شرط برقرار است تکرار کنید. بدون ساختارهای تکرار شما مجبورید همان تعداد کدها را بنویسید که بسیار خسته کننده است. مثلاً شما مجبورید ۱۰ بار جمله "Hello World." را تایپ کنید مانند مثال زیر:

```
cout << "Hello World." << endl;
cout << "Hello World." << endl;
cout << "Hello World." << endl;
cout << "Hello World." << endl;
cout << "Hello World." << endl;
cout << "Hello World." << endl;
cout << "Hello World." << endl;
cout << "Hello World." << endl;
cout << "Hello World." << endl;
cout << "Hello World." << endl;
```

البته شما می‌توانید با کپی کردن این تعداد کد را راحت بنویسید ولی این کار در کل کیفیت کدنویسی را پایین می‌آورد. راه بهتر برای نوشتن کدهای بالا استفاده از حلقه‌ها است. حلقه‌ها در C++ عبارت‌اند از:

- `while`

- do while •
- for •

## While حلقه

ابتدایی‌ترین ساختار تکرار در C++ حلقه While است. ابتدا یک شرط را مورد بررسی قرار می‌دهد و تا زمانی که شرط برقرار باشد کدهای درون بلوک اجرا می‌شوند. ساختار حلقه While به صورت زیر است:

```
while(condition)
{
    code to loop;
}
```

می‌بینید که ساختار While مانند ساختار if بسیار ساده است. ابتدا یک شرط را که نتیجه آن یک مقدار بولی است می‌نویسیم اگر نتیجه درست یا true باشد سپس کدهای داخل بلوک While اجرا می‌شوند. اگر شرط غلط یا false باشد وقتی که برنامه به حلقه While برسد هیچ‌کدام از کدها را اجرا نمی‌کند. برای متوقف شدن حلقه باید مقادیر داخل حلقه While اصلاح شوند. به یک متغیر شمارنده در داخل بدنه حلقه نیاز داریم. این شمارنده برای آزمایش شرط مورد استفاده قرار می‌گیرد و ادامه یا توقف حلقه به نوعی به آن وابسته است. این شمارنده را در داخل بدنه باید کاهش یا افزایش دهیم. در برنامه زیر نحوه استفاده از حلقه While آمده است:

```
#include <iostream>

using namespace std;

int main()
{
    int counter = 1;

    while (counter <= 10)
    {
        cout << "Hello World!" << endl;
        counter++;
    }
}
```

[illegible]

برنامه بالا ۱۰ بار پیغام Hello World! را چاپ می‌کند. اگر از حلقه در مثال بالا استفاده نمی‌کردیم مجبور بودیم تمام ۱۰ خط را تایپ کنیم. اجازه دهید که نگاهی به کدهای برنامه فوق ببینیم. ابتدا در خط ۷ یک متغیر تعریف و از آن به عنوان شمارنده حلقه استفاده شده است. سپس به آن مقدار ۱ را اختصاص می‌دهیم چون اگر مقدار نداشته باشد نمی‌توان در شرط از آن استفاده کرد.

در خط ۹ حلقه while را وارد می‌کنیم. در حلقه while ابتدا مقدار اولیه شمارنده با ۱۰ مقایسه می‌شود که آیا از ۱۰ کمتر است یا با آن برابر است. نتیجه هر بار مقایسه ورود به بدنه حلقه while و چاپ پیغام است. همانطور که مشاهده می‌کنید بعد از هر بار مقایسه مقدار شمارنده یک واحد اضافه می‌شود (خط ۱۲). حلقه تا زمانی تکرار می‌شود که مقدار شمارنده از ۱۰ کمتر باشد.

اگر مقدار شمارنده یک بماند و آن را افزایش ندهیم و یا مقدار شرط هرگز false نشود یک حلقه بین‌هایت به وجود می‌آید. به این نکته توجه کنید که در شرط بالا به جای علامت < از <= استفاده شده است. اگر از علامت < استفاده می‌کردیم کد ما ۹ بار تکرار می‌شد چون مقدار اولیه ۱ است و هنگامی که شرط به ۱۰ برسد false می‌شود چون  $10 < 10$  نیست. اگر می‌خواهید یک حلقه بی‌نهایت ایجاد کنید که هیچگاه متوقف نشود باید یک شرط ایجاد کنید که همواره درست (true) باشد.

```
while(true)
{
    //code to loop
}
```

این تکنیک در برخی موارد کارایی دارد و آن زمانی است که شما بخواهید با استفاده از دستورات break و return که در آینده توضیح خواهیم داد از حلقه خارج شوید.

## حلقه do while

حلقه do while یکی دیگر از ساختارهای تکرار است. این حلقه بسیار شبیه حلقه while است با این تفاوت که در این حلقه ابتدا کد اجرا می‌شود و سپس شرط مورد بررسی قرار می‌گیرد. ساختار حلقه do while به صورت زیر است:

```
do
{
    code to repeat;
} while (condition);
```

همانطور که مشاهده می‌کنید شرط در آخر ساختار قرار دارد. این بدین معنی است که کدهای داخل بدنه حداقل یکبار اجرا می‌شوند. برخلاف حلقه while که اگر شرط نادرست باشد دستورات داخل بدنه اجرا نمی‌شوند. برای اثبات این موضوع به کدهای زیر توجه کنید:

کنید :

```
int number = 1;
do
{
    cout << "Hello World!" << endl;
} while (number > 10);
```

```
Hello World!
```

با اجرای کد بالا، اول دستورات بلوک do اجرا می‌شوند و بعد مقدار number با عدد ۱۰ مقایسه می‌شود. در نتیجه حتی اگر شرط نادرست باشد باز هم قسمت do حداقل یک بار اجرا می‌شوند.

```
int number = 1;
while (number > 10)
{
    cout << "Hello World!" << endl;
}
```

اما در کد بالا چون اول مقدار number ابتدا مورد مقایسه قرار می‌گیرد، اگر شرط درست نباشد دیگر کدی اجرا نمی‌شود. یکی از موارد برتری استفاده از حلقه do while نسبت به حلقه while زمانی است که شما بخواهید اطلاعاتی از کاربر دریافت کنید. در دو کد زیر، یک عملیات یکسان توسط دو حلقه while و do while پیاده سازی شده است:

```
//while version

cout << "Enter a number greater than 10: " << endl;
cin >> number;

while (number < 10)
{
    cout << "Enter a number greater than 10: " << endl;
    cin >> number;
}
```

```
//do while version

do
{
    cout << "Enter a number greater than 10: " << endl;
    cin >> number;
} while (number < 10);
```

مشاهده می‌کنید که از کدهای کمتری در بدنه do while نسبت به while استفاده شده است.

## حلقه for

یکی دیگر از ساختارهای تکرار حلقه for است. این حلقه عملی شبیه به حلقه while انجام می‌دهد و فقط دارای چند خصوصیت اضافی است. ساختار حلقه for به صورت زیر است:

```
for(initialization; condition; operation)
{
    code to repeat;
}
```

مقدار دهی اولیه (initialization) اولین مقداری است که به شمارنده حلقه می‌دهیم. شمارنده فقط در داخل حلقه for قابل دسترسی است.

شرط (condition) در اینجا مقدار شمارنده را با یک مقدار دیگر مقایسه می‌کند و تعیین می‌کند که حلقه ادامه یابد یا نه.

عملگر (operation) که مقدار اولیه متغیر را کاهش یا افزایش می‌دهد.

در زیر یک مثال از حلقه for آمده است:

```
#include <iostream>
using namespace std;

int main()
{
    for (int i = 1; i <= 10; i++)
    {
        cout << "Number " << i << endl;
    }
}
```

```
Number 1
Number 2
Number 3
Number 4
Number 5
Number 6
Number 7
Number 8
Number 9
Number 10
```

برنامه بالا اعداد ۱ تا ۱۰ را با استفاده از حلقه for می‌شمارد. ابتدا یک متغیر به عنوان شمارنده تعریف می‌کنیم و آن را با مقدار ۱ مقدار دهی اولیه می‌کنیم. سپس با استفاده از شرط آن را با مقدار ۱۰ مقایسه می‌کنیم که آیا کمتر است یا مساوی؟ توجه کنید که قسمت سوم حلقه (i++) فوراً اجرا نمی‌شود. کد اجرا می‌شود و ابتدا رشته Number و سپس مقدار جاری i یعنی ۱ را چاپ می‌کند. آنگاه یک واحد به مقدار i اضافه شده و مقدار i برابر ۲ می‌شود و بار دیگر i با عدد ۱۰ مقایسه می‌شود و این حلقه تا

زمانی که مقدار شرط true شود ادامه می‌یابد. حال اگر بخواهید معکوس برنامه بالا را پیاده سازی کنید یعنی اعداد از بزرگ به کوچک چاپ شوند باید به صورت زیر عمل کنید:

```
for (int i = 10; i > 0; i--)
{
    //code omitted
}
```

کد بالا اعداد را از ۱۰ به ۱ چاپ می‌کند (از بزرگ به کوچک). مقدار اولیه شمارنده را ۱۰ می‌دهیم و با استفاده از عملگر کاهش (--). برنامه ای که شمارش معکوس را انجام می‌دهد ایجاد می‌کنیم. می‌توان قسمت شرط و عملگر را به صورت‌های دیگر نیز تغییر داد. به عنوان مثال می‌توان از عملگرهای منطقی در قسمت شرط و از عملگرهای تخصیصی در قسمت عملگر افزایش یا کاهش استفاده کرد. همچنین می‌توانید از چندین متغیر در ساختار حلقه for استفاده کنید.

```
for (int i = 1, y = 2; i < 10 && y > 20; i++, y -= 2)
{
    //some code here
}
```

به این نکته توجه کنید که اگر از چندین متغیر شمارنده یا عملگر در حلقه for استفاده می‌کنید باید آن‌ها را با استفاده از کاما از هم جدا کنید.

