Math Review

This section is intended for quick math review only. We will assume that you've a knowledge of linear algebra and trigonometry already but just want a quick review.

If you don't have these math skills yet, no worry! You may follow up on these good free Linear Algebra Lectures at the following sites and you will catch up in no time:

http://ocw.mit.edu/OcwWeb/Mathematics/18-06Spring-2005/VideoLectures/index.htm
http://mathworld.wolfram.com/topics/LinearAlgebra.html
http://joshua.smcvt.edu/linearalgebra/

Useful Trigonometry Reference

<table>
<thead>
<tr>
<th>tan x = sin x / cos x</th>
<th>sin(x+y) = sinxcosy + cosxsiny</th>
</tr>
</thead>
<tbody>
<tr>
<td>sin(-x) = -sin x</td>
<td>cos(x+y) = cosxcosy + sinxsiny</td>
</tr>
</tbody>
</table>
\[
\cos(-x) = \cos x
\]
\[
\sin x \cdot \cos x = 1
\]
\[
\sin(\pi x) = -\sin x
\]
\[
\sin 2x = 2\sin x \cos x
\]
\[
\cos(x + \pi) = -\cos x
\]
\[
\sin a/A = \sin b/B = c/C
\]
\[
\cos(x + \pi) = -\sin x
\]
\[
C = A + B - 2AB\cos c
\]

### Angle Degrees Chart

<table>
<thead>
<tr>
<th>Angles in Degree</th>
<th>Sine</th>
<th>Cosine</th>
<th>Tangent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>1</td>
<td>0.0175</td>
<td>0.9998</td>
<td>0.0175</td>
</tr>
<tr>
<td>2</td>
<td>0.0349</td>
<td>0.9994</td>
<td>0.0349</td>
</tr>
<tr>
<td>3</td>
<td>0.0523</td>
<td>0.9986</td>
<td>0.0524</td>
</tr>
<tr>
<td>4</td>
<td>0.0698</td>
<td>0.9976</td>
<td>0.0699</td>
</tr>
<tr>
<td>5</td>
<td>0.0872</td>
<td>0.9962</td>
<td>0.0875</td>
</tr>
<tr>
<td>6</td>
<td>0.1045</td>
<td>0.9945</td>
<td>0.1051</td>
</tr>
<tr>
<td>7</td>
<td>0.1219</td>
<td>0.9925</td>
<td>0.1228</td>
</tr>
<tr>
<td>8</td>
<td>0.1392</td>
<td>0.9903</td>
<td>0.1405</td>
</tr>
<tr>
<td>9</td>
<td>0.1564</td>
<td>0.9877</td>
<td>0.1584</td>
</tr>
</tbody>
</table>

### Vector

Vector is an object that has magnitude with size, length, points, etc. and a direction. Vector doesn’t have a position.

Vector Operations:
\[\{Ux, Uy, Uz\} + \{Vx,Vy,Vz\} = \{(Ux + Vx), (Uy + Vy), (Uz + Vz)\}\]

Ex: \([1,2,3] + [4,5,6] = [(1+4),(2+5),(3+6)] = [5,7,9]\)

c\{Ux, Uy, Uz\} where c is scalar(number) = \[c*Ux, c*Uy, c*Uz\]

Ex: \(4[1,2,3] = [4*1,4*2,4*3] = [4,8,12]\)

Dot Product:

\[\{Ux, Uy, Uz\} \cdot \{Vx, Vy, Vz\} = [(Ux * Vx) + (Uy * Vy) + (Uz * Vz)]\]

where as \(U \cdot V = |u||v| \cos \theta\) such that \(|u| = \text{Square}(U \cdot V)\); 

Unit Vector: \(|V|=1\)

Cross Product: The cross product of two vectors return a vector that is perpendicular to the two.

\(U \times V = [(Uy*Vz - Uz * Uy), (Uz*Vx - Ux * Uz), (Ux*Vy - Uy * Ux)]\)

Example:

\(U \times V\) where \(U = [1,2,3]\) and \(V = [4,5,6]\)

\(|x, y, z| \quad |x, - , -|\)

\(|1, 2, 3| = |-, 2, 3| \quad \text{where - is omitted numbers and won't be count in this x operation.}\)

\(|4, 5, 6| \quad |-, 5, 6|\)

this will give you the first value of the answer which is \(x\) by doing the dot product of \([2 \cdot 6] - [3 \cdot 5] = -3\) for \(y,\)

\(|x, y, z| \quad |-, y , -|\)

\(|1, 2, 3| = |1, -, 3| \quad \text{where - is omitted numbers and won't be count in this x operation.}\)

\(|4, 5, 6| \quad |4, -, 6|\)

this will give you the second value of the answer which is \(z\) by doing the dot product of \([3 \cdot 4] - [1 \cdot 6] = 6\)
for z,

$$\begin{vmatrix}
  x, & y, & z \\
  1, & 2, & 3 \\
  4, & 5, & 6
\end{vmatrix} = \begin{vmatrix}
  1, & 2, & - \\
  4, & 5, & -
\end{vmatrix}$$

where - is omitted numbers and won't be count in this x operation.

this will give you the second value of the answer which is z by doing the dot product of $[1 \cdot 5] - [2 \cdot 4] = -2$

And our answer is $$[-3, 6, -2]$$

**Matrix**

Matrix is for to store translation, scaling and rotation of the object. Each row or column of the matrix composes of a vector. Each number in the matrix refers to a component.

Adding, Subtraction, and Multiplying a Scalar with a Matrix is same like Vector where you either add or subtract two vectors together.

Matrix looks like this:

11, 12, 13, 14, 15  
21, 22, 23, 24, 25  
31, 32, 33, 34, 35  
41, 42, 43, 44, 45

Remember that you can only perform the operations above the two matrix have the same dimension Matrix4x4 + Matrix4x4 etc..

When multiplying the two matrices, you perform the row X column and the inner dimension must match.

Ex: Matrix2x4 * Matrix 4x2 is fine

but

Matrix2x4 * Matrix 2x4 is not.

The identity matrix is where you can use to reset its orientation to 1.

We will explore matrix in greater detail in the class.

**Plane**
In general, the equation of the plane is $ax + by + cz = d$ where $[a,b,c]$ is the normal vector. Plane is very important concept in our 3D graphic application. Make sure you’ve grasp the concept below.

The following is some useful operation for plane.

**Problem 1:**

Given: A point $P = (1,1,1)$ and a vector, $N = [1,2,3]$. Find a plane through $P$ perpendicular to $N$ (where $N$ is a normal).

Answer:

To do this, let's calculate:

$XP$ where $X$ is some vector $[x,y,z]$

First,

$XP = P - X = [1,1,1] - [x,y,z] = [1-x,1-y,1-z]$

Second,

$N \cdot XP = [1,2,3] \cdot [1-x,1-y,1-z]$

after doing the dot product of those two, you know have $x + 2y + 3z = 6$; This is your plane equation!

**Problem 2:**

Given: three points $P = (1,1,1)$, $Q(0,1,0)$, and $R=(1,1,-5)$. Find a plane through these points.

Answer:

First, let's find the two vectors on these points

$PQ = P - Q = (1,0,1)$

$QR = R - Q = (1,0,-5)$

Now, let's find the normal based on these two vectors by doing the cross product of the two.

$N: PQ \times QR = [0,6,0]$

Finally, perform
Problem 3:

Given: The plane $x + 2y + 3z = 6$ from problem #1 ($N = [1,2,3]$, $P=(1,1,1)$). Find the distance from the origin $O=(0,0,0)$ to the closest point on the plane.

Answer:

Perform the following:

$OP: \ P - O = [1,1,1]$

distance: $OP \cdot \frac{N}{|N|} = \frac{6}{3.74} = 1.60$

Problem 4:

Given: The plane $x + y + z = 10$. Also, two points $A=(1,-2,0)$ and $B=(-2,1,0)$. Find whether these two points $A$ and $B$ are on the same side or opposite side of the plane.

Answer:

From the equation, we have the normal

$N = [1,1,1]$

Point on plane: We pick $P = (0,0,10)$. We just pick a point on the plane based on the equation. It can be $(10,0,0)$ or $(0,10,0)$.

$P = (0,0,10)$

Next, perform

$PA = A - P = (1,-2,-10)$

and

$PB = B - P = (-2,1,-10)$

Then, we do the dot product for both points with the normal $N$

$PA \cdot N = -11$
PB • N = -11

Both points have the same sign, so yes these points are on the same side of the plane.

The key idea is that if A and B are on the opposite side, PA • N and PB • N will have the opposite signs. Otherwise, they’ll be on the same side.
Welcome to the Introduction to C++! You're about to embark on the exciting journey to the C++ programming with an emphasis in game programming. If you're already an experienced C/C++ programmer, this course won't be too exciting for you. If you're totally a newbie, you will no longer be one after you've completed this set tutorials. One word before we go on: C++ is a huge language and has many features and block of codes to know and cover. The best ways to learn C++ are to buy a C++ book for references or learn more features that our tutorials don't cover, to practice and code c++ (if you're totally a newbie, recommend 2 hours a day to practice if you can till you get bored of the fundamentals), and lastly don't give up or loose the passion for it. A motivation and passionate are the key to your success. Try to motivate yourself if you get lost or bored of coding. That's enough of blah blah! Let's get going....

