CS 211 Notes

<div align="center">

Introduction to Programming

Jaeden Bardati

Last modified October 6, 2021

</div>

0 Course Overview
September 10, 2021

0.1 Objectives

The course is intended to teach how to develop a computer program to solve a problem. C++ is a tools that will be used to develop these skills and logical thinking. These skills will be transferable to other languages.

1 Computer Organization

1.1 Hardware
September 11, 2021

1.1.1 Components

Modern computers are built using the **Von Neumann machine**. There are three aspects:

- **Architecture**: **I/O** (User interaction) + **Memory** (Storage) + **CPU** (*CU*: Control Unit, *ALU*: Arithmetic and Logic Unit). These are all connected by a shared bus.

- **Stored Programs**: All programs and data are stored in memory (binary).

- **Sequential Execution**: Also called the **fetch-decode-execute** cycle. Instructions are **fetched** from memory, **decoded** by the CU and then **executed** by the ALU. If there is a result, it is stored back in memory.

1.2 Memory
September 11, 2021

The memory is organized in a **hierarchy**. At the bottom of the hierarchy is the Hard Drive (in TB). At the top is the CPU. Since the hard drive is slow, when some data from the hard drive is needed, it is first loaded into **RAM (Random Access Memory)** (in GB). The RAM is still too slow for the RAM, so the data is stored in **cache** (in KB or MB). Yet still, this is not fast enough for the CPU, so **registers** (in Bytes) in the CPU itself are used to store variables.
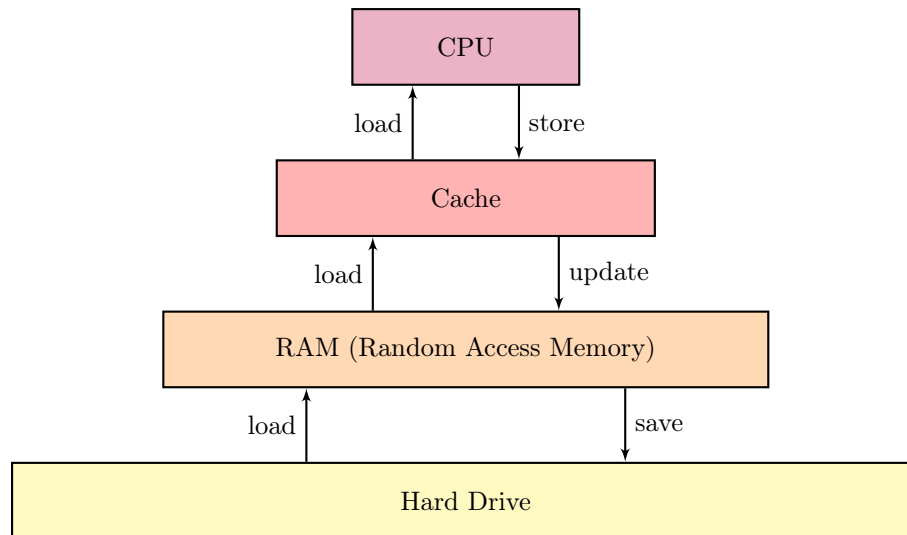
As you **go up** the hierarchy, the **speed increases**, but the **size decreases** and the **cost increases**.

1.2.1 RAM

Random access memory is organized in an array of Bytes ("words").