## حلقه‌های تو در تو (Nested Loops)

C++ به شما اجازه می‌دهد که از حلقه‌ها به صورت تو در تو استفاده کنید. اگر یک حلقه در داخل حلقه دیگر قرار بگیرد، به آن حلقه تو در تو گفته می‌شود. در این نوع حلقه‌ها، به ازای اجرای یک بار حلقه بیرونی، حلقه داخلی به طور کامل اجرا می‌شود. در زیر نحوه ایجاد حلقه تو در تو آمده است:

```
for (init; condition; increment)
{
    for (init; condition; increment)
    {
        //statement(s);
    }
    //statement(s);
}
```

```
while (condition)
{
```

```

while (condition)
{
    //statement(s);
}
//statement(s);
}

```

```

do
{
    //statement(s);
    do
    {
        //statement(s);
    } while (condition);
} while (condition);

```

نکته ای که در مورد حلقه‌های تو در تو وجود دارد این است که می‌توان از یک نوع حلقه در داخل نوع دیگر استفاده کرد. مثلاً می‌توان از حلقه for در داخل حلقه while استفاده نمود. در مثال زیر نحوه استفاده از این حلقه‌ها ذکر شده است. فرض کنید که می‌خواهید یک مستطیل با ۳ سطر و ۵ ستون ایجاد کنید:

```

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

```

```

* * * * *
* * * * *
* * * * *
* * * * *

```

در کد بالا به ازای یک بار اجرای حلقه for اول (خط ۶)، حلقه for دوم (۸-۱۱) به طور کامل اجرا می‌شود. یعنی وقتی مقدار i برابر عدد ۱ می‌شود، علامت \* توسط حلقه دوم ۵ بار چاپ می‌شود، وقتی i برابر ۲ می‌شود، دوباره علامت \* پنج بار چاپ می‌شود و ... . در کل منظور از دو حلقه for این است که در ۴ سطر علامت \* در ۵ ستون چاپ شود یا ۴ سطر ایجاد شود و در هر سطر ۵

بار علامت \* چاپ شود. خط ۱۲ هم برای ایجاد خط جدید است. یعنی وقتی حلقه داخلی به طور کامل اجرا شد، یک خط جدید ایجاد می شود و علامت های \* در خطوط جدید چاپ می شوند.

## خارج شدن از حلقه با استفاده از break و continue

گاهی اوقات با وجود درست بودن شرط می خواهیم حلقه متوقف شود. سؤال اینجاست که چطور این کار را انجام دهید؟ با استفاده از کلمه کلیدی break حلقه را متوقف کرده و با استفاده از کلمه کلیدی continue می توان بخشی از حلقه را رد کرد و به مرحله بعد رفت. برنامه زیر نحوه استفاده از break و continue را نشان می دهد:

```
#include <iostream>

using namespace std;

int main()
{
    cout << "Demonstrating the use of break" << endl;

    for (int x = 1; x < 10; x++)
    {
        if (x == 5)
            break;

        cout << "Number " << x << endl;
    }

    cout << endl;

    cout << "Demonstrating the use of continue." << endl;

    for (int x = 1; x < 10; x++)
    {
        if (x == 5)
            continue;

        cout << "Number " << x << endl;
    }
}
```

Demonstrating the use of break.

Number 1  
Number 2  
Number 3  
Number 4

Demonstrating the use of continue.

Number 1  
Number 2



```
Number 3
Number 4
Number 6
Number 7
Number 8
Number 9
```

در این برنامه از حلقه `for` برای نشان دادن کاربرد دو کلمه کلیدی فوق استفاده شده است اگر به جای `for` از حلقه های `while` و `do...while` استفاده می شد نتیجه یکسانی به دست می آمد. همانطور که در شرط برنامه (خط ۱۱) آمده است وقتی که مقدار `x` به عدد ۵ رسید سپس دستور `break` اجرا شود (خط ۱۲)

حلقه بلافاصله متوقف می شود حتی اگر شرط `x < 10` برقرار باشد. از طرف دیگر در خط ۲۴ حلقه `for` فقط برای یک تکرار خاص متوقف شده و سپس ادامه می یابد. (وقتی مقدار `x` برابر ۵ شود حلقه از ۵ رد شده و مقدار ۵ را چاپ نمی کند و بقیه مقادیر چاپ می شوند).

## آرایه ها

آرایه نوعی متغیر است که لیستی از آدرس های مجموعه ای از داده های هم نوع را در خود ذخیره می کند. تعریف چندین متغیر از یک نوع برای هدفی یکسان بسیار خسته کننده است. مثلاً اگر بخواهید صد متغیر از نوع اعداد صحیح تعریف کرده و از آن ها استفاده کنید. مطمئناً تعریف این همه متغیر بسیار کسالت آور و خسته کننده است. اما با استفاده از آرایه می توان همه آن ها را در یک خط تعریف کرد. در زیر راهی ساده برای تعریف یک آرایه نشان داده شده است:

```
datatype arrayName[length];
```

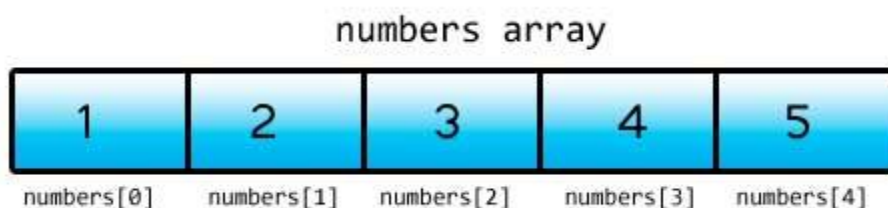
`Datatype` نوع داده هایی را نشان می دهد که آرایه در خود ذخیره می کند. کروه که بعد از نوع داده قرار می گیرد و نشان دهنده استفاده از آرایه است `length`. یا طول آرایه که به کامپایلر می گوید شما قصد دارید چه تعداد داده یا مقدار را در آرایه ذخیره کنید. `arrayName` که نام آرایه را نشان می دهد. هنگام نامگذاری آرایه بهتر است که نام آرایه نشان دهنده نوع آرایه باشد. به عنوان مثال برای نامگذاری آرایه ای که اعداد را در خود ذخیره می کند از کلمه `numbers` استفاده کنید. برای تعریف یک آرایه که ۵ مقدار از نوع اعداد صحیح در خود ذخیره می کند باید به صورت زیر عمل کنیم:

```
int numbers[5];
```

در این مثال ۵ آدرس از فضای حافظه کامپیوتر شما برای ذخیره ۵ مقدار رزرو می شود. حال چطور مقادیرمان را در هر یک از این آدرس ها ذخیره کنیم؟ برای دسترسی و اصلاح مقادیر آرایه از اندیس یا مکان آن ها استفاده می شود.

```
numbers[0] = 1;
numbers[1] = 2;
numbers[2] = 3;
numbers[3] = 4;
numbers[4] = 5;
```

اندیس یک آرایه از صفر شروع شده و به یک واحد کمتر از طول آرایه ختم می‌شود. به عنوان مثال شما یک آرایه ۵ عضوی دارید، اندیس آرایه از ۰ تا ۴ می‌باشد چون طول آرایه ۵ است پس ۵-۱ برابر است با ۴. این بدان معناست که اندیس ۰ نشان دهنده اولین عضو آرایه است و اندیس ۱ نشان دهنده دومین عضو و الی آخر. برای درک بهتر مثال بالا به شکل زیر توجه کنید:



به هر یک از اجزاء آرایه و اندیس‌های داخل گروه توجه کنید. کسانی که تازه شروع به برنامه نویسی کرده‌اند معمولاً در گذاشتن اندیس دچار اشتباه می‌شوند و مثلاً ممکن است در مثال بالا اندیس‌ها را از ۱ شروع کنند. یکی دیگر از راه‌های تعریف سریع و مقدار دهی یک آرایه به صورت زیر است:

```
datatype arrayName[length] = { val1, val2, ... valN };
```