A Brief Introduction to Computer Games Fundamentals

Brief History of Computer Games:

The history of computer games has seen the development towards more complexity and flexibility. Today's games are altogether more unpredictable and 'open' than was the norm just 1-15 years ago. But there are also considerable parallels between the newest 3D-shooters and their digital forefathers.

In 1952, A.S. Douglas wrote his Ph.D degree at the University of Cambridge on Human Computer interaction. Douglas created the first graphical computer game - a version of Tic Tac Toe. The game was programmed on a EDSAC vacuum-tube computer, which had a cathode ray tube display.

William Higinbotham created the first video game ever in 1958. His game, called "Tennis for Two", was created and played on a Brookhaven National Laboratory.

Spacewar is considerate one of the first computer games. In 1962 Harvard employee Stephen Russell constructed a computer game that was revolutionary for its time. Spacewar was the first game intended for computer use. Action games like Spacewar were well suited for quick battles and became the foundation for a rapidly growing industry. This industry mostly aimed at the so-called arcade machines. With hits like Space Invaders, computer games were soon moved out of the dimly lit arcades an into cafes and restaurants.
In the 1980's, sales from interactive entertainment rivaled those of the film industry (Sales: the game industry exceeded the film industry in the year between 2001-2002). Around 1980 a range of alternative genres began to appear. This happened primarily as a consequence of an increased focus on gaming consoles.

In 1976, the first adventure games were born. Later in 1980's the strategy games became very popular following in the wake of successes such as Prirates (Microprose, 1987) and SimCity (Maxis, 1987).

The CD-ROM made it possible to develop audiovisual elements without the constrictions of highly limited storage media. The games have become bigger and more 'photo realistic'. For a while game producers drew huge inspiration from movies but this tendency seems to have faded away by itself.

Computer games are now very often played on networks. This can be seen as a return to the social element of the arcades. From Lan and online multiplayer games make the gaming experience more unpredictable and exciting.

**Game Fundamentals:**

**What is a game?**

A computer game is a software program in which one or more players interact and make decisions through the control of game objects and resources, in pursuit of a goal.

**Type of Games:**

**Arcade games**

- the reaction speed is the most important aspect of the game. Typical examples are scrolling shooters and some maze games.

- these games are relatively easy to make and normally 2D are good enough

- a particular type of arcade games is the pinball game. These are a bit harder to create because you need natural ball movement.

**Puzzle games**

- clever thinking is the most important aspect. Many maz games are actually more based on puzzle solving rather than on reaction speed.

- other examples include board games and sliding puzzles.

**Role Playing Game (RPG)**
- you steer a character through a difficult world. Typical examples are Diablo, Baldur's Gate etc..

- the most important part of such a game is the development of the character. Meaning, a player controls the learning of new skills, becoming more powerful, and finding additional and better weapons or magic. At the same moment the opponents become more powerful as well

- rpg games create from 2D to 3D

- rpg games are harder to make and require rich of content

Adventure games

- the storyline is curcial. Most adventure games are largely 2D in the 90s and mostly 3D in today.

- the difficulty in creating an adventure game does not lie in the actions, but in creating an interesting, funny, and surprising story line, and in creating the corresponding artwork

First person shooters

- fast paced action and reaction speed

- need a 3D world to create the feeling of being there

- famous examples are Doom, Quake etc..

Third person shooters

- the player directly controls a game character through a hostile world

- example is Tomb Raider (which you will be making in quarter 11 or 12)

- fast action and discovering the game world

- good storyline and borrow aspects from adventure games

Sport games

- as we all know, examples are football, baseball etc...

Racing games

- a special type of a sport game because there are so many of them they deserve a category of their own.

- racing games can be both 2D and 3D
- one of the major challenges when making a racing game is to create convincing racing of the computer controlled cars

**What is a good game?**

Some games become extremely popular, making their creators millionaires, while other games, that look almost the same, become miserable failures. There is also a large collection of games that we never see because they were cancelled halfway of production and their creators went bankrupt.

What makes a game winner?

- a game involves players
- playing a game is about making decisions
- playing a game is about control
- good use of game objects and resources
- a game needs a goal
- good balancing of the game
- good storyline and characters design
- don't forget about good audio and graphic effects

**A Brief History of C/C++**

Who is the creator(s) of C? Brian Kernighan and Dennis Ritchie, created a new language called C in 1970. The C language became quickly spreading and using by many other programmers. C was intended for a procedure oriented language, which is the programmer's job to depict the data and then implements the steps to manipulate that data.

C++ language started in around 1980s by Bjarne Stroustrup, at Bell Labs. What he did was adding some new features to the C language. The most important feature he added was the 'classes' feature. This ways it allows programmer to organize and process their data and information more effectively. There are many more history about C++, and we don't go over them in this tutorial. However, you can check them out at [http://www.research.att.com/~bs](http://www.research.att.com/~bs) site. It has a lot of information related to the history.

**Getting Started....**
Note: You will need a C++ compiler (which is a program for you to build and run C++ codes). You can get some of the free compiler(s) at Borland, DJGPP, DevC++, DigitalMars C++, MinGW, TenDRA, IntelC++, AppleC++, WatCom, and Microsoft Visual C++/.NET. For this class, you can use the Microsoft Visual C++ 6 or .NET or G++ because lot of game companies use those compilers. You can request a free version of Introductory version of MSVC++6 or demo on .NET 2003 at http://msdn.microsoft.com/vstudio/

Before we dive into any codes, please have your the compiler install and launch it. Now, we want to create a workspace, project and file, the best place way to learn how to do this is to use the compiler's manual or MSDN, which has a large collection of notes, codes, and tutorials. Here are some supported compilers you can use in this tutorial:

**Setting Up WorkArea/Project Files:**

**G++ for Cygwin**

To setup and workspace for G++ in Cygwin,

1. Of course, you need to install the program. When you install the program, make sure you select the full packages

2. Go ahead and launch the cygwin program

3. You will need to setup the workspace for all of your program. To do this:

   a. Type in, (note $ sign below is the command prompt, no need to type it in)

   $ mkdir cplusplus

   the command above will let you create a folder called "cplusplus". This is the folder where you will put all of your class files.

   b. Next, you will need to create a c++ source file for testing purpose. You can vi or emacs or gedit editors to do so.

   For testing purpose, open the editor and copy the codes below:

   *****Below is the test code *****

```cpp
// my first c++ program
#include <iostream>
using namespace std;
int main()
{
```
Go ahead and save the file as hellp.cpp

c. Next, we need to compile the file we just made so that we can run it. To do this, type in the command below:

For UNIX, use this:

$ CC -g -ohello hello.cpp

The -g allows to turn on the debug feature, and the -ohello lets the compiler knows that this program is called 'hello'

For G++ in Cygwin, use this:

$ g++ -g -Wall -ohello hellp.cpp

The options are the same as above except we have the -Wall, which is to turn on the warnings feature.

d. After you've compiled, you can just run it by typing the program name in our case it's hello

$ hello
For MS Visual C++ 6.0

1. Click on File, New. You will see a pop up, then pick Projects, Win32 Console Application, type in the project name, specify the location, and click ok. A screen looks like below:

![New Project Window](image)

2. Next, click on an empty project, and click on "OK" button.
3. Next, you should see a workplace setup for you. Click on File, New, under Files tab, pick C++
Source file, and specify the file name like 'name.cpp' etc.... and click ok.
4. Copy the test code and press F7 to compile.
5. Note that at the bottom, you get to see the actual compile commands. This window will also tell you if there is any wrong with your code and at what lines. I will go into the debug mode later on.

6. Press Ctrl F5 to run the console, and press Enter to terminate.
For MS Visual .NET 2003

1. Start MS Visual .NET, click on File, New, Project, and a similar screen will show up below:


3. You see a screen below:
4. Highlight the Source Files folder, and click on File, Add New Item, choose C++ file (.cpp) like the screen below
Enter the file name, and click open.

5. Now, you're ready to copy the testing code above and save it, and press F7 to build it.

6. Press Ctrl F5 to run the console, and press Enter to terminate.
For Visual Studio .Net 2005 Express and Professional Version

1. Launch VS Studio application

2. Go to File -> New -> Project and pick Win32 -> highlight Win32 Console Application. If you're using professional version, be sure to expand out the Visual C++ Project tree. You should be at the screen below:

3. Fill in the project name and hit OK button.

4. You will see the Win32 Application Settings window. Choose Application Settings and check the Empty Project box
Hit Finish.

5. To add .cpp file to the project, right click on the Source Files folder in the explorer and choose Add -> New Item. Then you can just type in the name and hit Add.

6. To add .h file to the project, right click on the Header Files folder in the explore and choose Add-> New Item -> highlight the .h and give it a name. Hit Add
1) By now, you should have installed your Visual C++ Express. If not, please visit http://msdn.microsoft.com/vstudio/express/visualc/download/ for more information.

2) Next, if you haven't done so, install the Directx SDK at http://msdn.microsoft.com/directx/sdk/

3) Next, make sure you perform this installation to get your visual c++ express to work with win32 api at http://msdn.microsoft.com/vstudio/express/visualc/usingpsdk/default.aspx

4) Let's test it!