Words in RAM are addressed with a byte themselves (e.g. 01101101 is an address). These are typically written in hexadecimal (e.g. 6D).
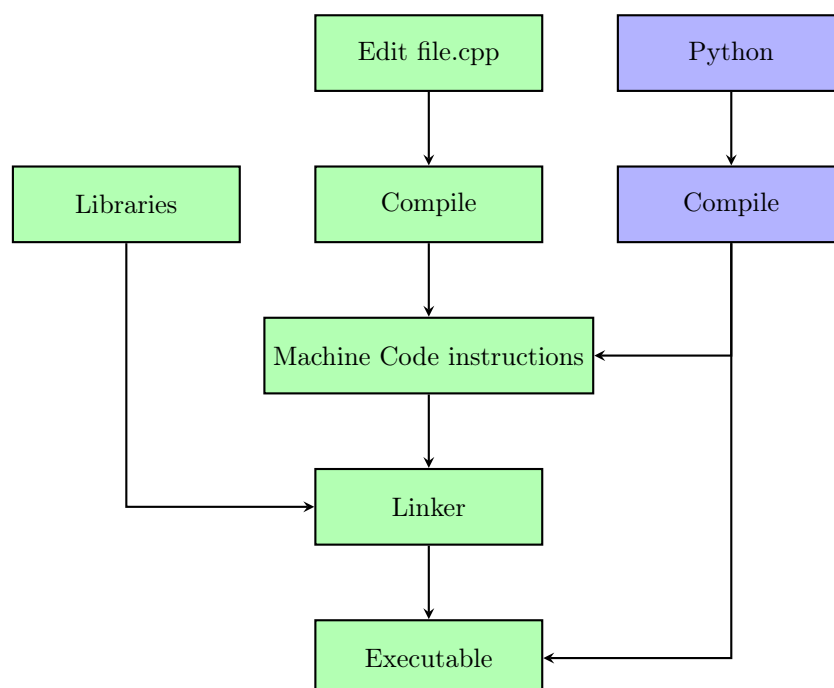
Words in RAM can be data or machine code instructions. Instructions contain a binary code for each operation (for example, addition). Instructions codes are dependent on the CPU.

2 Computer Organization

2.1 A Flow Chart: Program to Binary
September 12, 2021

This is a flow chart of what is done by the computer when compiling a C++ file. In blue is the Python equivalent.

Note that the bottom of the flow chart is the same for all programming languages, because in all languages, CPU-specific machine code is needed to execute code.

The process of catching errors is as according to the following flow chart.
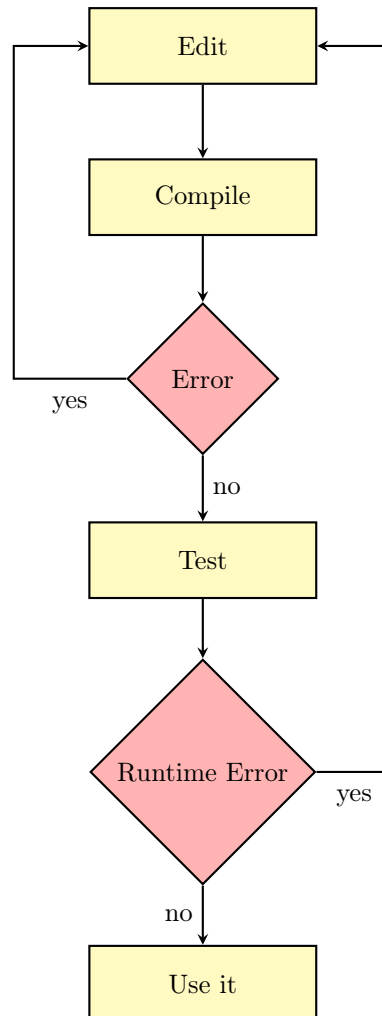
In this context, **errors** are caught by the compiler. This is opposed to **runtime errors**, which are not caught by the compiler. These can be something like division by zero or infinite loops.

2.2 First C++ Code
September 12, 2021

The following code is a hello world program in C++.

```cpp
// helloworld.cpp
#include <iostream>  // include statement allows the use of C++ libraries
#include <stdio.h>   // this library contains getchar()

using namespace std; // a standard environment (input from keyboard, output is the screen)

int main() {

   cout << "Hello world!" << endl;  // Prints "Hello world!" to the screen

   getchar();    // wait for user to type a character

   return 0;    // 0 means that the execution was successful
}
```

2.3 Data types
September 12, 2021

Variables are referred to as identifiers. Identifiers are memory locations accessed and modified.

