CLASS #1 NOTES
February 9, 2015
The ENS 117 Web Site

I use a web site to provide notes, classroom policy, supplemental information, and administrative information.

The web site can be found at:

www2.sunysuffolk.edu/lambiar

I update this site frequently, so check it often.
Parts of a Computer

The computer can be described very well with the simple diagram below. There are only three parts:

1. Memory – This part stores information in digital format.
2. Input / Output (I/O) – This part interfaces with people and devices outside of the computer.
3. Central Processing Unit (CPU) - This part can execute a set of instructions stored in memory, such as simple calculations and exchanging data with the other two parts.

Understanding how these three parts work will help us use the computer more effectively. We will examine each part separately.
Memory – Binary Numbers

“There are only 10 types of people in the world: those who understand binary, and those who don’t.“

Data is stored in memory as binary numbers. Unlike the decimal system, where each digit can have any of ten values (0 – 9), the binary digit (bit) can only be either 0 or 1.

Decimal numbers with more than one digit are defined by the powers of 10. So, the number 243\textsubscript{10} is interpreted as 2\cdot10^2 + 4\cdot10^1 + 3\cdot10^0 = 243\textsubscript{10}. In a similar fashion, a binary number with more than one bit is defined by the powers of 2. 1011\textsubscript{2} = 1\cdot2^3 + 0\cdot2^2 + 1\cdot2^1 + 1\cdot2^0 = 11\textsubscript{10}.

A three digit decimal number could represent 10^3 = 1000 objects (0 – 999). Similarly, an eight bit binary word can represent 2^8 = 256 objects (0 – 255). An eight bit binary word is called a byte, and is the basic unit of memory size.

Other number bases are commonly used, including octal (base 8, digits 0 - 7), and hexadecimal (base 16, digits 0 - 9 and A - F)

Memory – Size

Memory size can be expressed in a fashion similar to engineering notation, but instead of each size being different by a factor of $1000 \ (10^3)$ as it is in the decimal system, each size is different by a factor of $1024 \ (2^{10})$.

- 1 Kilobyte (KB) = 1024 bytes ~ 1 Thousand bytes
- 1 Megabyte (MB) = 1024 KB ~ 1 Million bytes
- 1 Gigabyte (GB) = 1024 MB ~ 1 Billion bytes
- 1 Terabyte (TB) = 1024 GB ~ 1 Trillion bytes
- 1 Petabyte (PB) = 1024 TB ~ 1 Billion Trillion bytes

A single byte can hold one character. The complete works of Shakespeare can be stored in about 5 MB. The first IBM PC hard drive had a capacity of 10 MB. A CD stores about 700 MB, a single layer DVD 4.7 GB, and a typical modern hard drive stores about 500 GB = 0.5 TB.
Memory – Addressing and Data Type

Each byte of information is stored at a particular address in memory. Just as you would need to know an address to send someone mail, you need an address to put data in memory (and to get it out, as well).

Each byte represents a number from 0 to 255. Every possible object that can be stored in memory (numbers, letters, pictures, music) is just a bunch of these bytes. What makes numbers different from letters and pictures and music is how the software interprets these numbers.

When we use data in our programs we will declare all variables. This will tell the computer how many bytes of memory to reserve for those variables, and how to interpret those bytes.
In our programming, the standard input is the keyboard and the standard output is the display screen. For historical reasons, these are known as console input and console output. Knowing this, it is easier to remember that our input command will be “cin >>” for console in, and the output command will be “cout << “.

Other I/O devices interface with standard data ports, such as USB and IEEE 1394. Input devices include mice, scanners, and cameras. Output devices include printers and speakers. Other types of ports are used for network connections (wired and wireless), Bluetooth, audio, and video.
The Central Processing Unit (CPU)

An executable program is stored as a list of instructions in memory. The CPU reads these instructions one at a time. Depending on what the instruction is, the CPU may do one of several things:

- Move data from one place (memory, or I/O) to another.
- Do a simple arithmetic operation on two numbers, like addition or subtraction.
- Do a relational operation on two numbers. Which number is bigger, or are they the same?
- Do a logical operation on individual bits of two number, like AND, OR, or NOT.