   - Download the 2D Engine sample in the Library. Launch the application and go to File - Create - Project, and you should have a similar screen below:

     ![Screen Shot]

     - Give a name and you should see a window below to choose "console application" and "empty project"
- click Finish

- right click on the source files and create new item - choose .cpp - and name it as "CMain.cpp"

- right click on the name of the project and choose properties

(click on the image to enlarge)

- Under C/C++, expand it to General and put in the path to your DirectX include directory under "Additional Include Directories". Example:
- then, go to Linker and find "Additional library directories" and put in your lib\x86 directx directory

Our is C:\Program Files\Microsoft DirectX SDK (October 2006)\Lib\x86
- then click on Input and put in this line

- click on Apply and OK

- One last thing, go to Tools - Options

(click on image to enlarge)

- Enter the path of Include and x86
Now you're done setting it up!

Alright, we should now have our workplace and project file setup, so let's get down to the code!

1. Basic structure of the first C++ Program

Go ahead and observe the following piece of code:

Code 1:

```
// my first c++ program
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello, this is my first C++ program";
    return 0;
}
```
Now, let's go over it.

The first (1) line in the code started with //my first C++ program.

// means comment. You can leave a comment or note in your code by using this symbol (double backslash). my first C++ program is the comment that we put in line 1. You can put as many as characters in one line until it reaches the limit. When you start a new line for comment, you will need to put in another // . Here's an example:

// my comment for this line here
// this is my new line for comment

You can also put the // after a code, but you can't put between it. Here's an example:

    using namespace std; //this is my comment for this line
    NOT
    using //this is my comment for this line using namespace std;

If you use or put // between the code, the compiler will read it as a comment and ignore whatever after //; and it won't execute the code you want to. Refer to the above example, the code in green will result in a comment, and the compiler isn't like it. It will be mad and crash your computer. No, I'm just kidding. It just gives you an error while you try to run it.

Here are some of the various commenting tricks or most common use:

/*****************************/
********** WARNING: this is an example of warning **********
********** box.
/*****************************/

//----------> Another, less important warning <----------

//>>>>>>>>>>>>>> Major section header <<<<<<<<<<<<<<<<<<<<

/*****************************/
* We use box comments to denote the beginning of
* a section or program usually contains
* Author: Name
* Change History:
/*****************************/
// simple comment explaining the next line in the code

/*
 * A medium-level comment explaining the next
 * dozen or so lines of code. Even though we don't have
 * the bold typeface we can **emphasize** 8 words
 */

Try to use comments in your code, in fact it's required in this class this you start out the box comments to start out your program.

Next, line 2 we have `#include <iostream>`

`#include` is pre-processor, which is an instruction to the compiler that takes care and loads whichever file needs for the code to perform. The compiler knows it is a system file, and therefore must be looked for in a special place, by the fact that the name is enclosed in `<>` characters.

In our case, we're asking the pre-processor, `#include`, to load up the file called `iostream.h`. The file `iostream.h` in our code was included in the `<>` characters. The `<>` characters around the name of the file tell the compiler to look in the system area for the file `iostream.h`. If we had given the name "myownfilecode.h" instead it would tell the compiler to look in the current directory. This means that we can set up libraries of our own routines and use them in our programs, a very useful feature. `iostream.h`, .h stands for header file. C++ comes with many header files like `iostream.h`, `cmath.h`, etc... We use `iostream` file for input and output streaming.

For instance:

```
#include <iostream> // include the file iostream.h for me
#include "myownfilecode.h" // this was the file I created and put some codes in here already
```

```
Note: If you're using old compiler, you might need to add the .h to iostream (eg..
iostream.h)
```

Next, line 3 we have `using namespace std;`

using is just a directive. `namespace` refers to the standard components of functions, variables, classes or emulations in std. In this case, using makes all std namespace available in the code. It ends with semicolon ; (which tells the compiler this is end of the code for this line, in other word, a terminating line of code). Normally, if we don't put `using namespace std;` then we have to type in the code in a long way. Instead of typing just `cout`, we have to type in `std::cout` (we will go over it soon). Here's what we mean by comparing the two codes below:
In the code 2, we took out the `using namespace std;` and we had to type in `std::cout` instead of just `cout`. Both examples above will display the same output. Whichever way you choose to code, it's up to you, but in this class we go the short way for now, which is using `namespace std;`.

Next, we have `int main()` in line 4. In C++, `main()` is our best friend. It's our main function for every C++ program except for windows programming. What it does is to process all of the commands, functions, and other necessary information you put in between the code and execute them. The syntax (or format) of `main()` function is

```cpp
int main()
{
    return 0;
}
```

It can also be

```cpp
void main()
{
    // your codes goes here
}
```

The `void main()` has no value or does not need to return any value, therefore, the `return 0;` in line 7 will not be there. So, in our example at line 4, we have `int` (stands for integer) that can return an integer value to the function, and `main()` is a function not responsible for taking any information from any function that calls it. In line 5, we have `{` brace. It's the open function of `main()` that allows us to put block of code below it. Brace is the same style like `()` as brace comes with the open `{` and close brace `}` (at line 9).

In line 6, we have `cout << "Hello, this is my first C++ program";` *cout* is a class in `iostream.h` file that reads whatever we typed in the line and displays it to the screen. What is a class? we will go over it in the later classes. Again, `iostream` file stands for `Input-Output-Stream`. In our case, it displays a string of characters in the " " quotation, which displays whatever inside of it to the screen. Again, the semicolon `;` tells the compiler that this is the terminating end of this function (just like a period in the end of sentence). Then, we have at line 7, `return 0;` It will just a return a value of Zero to the operating system that either success or fail. return 0 means for success, and any other number like return 1 means fail (we will go into this more later in the class). Finally, we put } brace for the closing of function at line 9.
Now, let's type in and compile our code and run it. You should see similar output on the screen below:

Note that you see the "Press any key to continue" next to the word 'program'.

You can bring it to the second line by putting the \n (means new line) before the ending " quotation line 6; so in our code at line 6, it should look like

    cout << "Hello, this is my first C++ program
";

or you can use the endline object like this:

    cout << "Hello, this is my first C++ program" << endl;

Now, try to run again and you should get a similar output below
Yes, it’s the same output but it looks nicer with the new line because we put in the escape sequence codes. If somehow your compiler just pop up and 'whooopp' and disappear right infront of you before you can chance to view it, you will need to put

```cpp
cin.get();
```

before return 0; line code.

This will stop the screen from disappearing away from you.

Yes, in iostream, there is an object call "cin". As you can tell, cin is just the opposite of cout. cin takes the input and store it to the variable. When you use the cin object to take an input data, use the double right arrow (right shift operator `>>`).

Example:

```cpp
cout << "Enter your password here:____\b\b\b\b\b\b";

int takeInput;

cin >> takeInput;

cout << "display takeInput " << takeInput;
```

The cin will store what ever integer number that a user put on the screen and stores it in the variable of takeInput. Simple as that!
You've seen the usage of 'n and 'b in our codes. You can also use other few escape sequence codes in our code.

Here's the few escape sequence codes you can use:

<table>
<thead>
<tr>
<th>Name</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>newline</td>
<td>\n</td>
</tr>
<tr>
<td>horizontal tab</td>
<td>\t</td>
</tr>
<tr>
<td>vertical tab</td>
<td>\v</td>
</tr>
<tr>
<td>backspace</td>
<td>\b</td>
</tr>
<tr>
<td>carriage return</td>
<td>\r</td>
</tr>
<tr>
<td>backslash</td>
<td>\</td>
</tr>
<tr>
<td>question mark</td>
<td>?</td>
</tr>
<tr>
<td>single quote</td>
<td>'</td>
</tr>
<tr>
<td>double quote</td>
<td>&quot;</td>
</tr>
</tbody>
</table>

To better use those escape sequences codes above, open a new .cpp file and try to display the following paragraph on the screen by using cout and the escape sequences.

DeWitt, 54, 5' 8", is the only one of the trio who participated in the gossipy movie that depicts Somers in a less than flattering light.

Ritter, also 54, currently stars in the 'ABC' comedy "8 Simple Rules for Dating My Teenage Daughter." He declined comment and smiled (\-\/).

"I know John is thrilled about it because he told me," DeWitt said. "He's very comfortable with it. I think he saw what I saw — really, really intensely, sensitive, personal things that were very public are going to be public once again."

Save the file and run it. You should see the same passage above on the screen.

2. What's a variable? How to declare it?

Variable in C++ Programming stores a value in memory for use. One of the way to think a variable in C++ is like when you have a child, you give a child a name of your choice so that you can refer to that child later. Now, the child has a name, an example: Jack. When you name the child "Jack", you declare that this child has a name "Jack" and your brain or memory automatically store that name in a slot of your memory or brain. Everytime you want to call the child's name you will use the word "Jack". The child's name that stored in your memory can be used and pass to whichever function of the brain that calls or uses it. This is the same thing as a variable in C++. You declare a variable with a name and type(s), and then you can use the variable later in your code. As you declare and initialize a variable, it gets to store in a memory address. Types can be integer, double, float, or long double etc... Here's the syntax for declaring a variable:

```cpp
int myVariable;
```
myVariable is now an integer. A quick note, C++ is case sensitive. So, int myVariable and int myvariable are two different variables.

Variables can be assigned a value such as number or text. You can assign a value to it by using the = sign, which is called assignment operators.

```
int myVariable = 20; // A C way to initialize and declare a variable

int myVariable(20); // A C++ way to initialize and declare a variable, which you must use more often in this class.
```

myVariable now has the value 20. What can you do when you've declared and assigned a value to the variable? Well you can use it in your code to refer to it.