Inside a main function (as above), the following code declares a variable of integer type in C++.

```cpp
int numYears;     // allocated space in memory to contain num of years
```

You can also initialize the variable with a value on declaration:

```cpp
int number = 5;   // declare and initialize (give a value too)
```

Some rules to follow when naming variables are:

- Names have meanings

- Must be case sensitive (e.g. numYears is not numyears)

- Consists of letters, numbers and underscores

- First character cannot be a number

Some types of variables are:

- Integers (e.g. -5, 0, +2) [int]

- Real numbers (floating point numbers or doubles, e.g. 2.453, -4.1987e7) [float or double]

- Booleans (e.g. true, false) [bool]

- Characters [char]

You can assign a value to a variable after declaring it:

```cpp
int width, height;
int area;

width = 5;
height = 3;

area = width * height;
```

Constants (denoted with the keyword "const") cannot be changed throughout the program. The convention is to use capital letters for constants.

```cpp
const double PI = 3.14159265;

int radius = 6;
double area = PI * radius * radius;
```

You would get an error if you were to try to reassign a constant.

```cpp
const double PI = 3.14159265;
PI = 3.14;  /// ERROR
```

2.4 Arithmetic operations

September 22, 2021

There are 5 basic arithmetic operations supported by C++

- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)
- Modulus or "Remainder" (%)

For example,

```cpp
int a = 5;
int b = 2;

cout << a + b / 2 << endl;    /// 6
cout << (a + b) / 2 << endl;  /// 3
cout << a / 2 + 3 << endl;    /// 5
cout << a / b / 3 << endl;    /// 0
```

This is done with integer operations since both arguments of each operation are integers (round down if the result of a division is a decimal). If one argument is a double/float, the result is a double/float. For example,

```cpp
int a = 5;
int b = 2;

cout << a + b / 2 << endl;    /// 6
cout << (a + b) / 2 << endl;  /// 3.5
cout << a / 2 + 3 << endl;    /// 5
cout << a / b / 3 << endl;    /// 0.83333333
```

2.4.1 Assigning and operations

When you are assigning a variable and performing an operation to it at the same time you can do the following:

```cpp
int x = 25;

x += 1    // equivalent to x = x + 1 (or x++)
x -= 2;   // equivalent to x = x - 2
x *= 3;   // equivalent to x = x * 3
x /= 4;   // equivalent to x = x / 4
x %= 5;   // equivalent to x = x / 5
```

2.4.2 Division when initializing or assigning a variable

Naively, to initialize a variable to the result of a division, you could try

```cpp
int div = 7/3;  /// 2
```

However, this results in 2, not the desired result of 2.3333. You could also try changing the variable type

```cpp
double div = 7/3;  /// 2
```

This does not work either, since the integer division happens before the result is assigned to the variable div. Instead, you have to force the compiler to recognize an argument as a float.

```
double div = 7.0/3;   /// 2.333333
```

If instead you had decided to make the variable type of div and integer and run it

```
int div = 7.0/3;   /// 2
```

It will perform the double operation, but save it in var as an integer.

A way of doing the float operation with two integers is by **casting** types. This can be done as follows.

```
double div = (double) 7/3;   /// 2.333333
double div2 = 7/(double)3;   /// 2.333333

int c = 7
double div3 = (double) c/3;   /// 2.333333
// Note, this does not change the type of c.
```

The remainder (%) operation returns the remainder of the division between two integers:

```
cout << 7 % 3 << endl;   /// 1
cout << 10 % 6 << endl;   /// 4
cout << 13 % 4 << endl;   /// 1
```

2.4.3   Type conversion

We can also convert types from doubles to integers by truncation or rounding:

```
double price = 2.55;

int sum = price;   /// truncate
int sum2 = price + 0.5  /// round to the nearest int
```

2.5   I/Os
September 22, 2021

To do this, we will use the iostream library and standard namespace.