The only intelligence the computer has is that which the programmer or user puts into the program!
Programming Levels

The lowest level of programming gives you the most control of the CPU. But, it requires the highest level of knowledge of the logical architecture of the CPU and the hardware within the computer (memory and I/O). At the highest level of programming, you don’t need any knowledge of the CPU, nor most details of the computer hardware. For example, printer drivers written for any model will do the hard work for you.

I will describe some levels from lowest to highest.

- Microcoding or Reduced Instruction Set Computer (RISC) – The CPU does no arithmetic functions and only a reduced set of logical and data moving instructions. Only specialized CPUs are programmed like this.
Programming Levels - continued

- Machine Language – This is what the typical CPU understands. It is programmed with a series of binary numbers. All higher level programming languages will need to be translated to this level before the CPU can act on it.

- Assembly Language – There is a one-to-one corresponding instruction between assembly language to machine language. You still need the specialized knowledge of the hardware, but instead of number, you can write in acronyms. For example, MOV A, B, would be the command to move the data from register B to register A. Assembler software translates the acronyms to the binary numbers.

- Interpreted Language – This language is written at a higher level, such as A = C + 2.5. Each instruction is read in sequence, and as it is read, it is translated to machine language by interpreter software. BASIC is an interpreted language.
Compiled Language is also written at a high level, but the instructions (called source code) are entirely converted to machine code by compiler software. The result, called object code, is linked with software library object code by linker software. The linker produces executable code that can run by itself on computers without the compliers or linkers. This is not true of the Express Edition of the software used in class. It requires the Visual C++ software on the machine. To distribute software, you need the professional version.

Visual C++ is an Integrated Development Environment (IDE). That means in addition to the compiler, libraries, and linker, it also includes a text editor, debugger, and everything else you need to create programs.
The Visual C++ IDE

The steps to create, debug, and run a program in Visual C++ will be shown in class today. Those steps will remain on this web site. It is important that you use the IDE enough to be able to create a program on your first exam. All homework assignments must be done either at home with your own computer, or using a college computer (in the library, for example) outside of class time.

This software will not run on a Mac. If you have a Mac, there are two ways you can do your homework. If you have a Windows virtual machine using Parallel Desktop or VM Ware, you can install the software directly. If not, you can use Xcode, which is either included on the Mac, or can be downloaded from Apple for free. An example of using Xcode is on the ENS 117 web site. Even if you use Xcode to do your homework, you will need to fluent in Visual C++, as that is the software that will be used for the exams.
Problem Solving

It is important to understand that no matter how powerful a computer is, the ability to solve the problem comes from the programmer, not the computer.

Incredible things have been done with very simple computers. The first electronic computer, the Mark 1 Colossus ran in November 1943 as part of the Bletchley Park code breaking facility in England. This computer used 1,500 vacuum tubes. This was followed by the Mark 2, with 2,400 vacuum tubes.

The Mark 2, which went into operation June 1, 1944, provided critical information to help with the D-Day landing at Normandy on June 6, 1944.

To put this in perspective, the simplest operation for a single bit (not byte) requires at least one vacuum tube. While the Colossus was a breakthrough in computing speed, the intelligence came from the programmers.
The Plan – How to Make a Dog House

To solve a complex engineering problem, a systematic approach needs to be taken. To illustrate this, let’s look at two ways to build a dog house.

One way to do it is to run to a home improvement store, grab a lot of materials, come home and then figure out how you’re going to put this stuff all together. This has the advantage of your starting to build right away, but the project is in trouble almost right from the start. Soon, you’ll find pieces that don’t go together, missing materials, and tools you don’t have. Even after many additional trips to the home improvement store, the dog house still may not come out right.