For instance:

```
#include <iostream>

using namespace std;

int main() {

int myVariable(20); //initialize and declare myVariable as an integer with the value of 20

cout << myVariable << endl; // endl stands for endline, something like \n character

return 0;
}
```

This will display "20" on the screen. You can use the myVariable for other things like subtract, minus, add or multiply or you can assign that variable.

More, when you declare a variable, make sure you don't use the same name that C++ reserved, which is called Keywords. C++ comes with many keywords like if, for, bool, int, etc..... The way to avoid is to use upper case in your variable (example myVariable or MyVariable etc..)

Variable Naming:

- variable name must start with letters or underscore but no numbers

- variable are case sensitive

- keywords can't be use as a variable name
Variable has many different types. You can also declare a variable as short, unsigned short, bool, char, double etc... Let's take a look at the table below:

Table 1.1

<table>
<thead>
<tr>
<th>Name of Types you can use to declare</th>
<th>size of the type</th>
<th>values that it can hold</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>1 byte</td>
<td>true or flase</td>
</tr>
<tr>
<td>double</td>
<td>8 bytes</td>
<td>2.2e-308 to 1.8e308</td>
</tr>
<tr>
<td>float</td>
<td>4 bytes</td>
<td>1.2e-38 to 3.4e38</td>
</tr>
<tr>
<td>char</td>
<td>1 byte</td>
<td>256 characters</td>
</tr>
<tr>
<td>int</td>
<td>4 bytes</td>
<td>-2,147,483,648 to 2,147,283,648</td>
</tr>
<tr>
<td>unsigned int</td>
<td>4 bytes</td>
<td>0 to 4,294,967,295</td>
</tr>
<tr>
<td>long int</td>
<td>4 bytes</td>
<td>same as int</td>
</tr>
<tr>
<td>unsigned long int</td>
<td>4 bytes</td>
<td>same as unsigned int</td>
</tr>
<tr>
<td>short int</td>
<td>2 bytes</td>
<td>-32,768 to 32,768</td>
</tr>
<tr>
<td>unsigned short int</td>
<td>2 bytes</td>
<td>0 to 65,535</td>
</tr>
</tbody>
</table>

When to use those types you asked? Just remember that signed type can hold positive and negative values, and unsigned type can't hold negative value.

You can use **bool** if you need to return a value true or false

**Example:**

```cpp
bool EnemyBossLevelOneDie = true;

bool EnemyBoxssLevelTwoDie = false;
```

You use **double** or float if you need large numbers, and it can only hold up to 19 digits, also decimal numbers. Mostly, you will use double more often than any others.

**Example:**

```cpp
double GunsAmmo(24.56);

double BossPower(98.67);
```

You use **char** when you need to declare a variable as character (**char Word;**). Char can only hold one character.

**Example:**
char ch;

cout << "Enter a letter: "; //ask a user for a character

cin >> ch; //take input of the character

char c = 'T'; //assign letter T to c char variable, remember there is only a single quote between T

int i = c; //assign c char variable above to i integer to get the ascii number for letter T

You use `float` when you want to use a decimal number like 6.5 or 3.14566 and 122345.34 etc... or 3.45E5. When you see the float number 3.45E5 (E notation), it means 3.45 multiplies or times by 100,000 or one followed by 5 zeros. The number, 3.45E-5, means 3.45 divides by the power of 5 or you moves the decimal to the left 5 places starting at 3. so the answer for this is 0.0000345. The numbers 3.45 and 3.45E5 are a double type. If you have a constant and want to assign it to a float type, you will need f or F suffix. A constant variable is a unchanged variable or a variable you can't change the data in it.

Example:

float PlayerOneHealthPercent(34.55);

float NumberOfKills(3.45e5); //you can use E or e notation

float NumberOfKills(3.45f); //this is a float constant

You use `int` when you want to use an integer. Remember int can only hold up between -2,147,483,648 to 2,147,283,648.

Example:

int MagicCollection;

int NumbersofMagic(9); //assign 9 to numbersofMagic integer.

You use `short`, `unsigned` or `long` when you need a number in the range listed in the table 1.2. Just remember, unsigned can't hold any negative numbers, but short can. Also, you use long for bigger integer.
unsigned int AgeofPlayer(-12); //this is wrong because unsigned can not hold any negative numbers
unsigned int AgeofPlayer(12); //this is correct
short months;
long Numb(560.334);

More ways to define variable:

**Const**

Constants are variable that cannot be changed while the program is running.

Example use of const:

```c
const int NumbersofMagic(9); //assign 9 to numbersofMagic integer as constant
```

Later in your program, if you happen to change the NumberofMagic, the compile will yell at you with those messages.

**Register**

Register tells the compiler to store the variable in the processor register not the normal physical memory.

Example of register use:

```c
register long Numb(560.334); //tell the compiler to store Numb in the register for faster use
```

**Redefining types**

You can use typedef to redefining variables to other alias or names.

Example use of typedef:

```c
typedef int SupInt;
```

When you declare that line above, you can now use the SupInt instead of int in your code.

When we talk about "assign", we mean that giving a value to something.
Things to remember when you name your variable:

- make sure it has a meaning name not just like myVariable (I used above). Use a good name for the variable!

- don't use keywords. The way you can avoid this by naming your variable as uppercase and using underscore

- don't start out with a number

- does not contains a $ sign like boss$one

- does not contains a space like boss one

Since we were talking about "assign" things, in C++, there is something called "Arithmetic Operators". What it means is the symbol of plus, minus in math that we've already seen.

+ means add (usage: myNumber1 + myNumber2)

- means subtract (usage: myNumber1 - myNumber2)

/ means divide (usage: myNumber1 / myNumber2)

* means multiply (usage: myNumber1 * myNumber2)

% means a modulus or remainder of dividing the first by the second numbers. It doesn't take a double variable. (usage: myNumber1 % myNumber2)

Note: myNumber1 and myNumber2 are variables, and don't forget to put semicolon at the end of code ;

like: myNumber1 = myNumber1 + myNumber2;

Other way you can do something like above:

myNumber1 += myNumber2;

The effect is the same.

Here is an example of using the Arithmetic Operators

#include<iostream>

using namespace std;

int main() {


int myNumber1(20); //initialize myNumber1 variable and assign it to a value of 20
int myNumber2(40); //initialize myNumber2 variable and assign it to a value of 40

int resultAdd, resultSubstract, resultDivide, resultMultiply, resultModulus; //another way to declare variables but not recommended

resultAdd = myNumber1 + myNumber2;
resultSubstract = myNumber1 - myNumber2;
resultDivide = myNumber1 / myNumber2;
resultMultiply = myNumber1 * myNumber2;
resultModulus = myNumber1 % myNumber2;

cout << "This is result from Adding myNumber 1 to myNumber2: " << resultAdd << endl;
cout << "This is result from Subtracting myNumber 1 to myNumber2: " << resultSubstract << endl;
cout << "This is result from Dividing myNumber 1 to myNumber2: " << resultDivide << endl;
cout << "This is result from Multiplying myNumber 1 to myNumber2: " << resultMultiply << endl;
cout << "This is result from Modulus myNumber 1 to myNumber2: " << resultModulus << endl;

return 0;
}

Open a new .cpp file, type in the code above and save it. Compile it and run it.
Let's look at this code below:

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

void displayFunctionA (); //prototype for displayFunctionA
int somethingFunctionB (int); //prototype of somethingFunctionB (int)
```

You should see a similar output like the above.

Let's look at this code below:

```cpp
cout.setf (ios_base::fixed, ios_base::floatfield);
```

Basically, this line is set a fixed point notation so that you can see the exact number on the output. Just place the code in the main() block somewhere before you use any type numbers variable and want to see a precise number.

Next is the user-defined functions. We will go in details on Functions in later lesson, but for now, we just want to know what is is and how to create one. You will need this for your homework assignments.

User-defined function is just a sub script or mini script within one big script. It's like a big corporation breaking down into departments. Each departments has its own role and jobs to do or function. This is the same thing like user defined in C++.

A quick view on function (we will go into more details later lessons).

How do you define a function? Lets take a look...