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

2.5.1   Reading inputs

You can get input from the keyboard using cin. For example,

```
int number;

cout << "Please enter a value between 0 and 10." << endl;

cin >> number;   // reads integer

cout << "The entered number is " << number;
```

You can also get multiple variables with the same cin.

```cpp
int length, width;

cout << "Please enter the length and width of the rectangle: ";

cin >> length >> width;

int area = length * width;
cout << "The area is " << area
```

2.5.2 Formatting outputs

When outputting the result of a float, we inevitably run into an issue:

```cpp
double result = 123456789.1284567;  /// 1.23457e+008

cout << result << endl;
```

This returns 1.23457e+008. To see all the digits, we must first pass result to fixed before cout.

```cpp
double result = 123456789.1284567;

cout << fixed << result << endl;  /// 123456789.128457
```

To properly format the output of doubles, we must include the iomanip library and use the setprecision(n) function.

```cpp
#include <iomanip>

double result = 123456789.1284567;

cout << fixed << setprecision(2) << result << endl;  /// 123456789.13
cout << result << endl;                    /// 123456789.13
```

This rounds the double to the nearest nth decimal. Note that result will now format that way if outputted without the fixed and setprecision call. However, this does not actually change the value of the result variable.

```cpp
double result2 = result + 1;

cout << setprecision(3) << result2 << endl;   /// 123456790.128
```

Finally, to set a limit to the number of characters that are displayed to an integer n, we can use the setw(n) function from the iomanip library like this

```cpp
cout << fixed << setprecision(2) << setw(15) << left << result << "End of line" << endl;
```

The result is "123456789.13   End of line". Note the exactly 15 characters until "End of line".

2.6 Strings
September 22, 2021

To use strings, you have to include the string library:

```cpp
#include <string>
```

2.6.1 Declaration

You can declare, initialize, assign, and output strings as such

```cpp
string lname;
string fname = "Jaeden";
lname = "Bardati";

cout << fname << " " << lname;   /// Jaeden Bardati
```

2.6.2 Concatenation

To combine strings together, we must use **concatenation**.

```cpp
string name = fname + " " + lname;   /// Jaeden Bardati
```

This method does not work for constants

```cpp
string greeting = "Hello" + " " + " World";   /// ERROR
```

An error occurs because we are trying to add the string " " to the constant string "Hello". A way to get around this is by adding a dummy variable before the concatenation.

```cpp
string empt;     // empt = ""
string greeting = empt + "Hello" + " " + " World";   /// Hello World
```

2.6.3 Inputting strings

To read in a string, we can naively try

```cpp
string myname;

cin >> myname;   /// Jaeden Bardati

cout << myname;     /// Jaeden
```

Only one word at a time is read in because cin treats things separated by spaces as distinct.

```cpp
string myname;
string myfname, mylname;

cin >> myfname >> mylname;       /// Jaeden Bardati

myname = myfname + " " + mylname;

cout << myname << endl;          /// Jaeden Bardati
```

To get a full sentence (string) with cin, use the function getline(cin, sentence); This will get the whole line and put it in the variable sentence.

2.6.4 String functions

A useful quantity to know for a given string could be its length. You can get it as follows

```cpp
int n = myname.length();

cout << "Your name has " << n-1 << " characters" << endl;
```

Here the number of characters is $n - 1$ since the space is not included.

You can also get substrings using substr(i, n) to get the n characters proceeding the character at location i. If only i in entered, it will extract all characters preceding the character at location i.

```
string sub = myname.substr(0, 6);  /// Jaeden
string sub2 = myname.substr(7);    /// Bardati
```

2.6.5   Characters

Characters are strings of length 1, however, they can be declared explicitly with the char keyword and are expressed using single quotes.
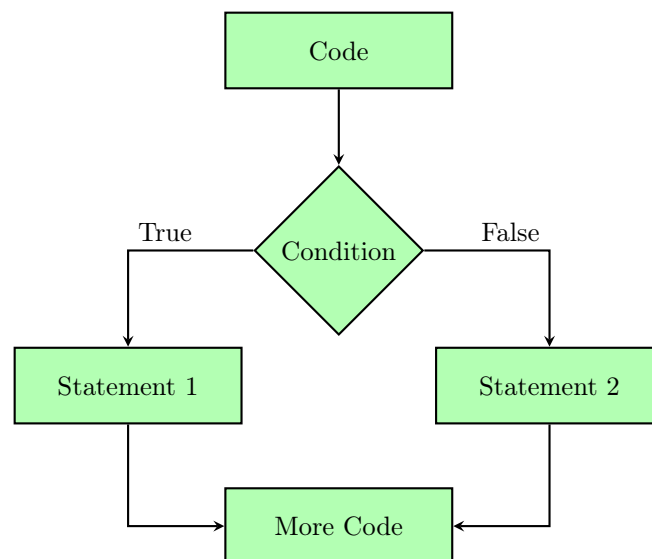
```
char c = 'Y';
```

2.7   The If Statement
September 23, 2021

We can also use conditional statements.