It sounds very silly to build a dog house this way, but it perfectly describes the technique of first time programmers. The first step for them is to go to the computer and start typing.
A better way to build the dog house would be to plan before you build.

First, you need to know the right size and design for your dog. Then, you can make a sketch showing how you expect to construct it. From this sketch, you’ll be able to estimate the materials you’ll need and in what quantities. Since you have a plan to build the dog house from start to finish, you’ll also know what tools will be needed.

This means one trip to the home improvement store, and a dog house that turns out perfectly.
Pseudocode

There are several methods to solve a problem. The one we’ll use in class is called pseudocode. We’ll start this with the earliest and simplest problems. The fact is, you don’t really need very much planning to solve the easiest problems, but you do need to learn how to plan.

Let’s go back to the yard for another example. Suppose you’re just going to put up a clothes line between two posts. Not much planning there. You could just buy a couple of posts and a post hole digger, and you’d be done. But planning, even this simple task, would prepare you for more complex tasks like a dog house or a particle accelerator.

So how will this relate to writing a program?
We’re going to use an example to learn the Visual C++ IDE. If we were going to develop this from the start, the pseudocode might be as shown below:

- **Setup**
  - Include pre-processor files (input / output)
  - Print an opening message

- **Input**
  - Prompt and enter the radius and the height

- **Calculate**
  - Compute volume and surface area

- **Output**
  - Output input parameters, radius and height
  - Output results, volume and surface area
<table>
<thead>
<tr>
<th>Category</th>
<th>Type</th>
<th>Size</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integral</td>
<td>char</td>
<td>1 byte</td>
<td>Type <code>char</code> is an integral type that usually contains members of the execution character set — in Microsoft C++, this is ASCII.</td>
</tr>
<tr>
<td></td>
<td>bool</td>
<td>1 byte</td>
<td>Type <code>bool</code> is an integral type that can have one of the two values <code>true</code> or <code>false</code>. Its size is unspecified.</td>
</tr>
<tr>
<td></td>
<td>short</td>
<td>2 byte</td>
<td>Type <code>short int</code> (or simply <code>short</code>) is an integral type that is larger than or equal to the size of type <code>char</code>, and shorter than or equal to the size of type <code>int</code>. Objects of type <code>short</code> can be declared as <code>signed short</code> or <code>unsigned short</code>. <code>Signed short</code> is a synonym for <code>short</code>.</td>
</tr>
<tr>
<td></td>
<td>int</td>
<td>4 bytes</td>
<td>Type <code>int</code> is an integral type that is larger than or equal to the size of type <code>short int</code>, and shorter than or equal to the size of type <code>long</code>. Objects of type <code>int</code> can be declared as <code>signed int</code> or <code>unsigned int</code>. <code>Signed int</code> is a synonym for <code>int</code>.</td>
</tr>
<tr>
<td></td>
<td>long</td>
<td>4 bytes</td>
<td>Type <code>long</code> (or <code>long int</code>) is an integral type that is larger than or equal to the size of type <code>int</code>. Objects of type <code>long</code> can be declared as <code>signed long</code> or <code>unsigned long</code>. <code>Signed long</code> is a synonym for <code>long</code>.</td>
</tr>
<tr>
<td>Floating</td>
<td>float</td>
<td>4 bytes</td>
<td>Type <code>float</code> is the smallest floating type.</td>
</tr>
<tr>
<td></td>
<td>double</td>
<td>8 bytes</td>
<td>Type <code>double</code> is a floating type that is larger than or equal to type <code>float</code>, but shorter than or equal to the size of type <code>long double</code>.(^1)</td>
</tr>
</tbody>
</table>
Constants and Variables

Care must be taken when writing constants. Visual C++ will interpret the number 5 as an integer, 5.0 as a double, ‘5’ as an alphanumeric character, and “5” as a string of alphanumeric characters.