```cpp
#include <iostream>
using namespace std;
void displayFunctionA (); //prototype for displayFunctionA
int somethingFunctionB (int); //prototype of somethingFunctionB (int)
```
int main()
{
    cout << "this is my displayFunctionA called:\n";

displayFunctionA(); //called displayFunctionA

    int num;

    cin >> num;

    int Result = somethingFunctionB (num); //let the somethingFunctionB do the calculating

    cout << "this is my second function with return value I entered:" << Result << "\n";

    return 0;
}

void displayFunctionA ()
{
    cout << "Please enter a number below:\n";
}

int somethingFunctionB (int b)
{
    return 6 * b;
}

As you can see the above code, to define a function, first we need to write a prototype for it. A prototype is like a default name of the new function you're going to create. The prototype of a function usually places before the main() program. Look at the third line in the blue code above:

void displayFunctionA (); //prototype for displayFunctionA

This is our prototype. We create a new function called "displayFunctionA()" as a void that returns no value or nothing. Another prototype we have is the line:
int somethingFunctionB (int); //prototype of somethingFunctionB (int)

This time we create a function called "somethingFunctionB" that returns to us an integer. Note that the int in the braces (int), this is our default initialization of our new variable.

To use this functions, we call it in our main like this:

displayFunctionA(); //called displayFunctionA

If your function doesn't return nothing but do something, then you can do this:

int Result = somethingFunctionB (num); //let the somethingFunctionB do the calculating

or

somethingFunctionB (num); //let the somethingFunctionB do the calculating

So, that's a very brief introduction to functions. We will get into more in-depth later on.

After we've created the prototypes, we can now create the functions. The code in red above are our new functions we just created. Note that the void function doesn't have the return line because it returns no value. The int somethingFunctionB function returns us the result of 6 multiplies b variable.
Loops, Decisions, and Relational Expressions

<table>
<thead>
<tr>
<th>Overview:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Relational Operators</td>
</tr>
<tr>
<td>- Loops and Decisions</td>
</tr>
<tr>
<td>- Logical Operators</td>
</tr>
<tr>
<td>- Precedence summary</td>
</tr>
<tr>
<td>- Other control statements</td>
</tr>
<tr>
<td>- Practice Assignments</td>
</tr>
</tbody>
</table>

**Relational Operators**

What is a relational operator?

A relational operator evaluates two different variables with assigned values. It can test for whether either one of the variables is greater than the other or less than the other one or both variables are equal or not equal.

Here is the list of the relational operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>This is a GREATER THAN operator.</td>
</tr>
<tr>
<td>&lt;</td>
<td>This is a LESS THAN operator.</td>
</tr>
<tr>
<td>==</td>
<td>This is an EQUAL operator.</td>
</tr>
<tr>
<td>!=</td>
<td>This is NOT EQUAL operator.</td>
</tr>
<tr>
<td>&gt;=</td>
<td>This is a GREATER THAN OR EQUAL TO operator</td>
</tr>
<tr>
<td>&lt;=</td>
<td>This is a LESS THAN OR EQUAL TO operator</td>
</tr>
</tbody>
</table>

Note: Be-careful and don't confused with the assignment operator with the equal operator. The assignment operator has one equal sign, =, that assigns value into a variable. The equal operator, has two equal signs ==, examines and tests for if the value is equal to the other value.

We will get to use the relational operators in the loops soon.

**Decisions and Loops**

We will start out with the simplest decisions, **if - else - else if**. In C++, sometime you want the program to make a decision whether it should do this part of the program and stop or do that part of the program and continue on to other part of the program. This can be a one time decision.

Here's the syntax of the if statement:
if (expression)
{
    statements;
}

Example of an if:

#include <iostream>

using namespace std;

int main()
{
    int x(2); //assign x with a value of 2
    int y(3); //assign y with value of 3

    cout << "Testing value x = 2 vs y = 3." << endl;

    if(x == y)
        cout << "x equals to y" << endl;

    if(x > y)
        cout << "x is greater than y" << endl;

    if(x < y)
        cout << "x is less than y" << endl;

    if (x >= y)
        cout << "x is greater than or equal to y" << endl;

    if (x <= y)
        cout << "x is less than or equal to y" << endl;

    if (x != y)
        cout << "x is not equal to y" << endl;

    return 0;
}

The above example shows that we have two variables x and y. The x variable has the value of 2, and the y variable has the value of 3. Then the program has the 6 logical operators in six different if statements to check if x variable meet the condition of y variable. If it does, do the statement in our case it just reports "x is less than y" etc...Let's look at one example of the if statement.
if (x == y)
{
    cout << "x equals to y" << endl;
}

First of all, if is the keyword of C++. Remember the naming convention that we can't declare a variable with a keyword. This is one of them. Next, we have the expression, x == y. We want the program to evaluate if x is equal to y then reports x equals to y. In our case, it does not equal to y.

So, by looking at the program. Which of the if statement(s) will show when you run the program?

You can look at the output below.

Let's look at another if sample:
```cpp
using namespace std;

int main()
{
    int userNumber(0); // declare userNumber as an interger with the value of 0 for default
    int setNumber(6); // declare setNumber as an interger with the value of 6 for default // this will be using to test if the userNumber match
                          
    cout << "Please enter a number: ";
    cin >> userNumber;

    if (userNumber == setNumber)
    {
        cout << "you\'re the man !" << endl;
        cout << "you guess it correctly, the number is 6." << endl;
    }
    return 0;
}
```

The if sample 2 above asks the user to enter the number and tests if the userNumber is equal to setNumber. If it does, it will display the message.

What if the userNumber does not match the setNumber and you want to display the message to the user "sorry the number you entered does not match"? You can use the else statement. We called it " if - else" statement

The syntax is:

```cpp
if (expression)
{
    statement;
} else {
    statement;
}
```

Let's add the else statement to our If sample 2 program.

```cpp
/***************************************************************************/
* Program: If sample 2
* *
* Author:
* Date Written:
* Version:
* Change history: add the else statement to the if
* */
```
Look at the new part of the code here:

```cpp
} else {
    cout << "no number match" << endl;
}
```

We just added the else statement to display "no number match" if the user enters different numbers but 6. Now, let's look at the if - else if - else statement

The syntax is:

```cpp
if (expression)
{
    statement;
}
else if (expression)
{
    statement;
}
else
{
    statement;
}
```
if statements chained together in this fashion are called nested if's. Notice that as soon as a condition is found that matches the expression, evaluation stops and statements inside that if are executed. Let's take our first single if sample program and modify it to have else if statement:

```cpp
#include <iostream>

using namespace std;

int main() {
    int x(2); //assign x with a value of 2
    int y(3); //assign y with value of 3

    cout << "Testing value x = 2 vs y = 3." << endl;

    if (x == y)
        cout << "x equals to y" << endl;
    else if (x > y)
        cout << "x is greater than y" << endl;
    else if (x < y)
        cout << "x is less than y" << endl;
    else if (x >= y)
        cout << "x is greater than or equal to y" << endl;
    else if (x <= y)
        cout << "x is less than or equal to y" << endl;
    else if (x != y)
        cout << "x is not equal to y" << endl;
    else
```
As you can see, we added the else if and else statement in the program we had earlier. The result this time is different than the first one. First the program runs and displays the cout statement "testing value x = 2 vs y = 3". Then it reaches the if statement. The if statement starts out to evaluate at the first statement or beginning of the statement, which is to test for if x == y. The statement is false, and it then moves on the next one which is to test if x > y, else if (x > y). It is false again, and it then moves on the next. This time it tests for if x is less than y, else if (x < y), which is true. The program then displays the statement below that expression, which is " cout "x is less than y" << endl; ", then it stops and quit the if statement. It will skip the rest of the testing expressions and won't even bother to test it. There is another way to make it shorter but also using the if-else if - else statement which is to use the logical operators. We will cover it soon. Below the output of the program above.

```
{ cout << "x and y can't be tested" << endl; }
return 0;
}
```

**The For loop**

Next, we will cover the for loop. In C++, you use the for loop when you how many times you want the loop to run. For example if you want to display a number from 1 to 100. You can do it with 100 cout statement or with one for loop.

Here's the syntax of for loop

The syntax is:
Now, we have the syntax. Let's use it to display our 100 numbers.

```cpp
#include <iostream>

using namespace std;

int main()
{
    cout << "Display number 1 to 100 using for loop: " << endl;
    for (int i = 1; i <= 100; i++)
    {
        cout << i << endl;
    }
    return 0;
}
```

Let's look at for sample 1.

We have the for loop that counts the number from 1 to 100.

According to our syntax, we have declared an initialization expression which is int i = 1; This means that we have an integer i that has a value of 1. Then, we have our test expression, i <= 100; , to evaluate the number from 1 to 100. We then have the condition expression which is to increment the variable i each time it goes through.

Let's look at another sample of for loop.

```cpp
#include <iostream>

using namespace std;

int main()
{
    cout << "Display number 1 to 100 using for loop: " << endl;
    for (int i = 1; i <= 100; i++)
    {
        cout << i << endl;
    }
    return 0;
}
Here's the output for the above program:

```
Enter a number: 2
2
4
6
8
Press any key to continue
```

Notice that this time our for loop is incrementing the number of the user input. Here's how it works for (int i = 0; i < 10; i = i + userNumb). We initialize our i equals to zero, then we have i to evaluate for 10 time starting 0 to 9 (remember it excludes the 10). Then we have i adds with the userNumb and assigns it back to i. Basically, the for loop will count by the number directed by the user input.

Sometime, you can combine loop within loop. Here's an example of for loop within for loop
#include <iostream>
#include <iomanip> // setprecision

using namespace std;

int main ()
{
    long leftNumber;
    long rightNumber;

    for (leftNumber = 0; leftNumber <= 12; leftNumber++) // column
    {
        for (rightNumber = 0; rightNumber <= 12; rightNumber++) // row
        {
            // print out "left x right = product"
            int product = leftNumber * rightNumber;

            cout << leftNumber << " * " << rightNumber << " = " << product << endl;
        }
    
    cout << endl; // extra blank line
}

return 0;
}

The for loop within for loop sample above will display a multiplication table. Here's the output below:
Also, in the for loop, you may have more than one initialization expression and testing expression like:

```c
for (int i = 0, b = 3; i < 30; i++, m++)
{
    //do this code for me
}
```

Note: the only different in the multiple expressions in the for loop is the comma. You need the comma to separate two expressions in the for loop.

**The while loop**

The while loop evaluates the test expression till it reaches the condition within that expression and it will then stop and exit the loop. The while loop is good for when you want a certain section of the code or expression to be true and you don't know how many times you want the loop to run for.