2.7.1   Flow Chart

2.7.2   Syntax

The if statement has the syntax

```
if (condition)
    statement1;
else
    statement2;
```

For multiple statements, we can use parentheses

```
if (condition) {
    statement;
    statement;
    ...
}
```

```
else {
    statement;
    statement;
    ...
}
```

2.7.3 The condition

A condition is a **boolean expression**. A boolean type can be **true** or **false**.

Boolean expressions are written using **relational operators**:

$<$	Less than
$>$	Greater than
$<=$	Less than or equal
$>=$	Greater than or equal
$==$	Equal to
$!=$	Not equal to

You can also chain relational operators with **logic operators**:

$\|\|$	OR
&&	AND
!	NOT

For example,

```
if (num < 20 || num % 3 == 1) // if num is < 20 or if the remainder of num / 3 is 1
    num = num * 2
else
    num = num / 2
```

2.7.4 Floating point precision error

Let us show an example of a floating point precision error. Note that here we are including the cmath library.

```
double root, number;

root = sqrt(3);
number = pow(root, 2);

cout << setprecision(18) << "Number is : " << number << endl; // 2.999999999999999956, not 3

if (number == 3)
    cout << "sqrt(3) * sqrt(3) = 3" << endl;
else
    cout << "What just happened?" << endl;
```

When running this in a main function, we obtain the "What just happened?" result. What just happened indeed? The issue is that floating point numbers are only precise to a certain number of digits. To account for this, we must set our own precision as follows:

```
double precision = 1E-14;

if (abs(number - 3) < precision)
    cout << "sqrt(3) * sqrt(3) = 3" << endl;
else
    cout << "What just happened?" << endl;
```

This returns the desired result of "sqrt(3) * sqrt(3) = 3".

2.7.5   String comparison (alphabetical order)

We can also use comparison operators for strings. For example,

```
string name1, name2;

cin >> name1 >> name 2;

if (name1 < name2)
   cout << name1 << endl << name2;
else
   cout << name2 << endl << name1;
```

This will return two entered strings in alphabetical order.

2.7.6   Nested ifs

Sometimes there are more than one options. To do this we can use nested ifs. You check conditions within conditions for multiple branchings.
For example, we can assign a letter grade for a given percentage grade.

$$
\begin{array}{ll}
A & 80 <= grade <= 100 \\
B & 65 <= grade < 80 \\
C & 50 <= grade < 65 \\
F & 0 <= grade < 50
\end{array}
$$

The implementation of this in code is

```
int grade;
char letter;

cout << "Please enter a numeric grade: ";
cin >> grade;

if (grade >= 50)
   if (grade >= 65)
      if (grade >= 80)
         letter = 'A';
      else
         letter = 'B';
   else
      letter = 'C';
else
   letter = 'F';

cout << "The letter grade is " << letter << endl;
```

2.8   The Switch Statement
September 30, 2021

The **switch statement** is an alternative to nested ifs. This statement can be used if it compares a boolean, integer or character against a constant of the same type. Each constant creates an **alternative** or **branch**.

The syntax is as follows:

```
int n;
cin >> n;

switch (n) {
    case 1: cout << "ONE"; break;
    case 2: cout << "TWO"; break;
    case 3: cout << "THREE"; break;
    case 4: cout << "FOUR"; break;
    case 5: cout << "FIVE"; break;
    default: cout << "Not a number between 1 and 5";
}
```

The general process of the switch statement is that it will check each "case" until it finds a match, then it will run what comes next. The "default" will run if no other case has run. Remember to put breaks after each case! No break is needed after the default case.