ر این روش شما می‌توانید فوراً بعد از تعریف اندازه آرایه مقادیر را در داخل آکولاد قرار دهید. به یاد داشته باشید که هر کدام از مقادیر را با استفاده از کاما از هم جدا کنید. همچنین تعداد مقادیر داخل آکولاد باید با اندازه آرایه تعریف شده برابر باشد. به مثال زیر توجه کنید:

```
int numbers[5] = { 1, 2, 3, 4, 5 };
```

این مثال با مثال قبل هیچ تفاوتی ندارد و تعداد خط‌های کدنویسی را کاهش می‌دهد. شما می‌توانید با استفاده از اندیس به مقدار هر یک از اجزاء آرایه دسترسی یابید و آن‌ها را به دلخواه تغییر دهید. تعداد اجزاء آرایه در مثال بالا ۵ است و ما ۵ مقدار را در آن قرار می‌دهیم. اگر تعداد مقادیری که در آرایه قرار می‌دهیم کمتر یا بیشتر از طول آرایه باشد با خطا مواجه می‌شویم. یک راه بسیار ساده تر برای تعریف آرایه به صورت زیر است:

```
int numbers[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

به سادگی و بدون احتیاج ذکر طول می‌توان مقادیر را در داخل آکولاد قرار داد. کامپایلر به صورت اتوماتیک با شمارش مقادیر طول آرایه را تشخیص می‌دهد.

## دستیابی به مقادیر آرایه با استفاده از حلقه for

در زیر مثالی در مورد استفاده از آرایه‌ها آمده است. در این برنامه ۵ مقدار از کاربر گرفته شده و میانگین آن‌ها حساب می‌شود:

```
#include <iostream>
using namespace std;

int main()
{
    int numbers[5];

    int total = 0;
    double average;

    for (int i = 0; i < size(numbers); i++)
    {
        cout << "Enter a number: ";
        cin >> numbers[i];
    }

    for (int i = 0; i < size(numbers); i++)
    {
        total += numbers[i];
    }

    average = total / (double)size(numbers);

    cout << "Average = " << average << endl;
}
```

```
Enter a number: 90
Enter a number: 85
Enter a number: 80
Enter a number: 87
Enter a number: 92
Average = 86
```

در خط ۶ یک آرایه تعریف شده است که می‌تواند ۵ عدد صحیح را در خود ذخیره کند. خطوط ۸ و ۹ متغیرهایی تعریف شده‌اند که از آن‌ها برای محاسبه میانگین استفاده می‌شود. توجه کنید که مقدار اولیه `total` صفر است تا از بروز خطا هنگام اضافه شدن مقدار به آن جلوگیری شود. در خطوط ۱۱ تا ۱۵ حلقه `for` برای تکرار و گرفتن ورودی از کاربر تعریف شده است. از متد `size()` برای تشخیص تعداد اجزای آرایه استفاده می‌شود. اگر چه می‌توانستیم به سادگی در حلقه `for` مقدار ۵ را برای شرط قرار دهیم ولی استفاده از خاصیت طول آرایه کار راحت‌تری است و می‌توانیم طول آرایه را تغییر دهیم و شرط حلقه `for` با تغییر جدید هماهنگ می‌شود. در خط ۱۴ ورودی دریافت شده از کاربر در آرایه ذخیره می‌شود. اندیس استفاده شده در `number` (خط ۱۴) مقدار `i` جاری در حلقه است. برای مثال در ابتدای حلقه مقدار `i` صفر است بنابراین وقتی در خط ۱۴ اولین داده از کاربر گرفته می‌شود اندیس آن برابر صفر می‌شود. در تکرار بعدی `i` یک واحد اضافه می‌شود و در نتیجه در خط ۱۴ و بعد از ورود دومین داده

توسط کاربر اندیس آن برابر یک می شود. این حالت تا زمانی که شرط در حلقه for برقرار است ادامه می یابد. در خطوط ۱۷-۲۰ از حلقه for دیگر برای دسترسی به مقدار هر یک از داده های آرایه استفاده شده است. در این حلقه نیز مانند حلقه قبل از مقدار متغیر شمارنده به عنوان اندیس استفاده می کنیم.

هر یک از اجزای عددی آرایه به متغیر total اضافه می شوند. بعد از پایان حلقه می توانیم میانگین اعداد را حساب کنیم (خط ۲۲). مقدار total را بر تعداد اجزای آرایه (تعداد عددها) تقسیم می کنیم. برای دسترسی به تعداد اجزای آرایه می توان از متد size() استفاده کرد. توجه کنید که در اینجا ما خروجی متد size() را به نوع double تبدیل کرده ایم بنابراین نتیجه عبارت یک مقدار از نوع double خواهد شد و دارای بخش کسری می باشد. حال اگر عملوندهای تقسیم را به نوع double تبدیل نکنیم نتیجه تقسیم یک عدد از نوع صحیح خواهد شد و دارای بخش کسری نیست. خط ۲۴ مقدار میانگین را در صفحه نمایش چاپ می کند. طول آرایه بعد از مقدار دهی نمی تواند تغییر کند. به عنوان مثال اگر یک آرایه را که شامل ۵ جز است مقدار دهی کنید دیگر نمی توانید آن را مثلاً به ۱۰ جز تغییر اندازه دهید. البته تعداد خاصی از کلاس ها مانند آرایه ها عمل می کنند و توانایی تغییر تعداد اجزای تشکیل دهنده خود را دارند. آرایه ها در برخی شرایط بسیار پر کاربرد هستند و تسلط شما بر این مفهوم و اینکه چطور از آن ها استفاده کنید بسیار مهم است.

## آرایه های چند بعدی

آرایه های چند بعدی آرایه هایی هستند که برای دسترسی به هر یک از عناصر آن ها باید از چندین اندیس استفاده کنیم. یک آرایه چند بعدی را می توان مانند یک جدول با تعدادی ستون و ردیف تصور کنید. با افزایش اندیس ها اندازه ابعاد آرایه نیز افزایش می یابد و آرایه های چند بعدی با بیش از دو اندیس به وجود می آیند. نحوه ایجاد یک آرایه با دو بعد به صورت زیر است:

```
datatype arrayName[lengthX][lengthY];
```

و یک آرایه سه بعدی به صورت زیر ایجاد می شود:

```
datatype arrayName[lengthX][lengthY][lengthZ];
```

می توان یک آرایه با تعداد زیادی بعد ایجاد کرد به شرطی که هر بعد دارای طول مشخصی باشد. به دلیل اینکه آرایه های سه بعدی یا آرایه های با بیشتر از دو بعد بسیار کمتر مورد استفاده قرار می گیرند اجازه بدهید که در این درس بر روی آرایه های دو بعدی تمرکز کنیم. در تعریف این نوع آرایه ابتدا نوع آرایه یعنی اینکه آرایه چه نوعی از انواع داده را در خود ذخیره می کند را مشخص می کنیم. سپس نام آرایه و در نهایت دو جفت کروشه قرار می دهیم. در یک آرایه دو بعدی برای دسترسی به هر یک از عناصر به دو مقدار

```
int numbers[3][5];
```

number [3][5]

number [0][1]	number [0][2]	number [0][3]	number [0][4]	number [0][5]
number [1][1]	number [1][2]	number [1][3]	number [1][4]	number [1][5]
number [2][1]	number [2][2]	number [2][3]	number [2][4]	number [2][5]

```
datatype arrayName[x][y] = { { r0c0, r0c1, ... r0cX },
                               { r1c0, r1c1, ... r1cX },
                               .
                               .
                               .
                               { rYc0, rYc1, ... rYcX }
                               };
```

```
datatype arrayName[][y] = { { r0c0, r0c1, ... r0cX },
                              { r1c0, r1c1, ... r1cX },
                              .
                              .
                              .
                              { rYc0, rYc1, ... rYcX }
                              };
```

به عنوان مثال:

```
int numbers[][5] = { { 1, 2, 3, 4, 5 },
                     { 6, 7, 8, 9, 10 },
                     { 11, 12, 13, 14, 15 }
                   };
```

و یا می‌توان مقدار دهی به عناصر را به صورت دستی انجام داد مانند:

```
array[0][0] = value;
array[0][1] = value;
array[0][2] = value;
array[1][0] = value;
array[1][1] = value;
array[1][2] = value;
array[2][0] = value;
array[2][1] = value;
array[2][2] = value;
```

همانطور که مشاهده می‌کنید برای دسترسی به هر یک از عناصر در یک آرایه دو بعدی به سادگی می‌توان از اندیس‌های X و Y و یک جفت کروشه مانند مثال استفاده کرد.

## گردش در میان عناصر آرایه‌های چند بعدی

گردش در میان عناصر آرایه‌های چند بعدی نیاز به کمی دقت دارد. برنامه زیر نشان می‌دهد که چطور از حلقه for برای خواندن همه مقادیر آرایه و تعیین انتهای ردیف‌ها استفاده کنید.

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int numbers[3][5] = {
7          { 1 , 2 , 3 , 4 , 5 },
8          { 6 , 7 , 8 , 9 , 10 },
9          { 11, 12, 13, 14, 15 }
10     };
11
12     for (int row = 0; row < size(numbers); row++)
13     {
14         for (int col = 0; col < size(numbers[0]); col++)
15         {
16             cout << numbers[row][col] << " ";
17         }
18
19         //Go to the next line
20         cout << endl;
21     }
22 }
```

```
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
```