Here's the syntax of the for loop.

The syntax is:
```cpp
while (test-condition) {
    statement;
    statement; //or more statements
}
```

Here's the example of while loop

```cpp
#include <iostream>

using namespace std;

int main() {
    int numb(6);
    int userInput;

    cout << "Enter a number: " << endl;
    while (userInput != numb) {
        cin >> userInput;
    }

    cout << endl;
    return 0;
}
```

Here's the output of the program:
The while loop sample 1 above tests for the expression, while (userInput != numb). First, we initialize the numb integer variable with the value of 6. Then, we have the while to ask the user to keep enter the number till it reaches the number 6. If the user does not enter the number 6, it will then keep asking the user to enter the number.

Here's another while loop sample: Remember the for loop that counts and displays the number from 1 to 100? Here's another way to do it by using the while loop.

```cpp
#include <iostream>

using namespace std;

int main()
{
    cout << "Display number 1 to 100 using while loop: " << endl;

    int i = 1;
    while (i < 101)
    {
        cout << i << endl;
        i++; // don't forget to increment by 1 or else you will go into
```
This time the while loop begins to test the condition, $i < 101$. It will then go through the loop for 100 times and exit at 101 time. The most important statement we added in this while loop is the update expression, $i++$. Don't forget to update the count each time the loop goes through or else it will stay at one number and keep looping till you terminate the program.

**The do-while loop**

The do while loop is the opposite of the while loop. The while loop evaluates the test expression at the beginning when the expression is false and goes through the body. Sometime if the while loop test expression is true then the body statement won't execute at all. The do-while loop lets the program to evaluate the body first before the test expression begins.

The syntax is:

```cpp
do
    { 
    statement;  
    statement;  //or more statements  
} while (test-expression); //note there is a semicolon at the end here
```

Let's look at our previous first while sample and turns it into the do while loop.

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

int main()
{
    int numb(6);
    int userInput;

    cout << "Enter a number: " << endl;
    do
    {
```
The above do-while loop has the same output with the while loop sample 1 we saw earlier. The different here is that the body gets to execute first before the while(userInput != numb); begins to evaluate.

Let's look at another do-while loop sample:

```cpp
#include <iostream>
#include <iomanip> //set precision
using namespace std;

int main()
{
    //initialization values
    double total(0.0); // sum user's input numbers
    long countEntries(0); // count user's inputs
    double average; // computed average
    double userInput; // individual value entered by user - do not need to declare as zero as a default value
    char ch; //get the userinput for yes or no

    cout << "Enter the first number: ";
    cin >> userInput;
    countEntries += 1;

    do
    {
        // prompt user
        cout << "Enter the next number (to be averages): ";
        cin >> userInput;

        countEntries += 1;
        if (userInput == 0.0)
        {
            break;
        }
    } while (userInput != numb);

    cout << endl;
    return 0;
}
```
else
{
    total = total + userInput;
}

cout << "Do you want to enter the number again? (y/n): "; // try again?
cin >> ch;
} while( ch != 'n' );
cout << endl;

// compute the average
average = total / countEntries;

// report the result
cout << "The average of the " << countEntries << " values you entered is " << fixed << setprecision(2) << average << " (to 2 places)." << endl;

return 0;
}

Here's the output of the program:

```
Enter the first number: 34
Enter the next number (to be averaged): 23
Do you want to enter the number again? (y/n): n

The average of the 2 values you entered is 11.50 (to 2 places).
```

What this program does is asking for a user the first number, then it jumps into the do while loop to ask for the second number or next numbers. The user can enter ask many numbers the would like till they hit the character 'n' on the keyboard to exit the loop. Then, the program compute for
the average of the numbers the user entered and output the average. The countEntries increment every time there is a new number entered.

Here's another do-while loop. This time we will build a random guess number game. We want the user to guess a random number of the computer generated. Let's look at the code below:

```c++
('/')[-* Program: random guess number game using do while loop */
* Author:
* Date Written:
* Version:
* Change history:
*-------------------------------------------------------*/
#include <iostream>
#include <ctime>  //seed time
#include <cstdlib>  //rand function
#include <cctype>
using namespace std;

int main ()
{
    int compNum;  // computer random number
    srand((unsigned)time(NULL));  // use time to make sure we don't get a same random number again
    compNum = (rand() % 100) + 1;  //number from 1 to 100 and avoid zero

    int userInput;  // user takes a guess of a number
    cout << "Guess my number between 1 and 100: ";
    cin >> userInput;

    int tries(0);  // user number it takes to guess the computer random number
    int olduserInput(0);  //user previous score
    char YorN(0);  //user want to play the game or not

    while (toupper(YorN) != 'N')
    {
        do
        {
            if (((userInput < 1) || (userInput > 100))
            {
                cout << "Please enter only number between 1 to 100\n";
            }
            else if (userInput < compNum)
            {
                cout << "Your number is too small. Try again." << endl;
                tries += 1;
            }
        } while (toupper(YorN) != 'N');
```
```cpp
else if (userInput > compNum)
{
    cout << "Your number is too big. Try again." << endl;
    tries += 1;
}

    cout << "Guess my number between 1 and 100: ";
    cin >> userInput;
} while (userInput != compNum);

olduserInput = tries;

    cout << "Good job, you it right in " << tries << " tries." << endl;
    cout << "Your previous tries was " << olduserInput << endl;
    cout << " Do you want to play the game again? Y or N: ";
    cin >> YorN;
}
return 0;
}
```

Here's the output of the program:

```
Guess my number between 1 and 100: 45
Your number is too big. Try again.
Guess my number between 1 and 100: 23
Your number is too big. Try again.
Guess my number between 1 and 100: 10
Your number is too big. Try again.
Guess my number between 1 and 100: 5
Your number is too small. Try again.
Guess my number between 1 and 100: 7
Your number is too small. Try again.
Guess my number between 1 and 100: 8
Your number is too small. Try again.
Guess my number between 1 and 100: 9
Good job, you it right in 6 tries.
Your previous tries was 6
Do you want to play the game again? Y or N: n
Press any key to continue...
```

There is a bug in this program, and I will let you try to debug it. If you're using the MS Visual C++ 6, hit F10 to run the debugger. You should see the debugger similar below:
Right click on the border not the white page area at line, int main(), and choose Insert Breakpoint. Scroll down to the line, return 0(), and insert another breakpoint. You should see a red icon appear beside it. This is the breakpoint. It means that the compiler should start to run the program in between these two breakpoints. Click on the Debug menu, and you will see the list of options for debugging. Step into means you want to go into the function and see what happens.
Step over means you don't want to go into this function on this line and you want to move on to
the next line. So go ahead and find the bug! We will go into more debugging later lessons. But
this is just an overview for now.

**The Switch Statement**

The switch statement is good for when you want to do a menu or you have large choices for the
user or program to make and do.

Here's the syntax for it:

```
switch(expression)
{
    case label1:
        statement;  //or more statements
        break;
    case label2:
        statement;  //or more statements
        break;
    default:
        statement;  //or more statements
}
```

Here's an example:

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

int main()
{
    int selections;
    cout << "Please enter a number for selection (1,2, or 3): ";
    cin >> selections;
    switch(selections)
    {
    } // end of switch
```
The program asks the user to enter a number between 1 to 3. Then the program runs the switch statement to see if the user picks either those numbers. If there is a matching type number, then it will execute that part of the code. In our case, example, user enter 2, the case 2 of the switch statement will execute and displays "you selected second selection" on the screen.

Here's a brief example of switch statement using single character:

```
switch(direction)
{
    case 'n':
        //do something here
        break;
    default:
        //the default statement here
}
```

**Logical Operators**

In C++, logical operators use to compare two expressions or evaluate two expressions. There are three of them:

```
&& means AND

|| means OR

! means NOT
```
If you recall to our first guessing game number above, the line, if ((userInput < 1) || (userInput > 100)), compares two expressions userInput < 1 vs userInput > 100. If either one of this expression is true then the if statement body will execute. If we change it to, if ((userInput < 1) && (userInput > 100)), then we want both expression to be true inorder for the body statement to be executed. The NOT operator works a little different. You have to have the ! infront of the expression like

if (!(userInput < 1) || !(userInput > 100))

It means if not userInput <1 and not userInput > 100 then go ahead and execute the body statement.

**Precedence and Other Control Statements**

The following is the table of which type of operator has the highest precedence in C++. The operator with the highest precedence will execute and get to evaluate first before the other precedence.

<table>
<thead>
<tr>
<th>Operator type</th>
<th>Operators</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unary</td>
<td>!, ++, --, +, -</td>
<td>Highest</td>
</tr>
<tr>
<td>Arithmetic</td>
<td>*, /, %, +, -</td>
<td></td>
</tr>
<tr>
<td>Relational</td>
<td>&lt;, &gt;, &gt;=, &lt;=, ==, !=</td>
<td></td>
</tr>
<tr>
<td>Logical</td>
<td>&amp;&amp;,</td>
<td></td>
</tr>
<tr>
<td>Conditional</td>
<td>?:</td>
<td></td>
</tr>
<tr>
<td>Assignment</td>
<td>=, +=, -=, *=, /=, %=</td>
<td>Lowest</td>
</tr>
</tbody>
</table>

For other control statements,

There is a break and the continue statement.

You can use break in the loop or decision statements if you want a certain part of the loop to stop when the condition is met.