You can also stack cases as follows:
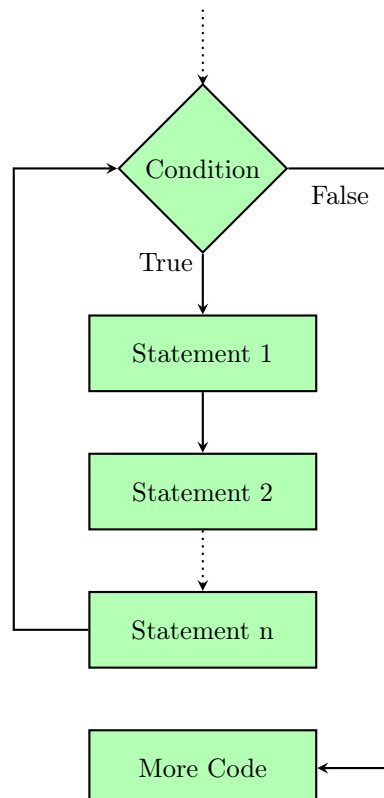
```
char c;
cin >> c;

switch (c) {
    case 'y': case 'Y': cout << "YES"; break;
    case 'n': case 'N': cout << "NO"; break;
    default: cout << "Not y or n";
}
```

2.9 The While Loop
September 30, 2021

The **while loop** can be used when a sequence of statements must be repeated multiple times so long as a condition is true. The following is a flow chart of the process.

12

The syntax for a while loop of *one statement* is

```
while (condition)
    statement;
```

Similarly, a while loop of *multiple statements* is made possible using parenthesis

```
while (condition) {
    statement;
    statement;
    ...
}
```

There are a few consideration we need to take into account:

- The condition must be initially true, other wise the loop will never execute

- If the condition is initially true, the loop executes at least once

- The loop executes until the condition becomes false

- The condition becomes false as a results of a statement in the loop

- If the condition never becomes false, the loop will be infinite

The following code is an example program using a while loop, to calculate how many years it takes for a bank account balance to double with a given rate:

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

int main() {
    // Calculates the number of years to double the balance of a bank account.
    const double RATE = 0.03;

    double bal;

    cout << "Please enter an initial balance: ";
    cin >> bal;

    double target = 2 * bal;
    int years = 0;

    while(bal < target){
        bal += (bal * RATE);
        years++;
    }

    cout << "You require " << years << " years to double your balance.";

    return 0;
}
```

Quick Aside: In order to get the last number of a digit, you can use mod 10. For example, to get the last number of 729, we find that $729 \% 10 = 9$. We can use integer division by 10 to make 729 become 72 and then get the last number of that in order to get the second number form the right (here, 2).

Another Quick Aside: If a variable is entered that is not of the right type in cin, it will fail. There is a function that checks if it fails: cin.fail(). It will return false if cin has not failed, and true if it has. After a cin fails, all the subsequent cins will fail. To allow for subsequent cins after failure, we can use cin.clear(). However, since the failed character is still in the buffer, you cin will fail again. Therefore, we need to also call cin.ignore(n, lastchar) after cin.clear() to ignore what is in the buffer that caused it to fail (up to n characters or up to lastchar). The following code demonstrates this.

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

int main(){

    int input;

    cout << "Enter an integer value: " << endl;

    cin >> input;
    cout << "You have entered: " << input << " Status of cin: " << cin.fail() << endl;

    cin.clear();
    cin.ignore(10000, '\n');

    cout << "The cin status is now : " << cin.fail() << endl;

    cin >> input;
    cout << "You have entered: " << input << " Status of cin: " << cin.fail() << endl;

    return 0;
}
```
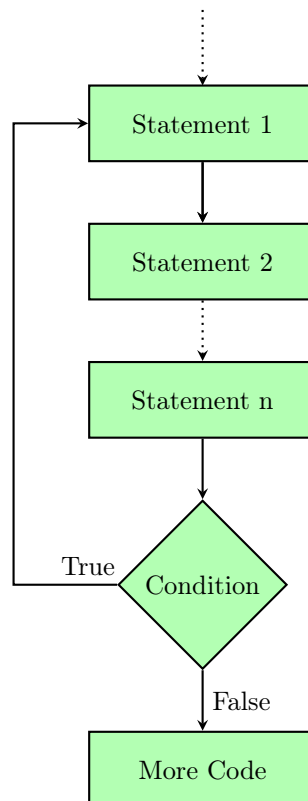
2.10   The Do-While Loop
October 6, 2021

The **do-while loop** can be used when a sequence of statements must be repeated multiple times so long as a condition is true. The following is a flow chart of the process.