همانطور که در مثال بالا نشان داده شده است با استفاده از یک حلقه `for` نمی‌توان به مقادیر دسترسی یافت بلکه به یک حلقه `for` تو در تو نیاز داریم، زیرا آرایه دو بعدی به صورت یک جدول شامل سطر و ستون است، پس لازم است که از یک حلقه `for` برای گردش در میان سطرها و از حلقه `for` دیگر برای گردش در میان ستون‌های این جدول (آرایه) استفاده کنیم. اولین حلقه `for` (خط ۱۲) برای گردش در میان ردیف‌های آرایه به کار می‌رود. این حلقه تا زمانی ادامه می‌یابد که مقدار ردیف کمتر از طول اولین بعد باشد (زیرا اندیس ابعاد آرایه از صفر شروع می‌شود. در مثال بالا مقدار اولین بعد برابر ۳ است). در این مثال از متد `size()` استفاده کرده‌ایم. این متد طول آرایه را در یک بعد خاص نشان می‌دهد. به عنوان مثال برای به دست آوردن طول اولین یا همان تعداد سطرها کافیست که نام آرایه را به این متد ارسال می‌کنیم.

در داخل اولین حلقه `for` حلقه `for` دیگری تعریف شده است (خط ۱۴). در این حلقه یک شمارنده برای شمارش تعداد ستون‌های (`col`) هر ردیف تعریف شده است و در شرط داخل آن بار دیگر از متد `size()` استفاده شده است، ولی این بار مقدار `numbers[0]` را به آن ارسال می‌کنیم تا طول بعد دوم آرایه را به دست آوریم. پس به عنوان مثال وقتی که مقدار ردیف (`row`) صفر باشد، حلقه دوم از `[0][0]` تا `[4][0]` اجرا می‌شود. سپس مقدار هر عنصر از آرایه را با استفاده از حلقه نشان می‌دهیم، اگر مقدار ردیف (`row`) برابر ۰ و مقدار ستون (`col`) برابر ۰ باشد مقدار عنصری که در ستون ۱ و ردیف ۱ (`numbers[0][0]`) قرار دارد نشان داده خواهد شد که در مثال بالا عدد ۱ است.

بعد از اینکه دومین حلقه تکرار به پایان رسید، فوراً دستورات بعد از آن اجرا خواهند شد، که در اینجا دستور `<< endl; cout` که به برنامه اطلاع می‌دهد که به خط بعد برود. سپس حلقه با اضافه کردن یک واحد به مقدار `row` این فرایند را دوباره تکرار می‌کند. سپس دومین حلقه `for` اجرا شده و مقادیر دومین ردیف نمایش داده می‌شود. این فرایند تا زمانی اجرا می‌شود که مقدار `row` کمتر از طول اولین بعد باشد. حال بیایید آنچه را از قبل یاد گرفته‌ایم در یک برنامه به کار ببریم. این برنامه نمره چهار درس مربوط به سه دانش آموز را از ما می‌گیرد و معدل سه دانش آموز را حساب می‌کند.

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      double studentGrades[3][4];
8      double total;
9
10     for (int student = 0; student < size(studentGrades); student++)
11     {
```

```
12     total = 0;
13
14     cout << "Enter grades for Student " << (student + 1) << endl;
15
16     for (int grade = 0; grade < size(studentGrades[0]); grade++)
17     {
18         cout << "Enter Grade #" << (grade + 1) << " : " ;
19         cin >> studentGrades[student][grade];
20         total += studentGrades[student][grade];
21     }
22
23     cout << "Average is " << (total / size(studentGrades[0])) << endl;
24     cout << endl;
25 }
26 }
```

```
Enter grades for Student 1
Enter Grade #1: 92
Enter Grade #2: 87
Enter Grade #3: 89
Enter Grade #4: 95
Average is 90.75
```

```
Enter grades for Student 2
Enter Grade #1: 85
Enter Grade #2: 85
Enter Grade #3: 86
Enter Grade #4: 87
Average is 85.75
```

```
Enter grades for Student 3
Enter Grade #1: 90
Enter Grade #2: 90
Enter Grade #3: 90
Enter Grade #4: 90
Average is 90.00
```

در برنامه بالا یک آرایه چند بعدی از نوع double تعریف شده است (خط ۷). همچنین یک متغیر به نام total تعریف می‌کنیم که جمع نمرات وارد شده برای دانش آموز در آن قرار می‌گیرد. حال وارد حلقه for تو در تو می‌شویم (خط ۱۰) در اولین حلقه for یک متغیر به نام student تعریف کرده‌ایم که مقادیر اولین بعد آرایه (که همان تعداد دانش آموزان است) در آن قرار می‌گیرد. از متد متد size() هم برای تشخیص تعداد دانش آموزان استفاده شده است. وارد بدنه حلقه for می‌شویم. در خط ۱۲ مقدار متغیر total را برابر صفر قرار می‌دهیم. سپس برنامه یک پیغام را نشان می‌دهد و از شما می‌خواهد که نمرات دانش آموز را وارد کنید. (student + 1) عدد ۱ را به student اضافه کرده‌ایم تا به جای نمایش Student 0، با Student 1 شروع شود، تا طبیعی تر به نظر برسد.



سپس به دومین حلقه for در خط ۱۶ می‌رسیم. وظیفه این حلقه گردش در میان دومین بعد که همان نمرات دانش آموز است می‌باشد. برنامه چهار نمره مربوط به دانش آموز را می‌گیرد. هر وقت که برنامه یک نمره را از کاربر دریافت می‌کند، نمره به متغیر total اضافه می‌شود. وقتی همه نمره‌ها وارد شدند، متغیر total هم جمع همه نمرات را نشان می‌دهد. در خطوط ۲۳-۲۴ معدل دانش آموز نشان داده می‌شود. معدل از تقسیم کردن total (جمع) بر تعداد نمرات به دست می‌آید. از size(studentGrades[0]) هم برای به دست آوردن تعداد نمرات استفاده می‌شود.

## متد

متدها به شما اجازه می‌دهند که یک رفتار یا وظیفه را تعریف کنید و مجموعه ای از کدها هستند که در هر جای برنامه می‌توان از آن‌ها استفاده کرد. متدها دارای آرگومان‌هایی هستند که وظیفه متد را مشخص می‌کنند. نمی‌توان یک متد را در داخل متد دیگر تعریف کرد. وقتی که شما در برنامه یک متد را صدا می‌زنید برنامه به قسمت تعریف متد رفته و کدهای آن را اجرا می‌کند. در C++ متدی وجود دارد که نقطه آغاز هر برنامه است و بدون آن برنامه‌ها نمی‌دانند با ید از کجا شروع شوند، این متد main() نام دارد. پارامترها همان چیزهایی هستند که متد منتظر دریافت آن‌ها است.

آرگومان‌ها مقادیری هستند که به پارامترها ارسال می‌شوند.

گاهی اوقات دو کلمه پارامتر و آرگومان به یک منظور به کار می‌روند. ساده‌ترین ساختار یک متد به صورت زیر است:

```
returnType MethodName(Parameter List)
{
    code to execute;
}
```

به برنامه ساده زیر توجه کنید. در این برنامه از یک متد برای چاپ یک پیغام در صفحه نمایش استفاده شده است:

```
1  #include <iostream>
2  using namespace std;
3
4  void PrintMessage()
5  {
6      cout << "Hello World!";
7  }
8
9  int main()
10 {
11     PrintMessage();
12 }
```

در خطوط ۷-۴ یک متد تعریف کرده ایم. مکان تعریف آن در داخل کلاس مهم نیست. به عنوان مثال می توانید آن را زیر متد `main()` تعریف کنید. می توان این متد را در داخل متد دیگر صدا زد (فراخوانی کرد). متد دیگر ما در اینجا متد `main()` است که می توانیم در داخل آن نام متدی که برای چاپ یک پیغام تعریف کرده ایم (یعنی متد `PrintMessage()`) را صدا بزنیم. در تعریف متد بالا کلمه کلیدی `void` آمده است که نشان دهنده آن است که متد مقدار برگشتی ندارد. در درس آینده در مورد مقدار برگشتی از یک متد و استفاده از آن برای اهداف مختلف توضیح داده خواهد شد. نام متد ما `PrintMessage()` است. به این نکته توجه کنید که در نامگذاری متد از روش پاسکال (حرف اول هر کلمه بزرگ نوشته می شود) استفاده کرده ایم. این روش نامگذاری قراردادی است و می توان از این روش استفاده نکرد، اما پیشنهاد می شود که از این روش برای تشخیص متدها استفاده کنید. بهتر است در نامگذاری متدها از کلماتی استفاده شود که کار متد را مشخص می کند مثلاً نامهایی مانند `GoToBed` یا `OpenDoor`. همچنین به عنوان مثال اگر مقدار برگشتی متد یک مقدار بولی باشد می توانید اسم متد خود را به صورت یک کلمه سوالی انتخاب کنید مانند `IsLeapyear` یا `IsTeenager...` ولی از گذاشتن علامت سؤال در آخر اسم متد خودداری کنید. دو پرانتزی که بعد از نام می آید نشان دهنده آن است که نام متد متعلق به یک متد است. در این مثال در داخل پرانتزها هیچ چیزی نوشته نشده چون پارامتری ندارد. در درس های آینده در مورد متدها بیشتر توضیح می دهیم. بعد از پرانتزها دو آکولاد قرار می دهیم که بدنه متد را تشکیل می دهد و کدهایی را که می خواهیم اجرا شوند را در داخل این آکولادها می نویسیم. در داخل متد `main()` متدی را که در خط ۱۱ ایجاد کرده ایم را صدا می زنیم. برای صدا زدن یک متد کافیست نام آن را نوشته و بعد از نام پرانتزها را قرار دهیم.

اگر متد دارای پارامتر باشد باید شما آراگومانها را به ترتیب در داخل پرانتزها قرار دهید. در این مورد نیز در درس های آینده توضیح بیشتری می دهیم. با صدا زدن یک متد کدهای داخل بدنه آن اجرا می شوند. برای اجرای متد `PrintMessage()` برنامه از متد `main()` به محل تعریف متد `PrintMessage()` می رود. مثلاً وقتی ما متد `PrintMessage()` را در خط ۱۱ صدا می زنیم برنامه از خط ۱۱ به خط ۴، یعنی جایی که متد تعریف شده می رود. اکنون ما یک متد در برنامه `class` داریم و همه متدهای این برنامه می توانند آن را صدا بزنند.

## مقدار برگشتی از یک متد

متدها می توانند مقدار برگشتی از هر نوع داده ای داشته باشند. این مقادیر می توانند در محاسبات یا به دست آوردن یک داده مورد استفاده قرار بگیرند. در زندگی روزمره فرض کنید که کارمند شما یک متد است و شما او را صدا می زنید و از او می خواهید که کار یک سند را به پایان برساند. سپس از او می خواهید که بعد از اتمام کارش سند را به شما تحویل دهد. سند همان مقدار برگشتی

متد است. نکته مهم در مورد یک متد، مقدار برگشتی و نحوه استفاده شما از آن است. برگشت یک مقدار از یک متد آسان است. کافیهست در تعریف متد به روش زیر عمل کنید :

```
returnType MethodName()
{
    return value;
}
```

returnType در اینجا نوع داده ای مقدار برگشتی را مشخص می کند (bool,int). در داخل بدنه متد کلمه کلیدی return و بعد از آن یک مقدار یا عبارتی که نتیجه آن یک مقدار است را می نویسیم. نوع این مقدار برگشتی باید از انواع ساده بوده و در هنگام نامگذاری متد و قبل از نام متد ذکر شود. اگر متد ما مقدار برگشتی نداشته باشد باید از کلمه void قبل از نام متد استفاده کنیم. مثال زیر یک متد که دارای مقدار برگشتی است را نشان می دهد.

```
1  #include <iostream>
2  using namespace std;
3
4  int CalculateSum()
5  {
6      int firstNumber = 10;
7      int secondNumber = 5;
8
9      int sum = firstNumber + secondNumber;
10
11     return sum;
12 }
13
14 int main()
15 {
16     int result = CalculateSum();
17
18     cout << "Sum is " << result;
19 }
```

همانطور که در خط ۴ مثال فوق مشاهده می کنید هنگام تعریف متد از کلمه int به جای void استفاده کرده ایم که نشان دهنده آن است که متد ما دارای مقدار برگشتی از نوع اعداد صحیح است. در خطوط ۶ و ۷ دو متغیر تعریف و مقدار دهی شده اند.

توجه کنید که این متغیرها، متغیرهای محلی هستند. و این بدان معنی است که این متغیرها در سایر متدها مانند متد main() قابل دسترسی نیستند و فقط در متدی که در آن تعریف شده اند قابل استفاده هستند. در خط ۱۰ جمع دو متغیر در متغیر sum قرار می گیرد. در خط ۱۱ مقدار برگشتی sum توسط دستور return فراخوانی می شود. در داخل متد main() یک متغیره نام result در خط ۱۶ تعریف می کنیم و متد CalculateSum() را فراخوانی می کنیم.

متد `CalculateSum()` مقدار ۱۵ را بر می گرداند که در داخل متغیر `result` ذخیره می شود. در خط ۱۸ مقدار ذخیره شده در متغیر `result` چاپ می شود. متدی که در این مثال ذکر شد متد کاربردی و مفیدی نیست. با وجودیکه کدهای زیادی در متد بالا نوشته شده ولی همیشه مقدار برگشتی ۱۵ است، در حالیکه می توانستیم به راحتی یک متغیر تعریف کرده و مقدار ۱۵ را به آن اختصاص دهیم. این متد در صورتی کارآمد است که پارامترهایی به آن اضافه شود که در درس های آینده توضیح خواهیم داد. هنگامی که می خواهیم در داخل یک متد از دستور `if` یا `switch` استفاده کنیم باید تمام کدها دارای مقدار برگشتی باشند. برای درک بهتر این مطلب به مثال زیر توجه کنید:

```

1  #include <iostream>
2  using namespace std;
3
4  int GetNumber()
5  {
6      int number;
7
8      cout << "Enter a number greater than 10: ";
9      cin >> number;
10
11     if (number > 10)
12     {
13         return number;
14     }
15     else
16     {
17         return 0;
18     }
19 }
20
21 int main()
22 {
23     int result = GetNumber();
24
25     cout << "Result is " << result;
26 }
```

```

Enter a number greater than 10: 11
Result = 11
```

```

Enter a number greater than 10: 9
Result = 0
```

در خطوط ۴-۱۹ یک متد با نام `GetNumber()` تعریف شده است که از کاربر یک عدد بزرگتر از ۱۰ را می خواهد. اگر عدد وارد شده توسط کاربر درست نباشد متد مقدار صفر را بر می گرداند. و اگر قسمت `else` دستور `if` و یا دستور `return` را از آن حذف کنیم در هنگام اجرای برنامه با پیغام خطا مواجه می شویم.

چون اگر شرط دستور `if` نادرست باشد (کاربر مقداری کمتر از ۱۰ را وارد کند) برنامه به قسمت `else` می رود تا مقدار صفر را بر گرداند و چون قسمت `else` حذف شده است برنامه با خطا مواجه می شود و همچنین اگر دستور `return` حذف شود چون برنامه

نیاز به مقدار برگشتی دارد پیغام خطا می‌دهد. و آخرین مطلبی که در این درس می‌خواهیم به شما آموزش دهیم این است که شما می‌توانید از یک متد که مقدار برگشتی ندارد خارج شوید. حتی اگر از نوع داده ای void در یک متد استفاده می‌کنید باز هم می‌توانید کلمه کلیدی return را در آن به کار ببرید. استفاده از return باعث خروج از بدنه متد و اجرای کدهای بعد از آن می‌شود.