Example:

```c++
for (int i = 0; i < 10; i = i * i + j) {
    if (x == 1) {
        cout << "x is 1 and I want the loop to stop";
        break;
    }
}
```

The for loop will break and stop looping if the if (x == 1) is true.

The continue statement takes you back to the beginning of the loop. When you uses it, be sure to include the semicolon at the end.
Structures and Functions

Overview:
- Structures
- Functions
- Practice Assignments

Structures

If you're designing a game and you want to store some specific information about a character in the game such as character's health, ammo, number of coin collects etc., you can do it with structures. C++ Structures allows us to group different data types into a single structure that relates to a specific type such as a character. A structure is more like a user defined type where you define a type and creates data for that type. A structure usage is similar to class objects usage which we will discuss it in later lessons.

Here's the syntax of the structure

```cpp
struct structName
{
    data types; //data types can be float, int, double etc...
};
//note there is a semicolon at the end - don't forget!
```

Example of a structure:

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

//-------------------Structure define-------------------
struct Player
{
```
```cpp
char Name[6]; //create a Name char that can hold up to 6 characters
int Age; //Player age
double score; //player score
}

//-------------------End of Structure define---------------------------------

int main()
{
    Player PlayerOne =
    {
        "Smith",
        15,
        91.25
    }; //create structure PlayerOne type of Player and assign values to data types of Player

    cout << "Lets call each data type of Player " << endl;
    cout << "PlayerOne Name is " << PlayerOne.Name << endl;
    cout << "PlayerOne Age is " << PlayerOne.Age << endl;
    cout << "PlayerOne Score is " << PlayerOne.score << endl;

    return 0;
}
```

In the above structure sample 1, we define Player as a structure type. In the Player structure, we have three different data types. First one, we have char Name[6] as a Name character. You will also notice there are two brackets, [ and ] with a number between it. This is called an 'array'. An array is a list of same data type grouping together. We'll talk and discuss about Array in later lessons. For now, just know that char Name[6] means a Name character type that can hold or store up to 6 characters. Next, we have a second data type which is Age integer, and the third one is score as a double type. Now, we have our data types declare. We can then use it in our main section and assign value to it. In our main, we declare a new type, known as identifier, PlayerOne for the Player structure. Then we use the structure syntax to assign values to it. In the example above, we assign Smith to char Name[6] type, 15 to Age integer type, and 91.25 to double type. You can also create another identifier for Player structure. Here's an another example of the new identifier:

```cpp
Player EnemyBoss =
{
    "Boss",
    36,
    99.99
};
```

As you can see the above example, we just have created the new identifier, EnemyBoss, for the Player structure and assigned values to the data types.
Next, we use the dot operator to access each data types, known as membership, of Player structure. The syntax for access a data or membership of a structure is:

```
identifier.datatype
```

In our structure sample 1, we access the Name data type membership of Player structure by using, Player.Name

If you recall in our definition for structure sample 1, we have char Name[6] so we use the data type of Name char to access the name. This is done the same thing for Age data, PlayerOne.Age and score data, PlayerOne.score. You can add two data together by using arithmetic signs like

```
PlayerOne.Age + PlayerOne.score;
```

Here's an output for the structure sample 1:

```
Let's call each data type of Player
PlayerOne Name is Smith
PlayerOne Age is 15
PlayerOne Score is 91.25
Press any key to continue
```

Let's look at another structure sample:

```cpp
/***********************************************************/
* Program: structure sample 2
* 
* Author:
* Date Written:
* Version:
* Change history:
```
In the structure sample 2, we create a new structure and define it as 'weapons'. Our weapons structure has two data types of int, ammo and rounds. In our main(), we assign values to our data, and then we access and call the data membership with the dot operator, Kfifteen.ammo and Kfifteen.rounds

Here is an output of the program:
As you know in previous lesson, you can nest loop within loop. Yes, you can nest struct within struct. Let's look at the example below:

```cpp
#include <iostream>

using namespace std;

//--------------------Structure Define for Player-----------------------------
struct Player {
    char Name[6]; //create a Name char that can hold up to 6 characters
    int Age; //Player age
    double score; //player score
};

//--------------------Structure Define for Weapons----------------------------
struct Weapons {
```
Player ammo; //Player ammo declaration using the Player struct when nesting struct within struct
Player rounds; //Player health declaration using the Player struct when nesting struct within struct

int main()
{
    Player PlayerOneA = 
    {
        "Smith",
        15,
        91.25
    }; //create structure PlayerOne type of Player and assign values to data types of Player

    //create PlayerOne indentifier for Player structure
    Weapons PlayerOne;

    //assigns values to both Player and Weapons structures using identifier
    PlayerOne.ammo.Age = 15; //assigns age and ammo with a value 15
    PlayerOne.rounds.score = 99; //assigns score and health same value of 99

    cout << "Lets call each data type of Player " << endl;
    cout << "PlayerOne Name is " << PlayerOneA.Name << endl;
    cout << "PlayerOne Age is " << PlayerOneA.Age << endl;
    cout << "PlayerOne Score is " << PlayerOneA.score << endl;
    cout << "PlayerOne Ammo is " << PlayerOne.ammo.Age << endl;
    cout << "PlayerOne Rounds is " << PlayerOne.rounds.score << endl;
    
    return 0;
}

Lets analyze the codes:

When using struct nested within struct, use two dot operators. As you can see our struct Player stays the same, we have a new struct named 'weapons'. Our weapons struct consists of two new new data type, ammo and health. We declare them as Player data type because we want Player and Weapons to be connect. In terms of Player has weapons or Player uses weapons. Therefore, inorder to reference weapons as a player data type we use the Player structure name, Player: Player ammo, Player rounds. In our main(), we assign Player data type with name and default values, 15 and 91.25 for age and score. Then, we have a new indentifier

Weapons PlayerOne;

After we've defined our new identifier, we could then use it. The next line, we have

PlayerOne.ammo.Age = 15;
it means PlayerOne is an identifier for Weapons structure, and go ahead and take Age data membership of inner Player structure and applies it to ammo data membership in the Weapons structure and assign it to the value of 15.

The rest shows you how to output the PlayerOne Weapons structure.

Here is an output of the structure within structure:

```
A quick look at Array of Structures, Unions, and Enumerations:

A. Array of Structures

Let's look at this sample:

```c
Player PlayerOneA =
{  
    "Smith",
    15,
    91.25
}; //create structure PlayerOne type of Player and assign values to data types of Player
```

You can use array of structures to assign two different data types. Again, we will discuss about array later in lesson. Here's an example:

```c
Player PlayerOneA[2] =
{  
    {"Smith", 15, 91.25}, //first structure in array
```
As you can tell without knowing how exactly array works. We have declared PlayerOneA with an array value of 2 so that we can create two different data fields or lines. If you want to create 3 data fields or lines, just change the value 2 to 3 and create another line for it in the total of 3.

You can access by number 0 to 1 to 2.

Example of accessing a variable with one array in structure:

PlayerOneA[1].name;

PlayerOneA[1].Age;

B. Unions

Unions is similar to structure that can hold data with different types (int, long, double, etc..) but it can only hold one data at a time.

The syntax for Unions is:

```c
union identifier {
    data types;
};
```

Example:

```c
union Player {
    char Name[6];
    int Age;
    double score;
};

int main() {
    Player PlayerOne;

    PlayerOne.Name = "Smith";
    cout << "Name is declared as " << PlayerOne.Name << ". Now it\'s Age to declare " << endl;

    PlayerOne.Age = 15;
    cout << "do something here for age, for now just show age: " << PlayerOne.Age << endl;

    return 0;
}
```
The above sample has a Player as a union type. Then in our main(), we declare and assign a value to one data type or variable of union at one a time. We use it that variable, then we assign the next one.

C. Enumerations

Enumeration is a set of symbolic constants where you can prefer to it in a set of numbers. Enumeration starts to count the first value as 0 and so on till the last value. These values can't be change. You can then use the value as a constant.

Here's an example of enumerations:

```cpp
enum primarycolor {blue, red, yellow};
```

This means that primarycolor is a type of enum and has the members of blue, red, yellow - which counts starting from blue as 0, red is 1, and yellow is 2 by default.

To prefer to it:

```cpp
primarycolor Artcolor;
Artcolor(1); //will display red
```

Another example:

```cpp
enum primarycolor {blue, red = 20, yellow};
```

In the line above, blue is 0, red is 20, and yellow is 21 because any unassigned value after the assigned value in the enum is larger by one than their predecessor.

Here's another example in partial incomplete program:

```cpp
enum GameLevels
{
    Easiest = 0;
    Easy = 1;
    Medium = 2;
    Hard = 3;
    Dety = 4;
};

switch (GameLevels)
{
    case Easiest:
        Increased_Ammo = 999;
```
So in the example above, we just specify the GameLevels from easiest to dety with a set of constants. Then we use the switch statement to prefer to it.

### Functions

Function is a small program or block of code to do specific thing within your actual program. You use function to organize your programs and to reduce the headache down the road.