This is different form the regular while loop because it always executes at least once. If the condition is initially false, there is one one execution. If the condition is initial true, the body of the loop is executed until the condition becomes false. The minimum number of iterations of a do-while loop is 1, whereas

the minimum number of iterations for a while-loop is 0.

The syntax in C++ for one statement is

```
do
    statement;
while(condition);
```

For many statements, you need curly braces as usual.

```
do {
    statement1;
    statement2;
    ...
}
while(condition);
```

An example application of a do-while loop is in input validation:

```
/// input validation using a do-while
do {
    cout << "Please enter a value for 0 < n < 100 ";
    cin >> n;
} while(n <= 0 || n >= 100);
```

2.11   The For Loop
October 6, 2021

The for loop is yet another iterative statement. For loops are based on the use of counters. It has the syntax:

```
for(statement1;statement2;statement3)
    statement;
```

Or, for multiple statement loops

```
for(statement1;statement2;statement3) {
...
statement;
...
}
```

The statement1 is for counter initialization.
The statement2 is for the loop condition.
The statement3 is to update the counter.

To demonstrate this we will compare it to the while loop.

```
int i = 1;                          for (int i=1; i<=n; i++)
while (i <= n) {                        sum = sum + i
    sum = sum + i;
    i = i + 1;
}
```

We can see that the for loop is much more compact. It is preferred to use a for loop when you are using counters.

2.11.1   Nested For Loops

Just as if statements can be nested, so can loops. Nested loops are usuful since sometimes it is necessary to loop over loops (or repeat loops a certain number of times).

For example, if we wanted to draw an n by n square of *s. We could use a nested loop:

```cpp
int n = 4;

for(int i = 1; i <= n; i++) {
   for (int j = 1; j <= n; j++)
      cout << "*";
   cout << endl;
}
```

You can even change the number of iterations in the inner loop via nested loops. For example, if we wanted the pattern

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

Then, we could use the following code.

```cpp
int n = 4;

for(int i = 1; i <= n; i++) {
   for (int j = 1; j <= i; j++)
   cout << "*";
   cout << endl;
}
```

2.12   Random Variables
October 6, 2021

Sometimes it is useful to have (pseudo)random numbers to use in a program. The library that we need to include is

```cpp
#include <cstdlib>
```

Then, to get two random numbers, we can do

```cpp
int r1, r2;

r1 = rand();
r2 = rand();

cout << r1 << endl << r2 << endl;
```

Running it again though, you can notice that it gives you the same numbers as before!

This is because the the number is not truly random, only pseudo-random (random-looking). They are really just values that are generated from a mathematical sequence.

This sequence has to start somewhere and that number is called the "seed". We can set the seed value using

```cpp
srand(765); // 765 is some arbituary number
```

If we want this to change with time, we need to input the computer time. To do this we will import the following library:

```
#include <ctime>
```

We can get the time in seconds since January 1st 1970 using

```
cout << time(0);
```

We can use this as the seed to make the starting random numbers change over time.

```
srand(time(0));
```

What is the range of the outputted randome numbers by rand()? It is between 0 and RAND_MAX. This is a constant defined in the cstdlib library. We can get this value ourselves:

```
cout << RAND_MAX << endl;
```

If we want doubles instead, we can do

```
int r = rand();
double x = r*1.0/RAND_MAX;
```

The variable x is a random variable between 0 and 1.

What if we want to generate random variables between and arbitrary a and b? To do this, we need to follow this procedure:

- Generate r between 0 and RAND_MAX (via rand())

- Generate x between 0 and 1 (via x = r*1.0/RAND_MAX))

- Generate y = a + (b-a)*x

So, if x = 0, y = a + (b-a)*0 = a. If x = 1, y = a + (b-a)*1 = a + b - a = b.

In code, this is

```
int r = rand();
double x = r*1.0/RAND_MAX;
double y = -1 + 2 * x;

cout << y << endl;
```