```

1  #include <iostream>
2  using namespace std;
3
4  void TestReturnExit()
5  {
6      cout << "Line 1 inside the method TestReturnExit()" << endl;
7      cout << "Line 2 inside the method TestReturnExit()" << endl;
8
9      return;
10
11     //The following lines will not execute
12     cout << "Line 3 inside the method TestReturnExit()" << endl;
13     cout << "Line 4 inside the method TestReturnExit()" << endl;
14 }
15
16 int main()
17 {
18     TestReturnExit();
19
20     cout << "Hello World!";
21 }

```

```

Line 1 inside the method TestReturnExit()
Line 2 inside the method TestReturnExit()
Hello World!

```

در برنامه بالا نحوه خروج از متد با استفاده از کلمه کلیدی return نادیده گرفتن همه کدهای بعد از این کلمه کلیدی نشان داده شده است. در پایان برنامه متد تعریف شده (TestReturnExit()) در داخل متد main() فراخوانی و اجرا می‌شود.

## پارامترها و آرگومان‌ها

پارامترها داده‌های خامی هستند که متد آن‌ها را پردازش می‌کند و سپس اطلاعاتی را که به دنبال آن هستید در اختیار شما قرار می‌دهد. فرض کنید پارامترها مانند اطلاعاتی هستند که شما به یک کارمند می‌دهید که بر طبق آن‌ها کارش را به پایان برساند. یک متد می‌تواند هر تعداد پارامتر داشته باشد. هر پارامتر می‌تواند از انواع مختلف داده باشد. در زیر یک متد با N پارامتر نشان داده شده است:

```

returnType MethodName(datatype param1, datatype param2, ... datatype paramN)
{

```

```
code to execute;  
}
```

پارامترها بعد از نام متد و بین پرانتزها قرار می گیرند. بر اساس کاری که متد انجام می دهد می توان تعداد پارامترهای زیادی به متد اضافه کرد. بعد از فراخوانی یک متد باید آرگومان های آن را نیز تأمین کنید. آرگومان ها مقادیری هستند که به پارامترها اختصاص داده می شوند. ترتیب ارسال آرگومان ها به پارامترها مهم است. عدم رعایت ترتیب در ارسال آرگومان ها باعث به وجود آمدن خطای منطقی و خطای زمان اجرا می شود. اجازه بدهید که یک مثال بزنیم:

```
1  #include <iostream>  
2  
3  using namespace std;  
4  
5  int CalculateSum(int number1, int number2)  
6  {  
7      return number1 + number2;  
8  }  
9  
10 int main()  
11 {  
12     int num1, num2;  
13  
14     cout << "Enter the first number: ";  
15     cin >> num1;  
16     cout << "Enter the second number: ";  
17     cin >> num2;  
18  
19     cout << "Sum = " << CalculateSum(num1, num2);  
20 }
```

```
Enter the first number: 10  
Enter the second number: 5  
Sum = 15
```

در برنامه بالا یک متد به نام CalculateSum() (خطوط ۵-۸) تعریف شده است که وظیفه آن جمع مقدار دو عدد است. چون این متد مقدار دو عدد صحیح را با هم جمع می کند پس نوع برگشتی ما نیز باید int باشد. متد دارای دو پارامتر است که اعداد را به آن ها ارسال می کنیم. به نوع داده ای پارامترها توجه کنید. هر دو پارامتر یعنی number1 و number2 مقادیری از نوع اعداد صحیح (int) دریافت می کنند. در بدنه متد دستور return نتیجه جمع دو عدد را بر می گرداند. در داخل متد Main برنامه از کاربر دو مقدار را درخواست می کند و آن ها را داخل متغیرها قرار می دهد. حال متد را که آرگومان های آن را آماده کرده ایم فراخوانی می کنیم. مقدار num1 به پارامتر اول و مقدار num2 به پارامتر دوم ارسال می شود. حال اگر مکان دو مقدار را هنگام ارسال به متد تغییر دهیم (یعنی مقدار num2 به پارامتر اول و مقدار num1 به پارامتر دوم ارسال شود) هیچ تغییری در نتیجه متد ندارد چون جمع خاصیت جابه جایی دارد.

فقط به یاد داشته باشید که باید ترتیب ارسال آرگومان‌ها هنگام فراخوانی متد دقیقاً با ترتیب قرار گیری پارامترها تعریف شده در متد مطابقت داشته باشد. بعد از ارسال مقادیر ۱۰ و ۵ به پارامترها ، پارامترها آن‌ها را دریافت می‌کنند. به این نکته نیز توجه کنید که نام پارامترها طبق قرارداد به شیوه کوهان شتری یا camelCasing (حرف اول دومین کلمه بزرگ نوشته می‌شود) نوشته می‌شود. در داخل بدنه متد (خط ۷) دو مقدار با هم جمع می‌شوند و نتیجه به متد فراخوان (متدی که متد CalculateSum() را فراخوانی می‌کند) ارسال می‌شود. در درس آینده از یک متغیر برای ذخیره نتیجه محاسبات استفاده می‌کنیم ولی در اینجا مشاهده می‌کنید که می‌توان به سادگی نتیجه جمع را نشان داد (خط ۷). در داخل متد main() از ما دو عدد که قرار است با هم جمع شوند درخواست می‌شود.

در خط ۱۹ متد CalculateSum() را فراخوانی می‌کنیم و دو مقدار صحیح به آن ارسال می‌کنیم. دو عدد صحیح در داخل متد با هم جمع شده و نتیجه آن‌ها برگردانده می‌شود. مقدار برگشت داده شده از متد به وسیله متد cout نمایش داده می‌شود (خط ۱۹). در برنامه زیر یک متد تعریف شده است که دارای دو پارامتر از دو نوع داده ای مختلف است:

```

1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  void ShowMessageAndNumber(string message, int number)
6  {
7      cout << message << endl;
8      cout << "Number = " << number;
9  }
10
11 int main()
12 {
13     ShowMessageAndNumber("Hello World!", 100);
14 }
```

```

Hello World!
Number = 100
```

در مثال بالا یک متدی تعریف شده است که اولین پارامتر آن مقداری از نوع رشته و دومین پارامتر آن مقداری از نوع int دریافت می‌کند. متد به سادگی دو مقداری که به آن ارسال شده است را نشان می‌دهد. در خط ۱۳ متد را اول با یک رشته و سپس یک عدد خاص فراخوانی می‌کنیم. حال اگر متد به صورت زیر فراخوانی می‌شد:

```
ShowMessageAndNumber(100, "Welcome to Gimme C++!");
```

در برنامه خطا به وجود می‌آید چون عدد ۱۰۰ به پارامتری از نوع رشته و رشته Hello World! به پارامتری از نوع اعداد صحیح ارسال می‌شد. این نشان می‌دهد که ترتیب ارسال آرگومان‌ها به پارامترها هنگام فراخوانی متد مهم است. به مثال ۱ توجه کنید در

آن مثال دو عدد از نوع `int` به پارامترها ارسال کردیم که ترتیب ارسال آن‌ها چون هردو پارامتر از یک نوع بودند مهم نبود. ولی اگر پارامترهای متد دارای اهداف خاصی باشند ترتیب ارسال آرگومان‌ها مهم است.

```
void ShowPersonStats(int age, int height)
{
    cout << "Age = " << age;
    cout << "Height = " << height;
}

//Using the proper order of arguments
ShowPersonStats(20, 160);

//Acceptable, but produces odd results
ShowPersonStats(160, 20);
```

در مثال بالا نشان داده شده است که حتی اگر متد دو آرگومان با یک نوع داده ای قبول کند باز هم بهتر است ترتیب بر اساس تعریف پارامترها رعایت شود. به عنوان مثال در اولین فراخوانی متد بالا اشکالی به چشم نمی‌آید چون سن شخص ۲۰ و قد او ۱۶۰ سانتی متر است. اگر آرگومان‌ها را به ترتیب ارسال نکنیم سن شخص ۱۶۰ و قد او ۲۰ سانتی متر می‌شود که به واقعیت نزدیک نیست. دانستن مبانی مقادیر برگشتی و ارسال آرگومان‌ها باعث می‌شود که شما متدهای کارآمد تری تعریف کنید. تکه کد زیر نشان می‌دهد که شما حتی می‌توانید مقدار برگشتی از یک متد را به عنوان آرگومان به متد دیگر ارسال کنید.

```
int MyMethod()
{
    return 5;
}

void AnotherMethod(int number)
{
    cout << number;
}

// Codes skipped for demonstration

AnotherMethod(MyMethod());
```

چون مقدار برگشتی متد `MyMethod()` عدد ۵ است و به عنوان آرگومان به متد `AnotherMethod()` ارسال می‌شود خروجی کد بالا هم عدد ۵ است.

## ارسال آرگومان‌ها به روش ارجاع

آرگومان‌ها را می‌توان به کمک ارجاع ارسال کرد. این بدان معناست که شما آدرس متغیری را ارسال می‌کنید نه مقدار آن را. ارسال با ارجاع زمانی مفید است که شما بخواهید یک آرگومان که دارای مقدار بزرگی است (مانند یک آبجکت) را ارسال کنید. در این



حالت وقتی که آرگومان ارسال شده را در داخل متد اصلاح می‌کنیم مقدار اصلی آرگومان در خارج از متد هم تغییر می‌کند. در زیر دستورالعمل پایه ای تعریف پارامترها که در آن‌ها به جای مقدار از آدرس استفاده شده است نشان داده شده است. برای ارسال یک متغیر با ارجاع کافیهست که قبل از نام پارامتر یک علامت & قرار دهید :

```
returnType MethodName(datatype &param1)
{
    code to execute;
}
```

اجازه دهید که تفاوت بین ارسال با ارجاع و ارسال با مقدار آرگومان را با یک مثال توضیح دهیم :

