Compiler

Machine Language is what the computer naturally executes and is a series of zeros and ones (binary). That level of language was the first ever created when computer were invented. It's great for the computer but more difficult for the human. So the humans created various ways to make their jobs easier to deal with the computer. Like so:

1. They first of all had to make a primitive operating system to allow running of programs
2. Then they made an Assembler that took more English like commands (like ADD X, 2) to convert directly, one-to-one, into machine language (like 100111000010)
3. After that someone came up with an idea to have a high-level language that was even more English like.
    a. One of the first high-level languages was FORTRAN (for FORmula TRANSlation)
    b. It came out about 1954 and was a jump into making programming much easier, especially for scientific applications.
    c. But it was lousy for business applications so the Department of Defense, of all organizations, sponsored a project to create a business-oriented language called COBOL (COmmon Business Oriented Language)
    d. It was quite English-like, some say too English-like (Add 2 to X). There is no such thing as a short COBOL program!
4. After that came quite a few languages
    a. ALGOL (ALGOrithm Language) was one of the many that came to light that was very elegant.
    b. Unfortunately, it didn't catch on but it influenced many languages like C and C++ (note: C++ came from C but has taken a life of its own).
    c. Bell Laboratories helped develop C AND C++. Both inventors worked at Bell Labs:
        i. Dennis Ritchie was the creator of C (used language of B to make it) in the early 1970s
        ii. Bjarne Stroustrup was the creator of C++ in the early 1980s
5. All of these languages needed a compiler to translate the high-level into machine language – never can entirely get away from that.

There are also "libraries" that contain commonly used and useful operations that programmers need. No need to reinvent the wheel (generally). In C and C++, these libraries are accessed using what is called "header files" (more on that later, for now just use them).

C++ and C (the predecessor to C++) compiler does the following steps:

1. Does the preprocessor that includes all header files and any other miscellaneous code/changes into one large program
2. Compiles the program
3. If the program has no errors, it will run the linker that puts library modules and other modules together into a runnable program

You can then run the program.

Note: C and C++ has a long history. Some of the earlier versions of C and C++ would start the main like:

```
void main(int argc, char** argv)
Or: void main(void)
```

Pseudocode

The process of programming needs to have a way for the programmer to "plan" her/his program. Just jumping in and writing the program directly (i.e. coding) is not the most efficient way.

One way of doing this is to get an algorithm that is a step-by-step sequence of instructions describing how to do a task. There are various ways of doing it. In this class, we'll use pseudocode that is an English-like way of describing the program BEFORE you code the program. Trust me, it goes much faster especially with more complicated programs.

We'll go over pseudocode shortly.

Formatting

To format in C++, you use the iomanip functions for floating point numbers. The basic commands you need for lab 1 are:

1. setprecision – sets how many digits to the right of the decimal point you want (only do once).
2. fixed – tells the formatter to ALWAYS put in the decimal point (prevents problems with formatting, only do once).
3. setw – sets the number of spaces to use for the next field (have to do every time for every field).

Example program:

```
#include <iostream>
#include <iomanip>
```

```
using namespace std;

int main()
{
   double number;
   cout << "Enter a floating point number: ";
   cin >> number;
   cout << setprecision(2) << fixed;
   cout << "Number formatted is " << setw(6) << number << endl;
   cout << "Number doubled is " << setw(6) << 2.0*number << endl;
   system("pause");
   return 0;
}
```

Escape sequences

Escape sequence	**Character generated**	**Description**
\n	Newline	Go to new line
\t	Horizontal tab	Go to next horizontal tab
\v	Vertical tab	Go to next vertical tab
\b	Backspace	Go back one space
\r	Carriage return	Have cursor go to start of current line
\f	Form feed	Generate a form feed
\a	Alert	Generate alert (usually a bell sound)
\\	Backslash	Generate a backslash
\?	Question mark	Generate a question mark
\’	Single quote	Generate a single quote
\”	Double quote	Generate a double quote
\nnn	Octal number	Starts with 0 and will translate octal value into an ASCII character from the ASCII code chart
\xhhh	Hexadecimal number	Will take a hexadecimal value and translate into an ASCII character from the ASCII code chart

10 rules for programmers

(modified from Kristina Shroyer’s notes)

1. Keep Your Cool
2. Work When You are Rested
3. **KEEP IT SIMPLE (the KISS principle)**
4. Give help/Get help
   - On Homework – **NOT** on tests
   - What is help?