The logic to declare the function as follow:

function prototype;

main() program {

functioncall; //in main

}

function definition {

//this is where you write your function code

}

Let's take our first function sample:

```c
/***********************
* Program: Function Sample 1
*
* Author:
* Date Written:
* Version:
* Change history: added the else if and else statement
*
***************************/
```
```cpp
#include <iostream>

using namespace std;

//---------Function Prototype--------------
void showMessage();

int main()
{
    showMessage(); //call the function showMessage we define
    return 0;
}

//---------Function Definition---------------
void showMessage()
{
    cout << "Hello, this my showMessage function" << endl;
}
```

In our Function Sample 1, we have a function a prototype,

void showMessage();

This means that we declare this is our new function as a void function. Remember void function
doesn't have a return type or it doesn't return anything. In C++, the compiler first run through this
line (void showMessage). It knows that there is a function showMessage and it can look for it in
the program whenever it encounters. Let's take a look at the bottom of our main(), we have a
function definition. This is where we can define function and what it does. In our Function
Sample 1, we define it to show the message "Hello, this my showMessage function". Get into
our main(), we can call the function by using its prototype identifier, showMessage() Hence we
have showMessage; This is the simplest function.

Remember the order to define the prototype and our function definition- it's best (for small
program) to define a prototype or declaration of a function before main(), then define the
function at the end of the main. For bigger program, you might want to add the function
prototypes in a header and function definition in another cpp files. It's a good programming
practice. In our later lessons, we get to split our bigger files up in multiple files such as headers.

Let's look at another sample of function.
* Change history:
* **************************************************/
#include <iostream>

using namespace std;

void swap(int&, int&); //function declaration or known as function prototype

int main()
{
    int firstNumb(0); //initialize firstNumb to zero as default
    int secondNumb(0); //initialize secondNumb to zero as default

    cout << "Please enter first number: ";
    cin >> firstNumb;
    cout << "Please enter second number: ";
    cin >> secondNumb;
    cout << "The numbers you entered are: " << firstNumb << " " << secondNumb << endl;
    cout << "Now, we're swapping the two numbers ....." << endl;
    swap(firstNumb, secondNumb); //swap function call
    cout << "After swap, the numbers are: " << firstNumb << " " << secondNumb << endl;

    return 0;
}

//function definition is below
void swap(int& s_numbOne, int& s_numbTwo)
{
    int temp;
    temp = s_numbOne;
    s_numbOne = s_numbTwo;
    s_numbTwo = temp;
}

Let's look at for for sample 2.

In our function prototype, we have void swap(int&, int&);

We declare a new function as a void type with two arguments, int& and int&. The ampersand, &,
means that this is a reference variable. The reference variable means this is an alias or just
another name for the variable it's referring to. The reference is also responsible for passing
arguments to functions. Look at our function definition:

```cpp
void swap(int& s_numbOne, int& s_numbTwo)
{
    int temp;
    temp = s_numbOne;
    s_numbOne = s_numbTwo;
    s_numbTwo = temp;
}
```
We define our swap function by swapping the s_numbOne to s_numbTwo, and s_numbTwo to
s_numbOne. Inorder for the swap function to work properly, we need a third variable, temp.
Also, look at the arguments for the swap function (int& s_numbOne, int& s_numbTwo). Here
we declare two integers variables as reference. In our main(), we call it with swap(firstNumb,
secondNumb);

When the compiler calls that function, the firstNumb variable is actually referring to the
s_numbOne in the swap function; and the secondNumb variable is also an alias to s_numbTwo
in the swap function.

Here is an output for that sample:

```
#include <iostream>
#include <iomanip> // setprecision
using namespace std;

float cal_CAsalestax(float); //function prototype for calculate california
sales tax

int main()
{
    float totalAmt(0); //initialize totalAmt to zero as default
    cout << "What's the total of the sales?: ";
    cin >> totalAmt;
    cout << "The total including tax is "
        << fixed << setprecision(2) //setprecision allow us to print out 2
decimal only
        << cal_CAsalestax(totalAmt) << endl; //calculate the sales tax
    return 0;
}

//function definition
float cal_CAsalestax(float totalAmt_entered)
{
    float total;
    total = totalAmt_entered * 1.0825;
    return total;
```
We start out by writing the function prototype as a type float, `float cal_CAsalestax(float);`

Then in our function definition, we have initialize the totalAmt_entered float variable and use it to multiply with 1.0825 (that's the current california sale tax), and we assign the result to the total float variable and return it to the function.

You can also do this:

```c++
float cal_CAsalestax(float totalAmt_entered)
{
    return totalAmt_entered * 1.0825;
}
```

The above will work the same. There is something new in our main() which is the setprecision. We use the setprecision (included and defined in the iomanip file)

```c++
fixed << setprecision(2)
```

This line above will set the output numbers to two decimal numbers like 1.29

Here's an example of the use of iomanip for setprecision:

```c++
#include <iostream>
#include <iomanip>

using namespace std;

int main ()
{
    // defaults
    cout << "Defaults:" << endl;
    cout << "Integer: " << 12345 << endl;
    cout << "Real: " << 3.0 << endl;
    cout << "Real: " << 12345.678901 << endl;
    cout << "Real: " << .000000123 << endl;

    // showpoint: display decimal point and trailing zeros for real numbers
    // The opposite is "noshowpoint".
    cout << showpoint;
    cout << "showpoint:" << endl;
    cout << "Real: " << 3.0 << endl;
    cout << "Integer: " << 12345 << endl;
}
```
// fixed: use fixed-point notation, not scientific (E) notation for reals

// The opposite is "scientific"
cout << fixed;
cout << "fixed:" << endl;
cout << "Integer: " << 12345 << endl;
cout << "Real: " << 3.0 << endl;
cout << "Real: " << 12345.678901 << endl;
cout << "Real: " << .000000123 << endl;

// setprecision(p): display p fractional digits for real values

// The default is 6 places.
// "setprecision" requires #include<iomanip>
cout << setprecision(8);
cout << "setprecision(8):" << endl;
cout << "Integer: " << 12345 << endl;
cout << "Real: " << 3.0 << endl;
cout << "Real: " << 12345.678901 << endl;
cout << "Real: " << .000000123 << endl;

return 0;
}

Here's an output for sample 3:
Let's look at another sample of function. This time we will look at the Recursion.

Recursion means the function calling itself.

```cpp
#include <iostream>

using namespace std;

int square(int); //function prototype for square

int main()
{
    int userNumb; //user entered number
    cout << "Enter a number: ";
    cin >> userNumb;
    cout << "The factorial of " << userNumb << " is " << square(userNumb) << endl;
    return 0;
}

//function definition
int square(int n)
{
    if (n > 1) {
        return 2 * square(n-1); //recursive call
    } else {
        return 1;
    }
}
```

In our sample 4, we have a recursive function prototype as a square int. Then we define the function definition and the return in that function call within a function, return 2 * square(n-1); it's a recursive. When using a recursive, makes sure you provide a way to end the recursion or else it will crash your program.

Here's the output:
The last topic for the function is the Inline Function. When you run the program with the inline function in it, the compiler inserts the code of the inline function when it gets compiled instead of jumping to the end or top to search for code in the function and call it. Sometime if your function is too long or too big, the compiler will treat your inline function as a normal function.

Here is an example of inline function:

```c++
#include <iostream>
#include <stdlib.h> // for rand function
#include <ctime> // for seed time

using namespace std;

inline void myrand(void)
{
    int n;
    srand((unsigned)time(NULL));
    n = (rand () % 100) + 1; // gets a random number between 1 to 100 --- without +1 the compiler will also generate a zero
}
cout << n << endl;
}

int main()
{
    myrand();
    return 0;
}

The above sample uses the inline function. The compiler will automatically inserts the function code when it gets compiled. It won't go anywhere else to search for it. In our example, the compiler will insert the myrand function into the main and will run it when it encounters the call. This is useful for small program because it will save us some memory usage. But for the bigger program we might have to use a normal function.

Here's the output of the program:
Assignments

Write a four function calculator program that lets the user to choose to perform either to subtract, multiply, addition, and division. The logic is this:

- prompt the user to choose the operation
- if the user choose one of the operations, bring up the operation and ask for two numbers to either subtract, multiply, divide, or adding.
- after the operation is done, ask the user whether to continue and choose another operation or quit the program

Write a text adventure game that do the following:
- display the player position at starting position like: "Your current location is: (1,1)"
- then ask the user to either move North, West, East, or South. If the user choose North, then increment the y. If the user choose South, decrement the y. If the user choose west, decrement the x. If the user choose east, increment the x.
- specify a location X==10,y==10. If the user reaches this location, display "You have entered the cave"
- give the user the option to exit the program by typing 'q' instead of Ctr-C

Write a program that encourages the user to enter two fractions, and then displays their sum in fractional form. Their sum can be calculated by \( \frac{a}{b} + \frac{c}{d} = \frac{a*d + b*c}{b*d} \). Optional: Express the result in LCD form.

Use for loops to make a program that display a pyramid of X on the screen look like this:

```
*********
*******
******
*****
***
*
```

Write a program that ask the user for the speed and then prints out a message based on the interval the inputted speed lies. For example:

- if 0 < speed < 10 then print out "you're a turtle!"
- if 10 < speed < 20 then print out "you're moving so slow!"
- ...
- ...
- if 80 < speed < 90 then print out "you're going too fast!"
- etc.

Be creative and come up with your own messages ;)

Write a program that displays your welcome message and then asks for a user name and display it to the screen.

Write a program that asks for a user number, then use that number and multiply with 10. Display the result to the screen.

Write a program that will take three input for two different points/vectors of x,y and z, original point on the plane. Have this program to calculate whether these two points on the same side or opposite of the plane, and display the result. Also, find the closest point on the plane.