```
1  #include <iostream>
2  using namespace std;
3
4  void ModifyNumberVal(int number)
5  {
6      number += 10;
7      cout << "Value of number inside method is " << number << endl;
8  }
9
10 void ModifyNumberRef(int &number)
11 {
12     number += 10;
13     cout << "Value of number inside method is " << number << endl;
14 }
15
16 int main()
17 {
18     int num = 5;
19
20     cout << "num = " << num << endl;
21
22     cout << "Passing num by value to method ModifyNumberVal() ..." << endl;
23     ModifyNumberVal(num);
24     cout << "Value of num after exiting the method is " << num << endl;
25
26     cout << "Passing num by ref to method ModifyNumberRef() ..." << endl;
27     ModifyNumberRef(num);
28     cout << "Value of num after exiting the method is " << num << endl;
29 }
```

```
num = 5
Passing num by value to method ModifyNumberVal() ...
Value of number inside method is 15
Value of num after exiting the method is 5
Passing num by ref to method ModifyNumberRef() ...
Value of number inside method is 15
Value of num after exiting the method is 15
```

در برنامه بالا دو متد که دارای یک هدف یکسان هستند تعریف شده‌اند و آن اضافه کردن عدد ۱۰ به مقداری است که به آن‌ها ارسال می‌شود. اولین متد (خطوط ۵-۹) دارای یک پارامتر است که نیاز به یک مقدار آرگومان (از نوع int) دارد. وقتی که متد را

صدا می‌زنیم و آرگومانی به آن اختصاص می‌دهیم (خط ۲۴)، کپی آرگومان به پارامتر متد ارسال می‌شود. بنابراین مقدار اصلی متغیر خارج از متد هیچ ارتباطی به پارامتر متد ندارد. سپس مقدار ۱۰ را به متغیر پارامتر (number) اضافه کرده و نتیجه را چاپ می‌کنیم. برای اثبات اینکه متغیر num هیچ تغییری نکرده است مقدار آن را یکبار دیگر چاپ کرده و مشاهده می‌کنیم که تغییری نکرده است. دومین متد (خطوط ۱۱-۱۵) نیاز به یک مقدار با ارجاع دارد. در این حالت به جای اینکه یک کپی از مقدار به عنوان آرگومان به آن ارسال شود آدرس متغیر به آن ارسال می‌شود. حال پارامتر به مقدار اصلی متغیر که زمان فراخوانی متد به آن ارسال می‌شود دسترسی دارد. وقتی که ما مقدار متغیر پارامتری که شامل آدرس متغیر اصلی است را تغییر می‌دهیم (خط ۱۳) در واقع مقدار متغیر اصلی در خارج از متد را تغییر داده‌ایم. در نهایت مقدار اصلی متغیر را وقتی که از متد خارج شدیم را نمایش می‌دهیم و مشاهده می‌شود که مقدار آن واقعاً تغییر کرده است.

## ارسال آرایه به عنوان آرگومان

می‌توان آرایه‌ها را به عنوان آرگومان به متد ارسال کرد. ابتدا شما باید پارامترهای متد را طوری تعریف کنید که آرایه دریافت کنند. به مثال زیر توجه کنید.

```

1  #include <iostream>
2  using namespace std;
3
4  void TestArray(int numbers[])
5  {
6      for (int i = 0; i <= sizeof(numbers) ; i++)
7      {
8          cout << numbers[i] << endl;
9      }
10 }
11
12 int main()
13 {
14     int array[] = { 1, 2, 3, 4, 5 };
15
16     TestArray(array);
17 }
```

```

1
2
3
4
5
```

مشاهده کردید که به سادگی می‌توان با گذاشتن کروشه بعد از نام پارامتر یک متد ایجاد کرد که پارامتر آن، آرایه دریافت می‌کند. وقتی متد در خط ۱۶ فراخوانی می‌شود، آرایه را فقط با استفاده از نام آن و بدون استفاده از اندیس ارسال می‌کنیم. پس آرایه‌ها هم

به روش ارجاع به متدها ارسال می‌شوند. در خطوط ۹-۶ از حلقه for برای دسترسی به اجزای اصلی آرایه که به عنوان آرگومان به متد ارسال کرده‌ایم استفاده می‌کنیم. در زیر نحوه ارسال یک آرایه به روش ارجاع نشان داده شده است.

```

1  #include <iostream>
2  using namespace std;
3
4  void IncrementElements(int numbers[])
5  {
6      for (int i = 0; i <= sizeof(numbers); i++)
7      {
8          numbers[i]++;
9      }
10 }
11
12 int main()
13 {
14     int array[] = { 1, 2, 3, 4, 5 };
15
16     IncrementElements(array);
17
18     for (int i = 0; i < size(array); i++)
19     {
20         cout << array[i] << endl;
21     }
22 }
```

```

2
3
4
5
6
```

برنامه بالا یک متد را نشان می‌دهد که یک آرایه را دریافت می‌کند و به هر یک از عناصر آن یک واحد اضافه می‌کند. در داخل متد ما مقادیر هر یک از اجزای آرایه را افزایش داده‌ایم. سپس از متد خارج شده و نتیجه را نشان می‌دهیم. مشاهده می‌کنید که هر یک از مقادیر اصلی متد هم اصلاح شده‌اند. می‌توان دو یا چند آرایه را به عنوان آرگومان به متد ارسال کرد:

```

void MyMethod(int param1[], int param2[])
{
    //code here
}
```

## محدوده متغیر

متغیرها در C++ دارای محدوده هستند. محدوده یک متغیر به شما می‌گوید که در کجای برنامه می‌توان از متغیر استفاده کرد و یا متغیر قابل دسترسی است. به عنوان مثال متغیری که در داخل یک متد تعریف می‌شود فقط در داخل بدنه متد قابل دسترسی است. می‌توان دو متغیر با نام یکسان در دو متد مختلف تعریف کرد. برنامه زیر این ادعا را اثبات می‌کند:

```
#include <iostream>
using namespace std;

void DemonstrateScope()
{
    int number = 5;

    cout << "number inside method DemonstrateScope() = " << number << endl;
}

int main()
{
    int number = 10;

    DemonstrateScope();

    cout << "number inside the Main method = " << number << endl;
}
```

```
number inside method DemonstrateScope() = 5
number inside the Main method = 10
```

مشاهده می‌کنید که حتی اگر ما دو متغیر با نام یکسان تعریف کنیم که دارای محدوده‌های متفاوتی هستند، می‌توان به هر کدام از آن‌ها مقادیر مختلفی اختصاص داد. متغیر تعریف شده در داخل متد main() هیچ ارتباطی به متغیر داخل متد DemonstrateScope() ندارد. وقتی به مبحث کلاس‌ها رسیدیم در این باره بیشتر توضیح خواهیم داد.

## پارامترهای اختیاری

پارامترهای اختیاری همانگونه که از اسمشان پیداست اختیاری هستند و می‌توان به آن‌ها آرگومان ارسال کرد یا نه. این پارامترها دارای مقادیر پیشفرضی هستند. اگر به اینگونه پارامترها آرگومانی ارسال نشود از مقادیر پیشفرض استفاده می‌کنند. به مثال زیر توجه کنید:

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
```

```

5 void PrintMessage(string message = "Welcome to Visual C# Tutorials!")
6 {
7     cout << message << endl;
8 }
9
10 int main()
11 {
12     PrintMessage();
13
14     PrintMessage("Learn C# Today!");
15 }

```

```

Welcome to Visual C# Tutorials!
Learn C# Today!

```

متد (`PrintMessage()` خطوط ۵-۸) یک پارامتر اختیاری دارد. برای تعریف یک پارامتر اختیاری می‌توان به آسانی و با استفاده از علامت = یک مقدار را به یک پارامتر اختصاص داد (مثال بالا خط ۵). دو بار متد را فراخوانی می‌کنیم. در اولین فراخوانی (خط ۱۲) ما آرگومانی به متد ارسال نمی‌کنیم بنابراین متد از مقدار پیشفرض (`Welcome to Visual C# Tutorials!`) استفاده می‌کند. در دومین فراخوانی (خط ۱۴) یک پیغام (آرگومان) به متد ارسال می‌کنیم که جایگزین مقدار پیشفرض پارامتر می‌شود. اگر از چندین پارامتر در متد استفاده می‌کنید همه پارامترهای اختیاری باید در آخر بقیه پارامترها ذکر شوند. به مثال‌های زیر توجه کنید.

```

void SomeMethod(int opt1 = 10, int opt2 = 20, int req1, int req2) //ERROR
void SomeMethod(int req1, int opt1 = 10, int req2, int opt2 = 20) //ERROR
void SomeMethod(int req1, int req2, int opt1 = 10, int opt2 = 20) //Correct

```

وقتی متدهای با چندین پارامتر اختیاری فراخوانی می‌شوند باید به پارامترهایی که از لحاظ مکانی در آخر بقیه پارامترها نیستند مقدار اختصاص داد. به یاد داشته باشید که نمی‌توان برای نادیده گرفتن یک پارامتر به صورت زیر عمل کرد:

```

void SomeMethod(int required1, int optional1 = 10, int optional2 = 20)
{
    //Some Code
}

// ... Code omitted for demonstration

SomeMethod(10, , 100); //Error