Variables must have a name. To be a valid variable name, it needs to follow these rules:

- It can only use the characters A – Z, a – z, numbers 0 – 9, or an underscore.
- The first character must be a character or an underscore.
- It can’t be a key word (see text Table 2-4, pg 40).
- Remember C++ is case sensitive. Max and max are two different variable names.

Longer names are more descriptive. That may mean a little more typing, but you’ll know what the names represent without thinking much.
Declaring Variables

All variables in C++ must be declared. Variables are declared by stating the data type followed by a variable name.

    double velocity;

More than one variable of the same type can be declared on the same line by separating with commas.

    int nickels, dimes, quarters;

Variables can be initialized at the same time they are declared.

    int Count = 0, min, max;
Arithmetic Operators

By itself, C++ has remarkably few arithmetic operators. To use trigonometric functions like `sin()` and `cos()`, or even exponentiation, we’ll need to include a library, `cmath`. For this week, we’ll just examine what is within C++ without libraries.

We have the basic four functions that operate on two numbers: add (+), subtract (-), multiply (*) and divide (/) as well as modulo (%), which is the remainder after division. There are some operators that use only one number: negation(-), pre-increment (++i), post-increment (i++), pre-decrement (--i), and post decrement (i--).

The assignment operator (=), is used to assign the value of the arithmetic expression on its right to the single variable on its left.
The Assignment Operator

The assignment operator looks like an equal sign, but it is not. It will evaluate the arithmetic expression on the right side of the operator, and put the resulting value into the variable on the left side of the operator. Here are some things to remember about the assignment operator:

- The left side of the operator must be only a variable. It cannot be a constant or an arithmetic expression.

- The same variable can be on both sides of the operator, as in \( x = x + 1 \); If \( x \) has the value five, the right side will be evaluated as six. Then, six will become the new value of \( x \).

- There are short hand notations for the assignment operator.

  \( x += 5 \); is the same as \( x = x + 5 \);
  \( x /= 5 \); is the same as \( x = x / 5 \);
Precedence of Operations

The arithmetic expression is evaluated from left to right. However, the computer compares the next operator to the one following it. If the following operator has higher precedence, then the following operation is performed first. In the following expression:

\[ c = 7 + 2 \times 5; \]

The variable \( c \) is assigned the value 17, and not 45, as multiplication has a higher precedence than addition. If you wanted this to be 45, you can use parenthesis.

\[ c = (7 + 2) \times 5; \]

The precedence of operations (including operators we didn’t discuss yet) can be found in the text in Appendix C, Table C-1 (pg 554).
We will write to the screen using the `cout` (console output) command. To start with, we’ll either output an arithmetic expression or a character string. A character string is a group of characters enclosed in double quotes (""). Each variable or string is preceded by the characters "<<". Using a backslash \" before a character sends command to the screen printer. \"\n\" in a character string causes a new line to print.

```cpp
x = 3;
y = 1;
cout << "The team has used " << y << " of their " << x << " time outs." ;
cout << "\nThey have " << x - y << "remaining.";
```

This will give us the screen display below:

The team has used 1 of their 3 time outs.
They have 2 remaining.
Input From the Keyboard

We will input from the keyboard using the `cin` (console input) command. To start with, we’ll input a value to a single variable. Arithmetic expressions are not permitted here. Always precede the `cin` command with a `cout` prompt, so the user is sure what information is needed.

```cpp
cout << “How many pills do you have? “;
cin >> NPills;
cout << “How many pills do you take each day? :
cin >> PillsPerDay;
cout << “\nYou have enough pills left for “ << NPills /PillsPerDay << “ days.”
```

This will result in the screen display below (assuming the given inputs):

How many pills do you have? 30
How many pills do you take each day? 2
You have enough pills left for 15 days.
The Structure of a C++ Program

//Program comments: program name, your name, date

Preprocessor directives: #include, using namespace std;

int main ( ) //The function header for main( )
{
    //This { marks the beginning of main( )

    All your statements go in here, generally starting with the variable declarations and ending with a return statement.

    return(0);
}
    //This } marks the end of main( )