- What is **NOT** help
  - Copying code
  - Copying code and only changing variable names or the order of certain statements
  - People who copy homework almost always do very poorly on the programming tests
- ***Ask me in Lab, Come to Office Hours or Email***

5. Study and Know the Rules for the Language
   - Syntax
   - Do not use globals (i.e. put variables outside the functions)
   - Do not use gotos (if you don't know what a goto is, don't worry about it)
6. Learn the Development Environment and Tools
   - Microsoft Visual C++ ***2019 (For Mac users, you can use Xcode)***
     - We'll use the full Microsoft Visual Studio version in class and you can use either of the following at home
       - Use Visual Studio Community, the free version by clicking on the following for a download: https://www.visualstudio.com/en-us/products/visual-studio-community-vs.aspx
       - May also use the older Visual Studio Express 2015
       - Mac users can download Xcode from the Apple store for free
       - Do NOT buy a compiler!
     - **Must use a real world compiler**
       - Part of the goal of this class is for you to go out of it knowing a real world, widely used IDE for C++
       - Xcode will also work. However, if you want to use Visual Studio on your Mac, you should use Boot Camp to boot your computer up using Windows and then use Microsoft Visual C++. Also, you can use Parallels, a virtual machine program, to run Windows and MacOSX at the same time (though you have to buy Parallels).
7. Understand the Problem You are Trying to Solve
8. **BUILD AND TEST YOUR SOFTWARE IN STEPS**
   - Compile Often
9. Save Early/Save Often
10. Practice and Study
    - Know both how to program and the programming concepts we discuss in lecture
      - Objective Exams
      - Programming Exams

Hexadecimal, octal, and binary

The computer works in binary at its most basic level. That is a representation using a power of 2. A decimal 12 would translate into binary as: 1100
That can be represented as: 1*2^3 + 1*2^2 + 0*2^1 + 0 * 2^0 = 8 + 4 + 0 + 0 = 12
But writing out all those bits is tedious so octal was invented that has digits from 0 to 7 (i.e. 8 values – has to be divisible by w). So you have 0, 1, 2, 3, 4, 5, 6, and 7 (no digit 8 or 9). That can be represented in 3 bits, so 12 in binary is: 001 100 which in octal is 014. Note, HAVE to put in the leading zero to state "octal" to C++.
However, it was a bit of a hassle with a LOT of bits. So it was extended to be 4 bits and not 3 or into Hexadecimal or 6 plus 10 digits. They used part of the alphabet to represent the extra 6 digits to get 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.
The 12 decimal grouping by 4 bits is 1100 which is 0xC. The "0x" says it is hexadecimal to C++.

Miscellaneous

When submitting your labs, just submit your ".cpp" and ".h" files(if you have any). Do NOT submit the entire project!

You will find your files under the project name, project name (that's not a stutter).

When you exit Visual studio, do a save all (just in case).

To assign a large value to a smaller variable, you use a cast operation that has the format of: variable = (cast type) expression

Example program:

```
#include <iostream>

using namespace std;

int main()
{
   double large = 13.942;
   int smaller;
   smaller = (int) large;
   cout << "From the large variable of " << large << endl;
   cout << "The smaller is now " << smaller << endl;
   return 0;
}
```

Global variables are discouraged. However, global constants are encouraged. And if you use a global constant, use the convention of having the name all upper case.

Constants are very useful in allowing easy changing of values (i.e. you don't have to hunt and change every instance of a value – just one place) and once they have a value set, they cannot be changed (useful to remove bugs).

For instance, if we took the previous program and used a global constant, it would look like:

```
#include <iostream>

using namespace std;

 const  double LARGE = 13.942;

int main()
{
   int smaller;
   smaller = (int) LARGE;
   cout << "From the large constant of " << LARGE << endl;
   cout << "The smaller variable is now " << smaller << endl;
   return 0;
}
```