```

## سربارگذاری متدها

سربارگذاری متدها (Method Overloading) به شما اجازه می‌دهد که دو متد با نام یکسان تعریف کنید که دارای امضا و تعداد پارامترهای مختلف هستند. برنامه از روی آرگومان‌هایی که شما به متد ارسال می‌کنید به صورت خودکار تشخیص می‌دهد که کدام متد را فراخوانی کرده‌اید یا کدام متد مد نظر شماست. امضای یک متد نشان دهنده ترتیب و نوع پارامترهای آن است. به مثال زیر توجه کنید:

```
void MyMethod(int x, double y, string z)
```

که امضای متد بالا

```
MyMethod(int, double, string)
```

به این نکته توجه کنید که نوع برگشتی و نام پارامترها شامل امضای متد نمی‌شوند. در مثال زیر نمونه ای از سربارگذاری متدها آمده است.

```
1  #include <iostream>
2  using namespace std;
3
4  void ShowMessage(double number)
5  {
6      cout << "Double version of the method was called." << endl;
7  }
8
9  void ShowMessage(int number)
10 {
11     cout << "Integer version of the method was called." << endl;
12 }
13
14 int main()
15 {
16     ShowMessage(9.99);
17     ShowMessage(9);
18 }
```

```
Double version of the method was called.
Integer version of the method was called.
```

در برنامه بالا دو متد با نام مشابه تعریف شده‌اند. اگر سربارگذاری متد توسط C++ پشتیبانی نمی‌شد برنامه زمان زیادی برای انتخاب یک متد از بین متدهایی که فراخوانی می‌شوند لازم داشت. رازی در نوع پارامترهای متد نهفته است. کامپایلر بین دو یا چند متد همنام در صورتی فرق می‌گذارد که پارامترهای متفاوتی داشته باشند. وقتی یک متد را فراخوانی می‌کنیم، متد نوع آرگومان‌ها را تشخیص می‌دهد.

در فراخوانی اول (خط ۱۶) ما یک مقدار `double` را به متد `ShowMessage()` ارسال کرده‌ایم در نتیجه متد `ShowMessage()` (خطوط ۷-۴) که دارای پارامتری از نوع `double` اجرا می‌شود. در بار دوم که متد فراخوانی می‌شود (خط ۱۷) ما یک مقدار `int` را به متد `ShowMessage()` ارسال می‌کنیم متد `ShowMessage()` (خطوط ۱۲-۹) که دارای پارامتری از نوع `int` است اجرا می‌شود. معنای اصلی سربارگذاری متد همین است که توضیح داده شد. هدف اصلی از سربارگذاری متدها این است که بتوان چندین متد که وظیفه یکسانی انجام می‌دهند را تعریف کرد.

## بازگشت (Recursion)

بازگشت (Recursion) ، فرایندی است که در آن متد مدام خود را فراخوانی می‌کند تا زمانی که به یک مقدار مورد نظر برسد. بازگشت یک مبحث پیچیده در برنامه نویسی است و تسلط به آن کار را حتی نیست. به این نکته هم توجه کنید که بازگشت باید در یک نقطه متوقف شود وگرنه برای بی نهایت بار، متد، خود را فراخوانی می‌کند. در این درس یک مثال ساده از بازگشت را برای شما توضیح می‌دهیم. فاکتوریل یک عدد صحیح مثبت ( $n!$ ) شامل حاصل ضرب همه اعداد مثبت صحیح کوچک‌تر یا مساوی آن می‌باشد. به فاکتوریل عدد ۵ توجه کنید.

$$5! = 5 * 4 * 3 * 2 * 1 = 120$$

بنابراین برای ساخت یک متد بازگشتی باید به فکر توقف آن هم باشیم. بر اساس توضیح بازگشت ، فاکتوریل فقط برای اعداد مثبت صحیح است. کوچک‌ترین عدد صحیح مثبت ۱ است. در نتیجه از این مقدار برای متوقف کردن بازگشت استفاده می‌کنیم.

```
#include <iostream>
using namespace std;

long Factorial(int number)
{
    if (number == 1)
        return 1;

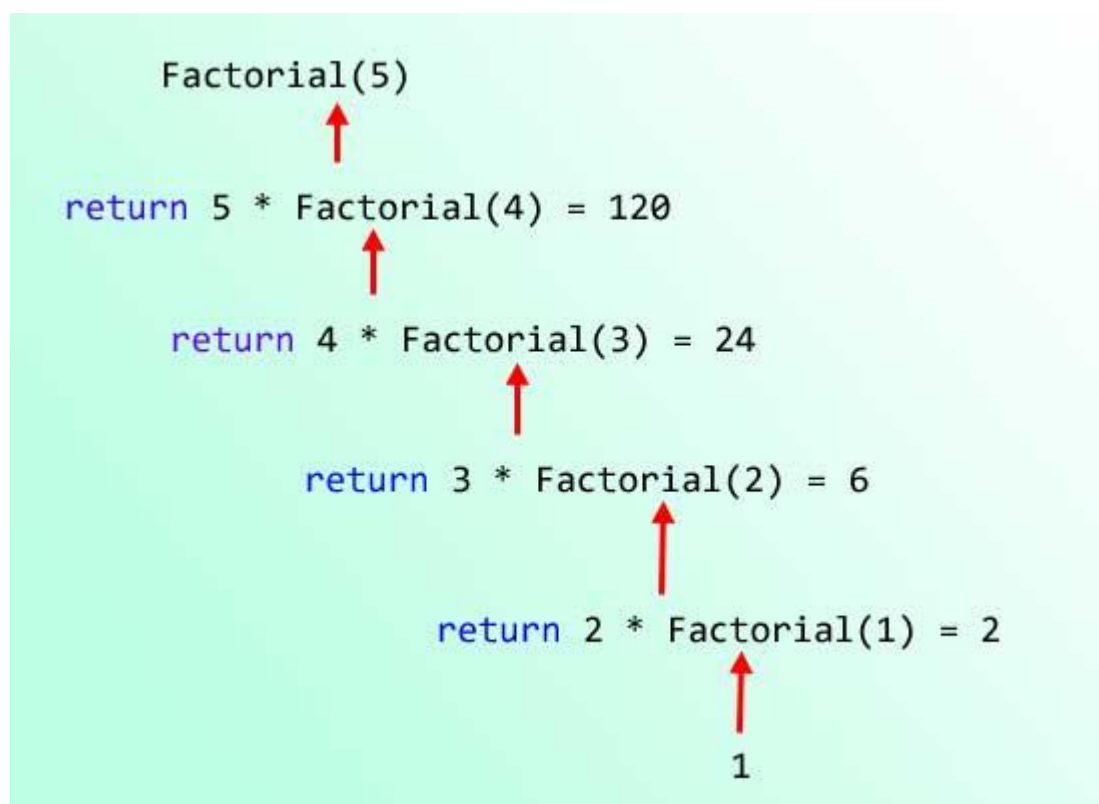
    return number * Factorial(number - 1);
}

int main()
{
    cout << Factorial(5);
}
```

120

متد مقدار بزرگی را بر می‌گرداند چون محاسبه فاکتوریل می‌تواند خیلی بزرگ باشد. متد یک آرگومان که یک عدد است و می‌تواند در محاسبه مورد استفاده قرار گیرد را می‌پذیرد. در داخل متد یک دستور `if` می‌نویسیم و در خط ۷ می‌گوییم که اگر آرگومان

ارسال شده برابر ۱ باشد سپس مقدار ۱ را برگردان در غیر اینصورت به خط بعد برو. این شرط باعث توقف تکرارها نیز می شود. در خط ۹ مقدار جاری متغیر number در عددی یک واحد کمتر از خودش (number - 1) ضرب می شود. در این خط متد Factorial خود را فراخوانی می کند و آرگومان آن در این خط همان number - 1 است. مثلاً اگر مقدار جاری number 10 باشد یعنی اگر ما بخواهیم فاکتوریل عدد ۱۰ را به دست بیاوریم آرگومان متد Factorial در اولین ضرب ۹ خواهد بود. فرایند ضرب تا زمانی ادامه می یابد که آرگومان ارسال شده با عدد ۱ برابر نشود. شکل زیر فاکتوریل عدد ۵ را نشان می دهد.



کد بالا را به وسیله یک حلقه for نیز می توان نوشت .

```

factorial = 1;
for(int counter = number; counter >= 1; counter--)
    factorial *= counter;
  
```

این کد از کد معادل بازگشتی آن آسان تر است. از بازگشت در زمینه های خاصی در علوم کامپیوتر استفاده می شود. استفاده از بازگشت حافظه زیادی اشغال می کند پس اگر سرعت برای شما مهم است از آن استفاده نکنید.

برای مشاهده سایر مطالب به سایت زیر مراجعه فرمایید:

[www.w3-farsi.com](http://www.w3-farsi.com)