CISC/CMPE320

- Reminders:
- CMPE320 students in onQ yet?
- Next week's tutorial? I'll let you know in Thursday's lecture…
- Course Web Site:
  http://research.cs.queensu.ca/home/cisc320
- Are you able to run a C++ console program in your IDE of choice?

Today

- "Hello World" Ritual and some explanation.
- Some history of C++.

Demo - "Hello World" in C++

- For simple console demos I'm going to use the Eclipse CDT.
- A few notes:
  - Make sure your project has the proper "Includes" links.
  - You should provide the extension ".cpp" for a source code file, or ".h" for a header, or declaration file.
  - Suggest that you turn off automatic building. (Off by default…)
  - But, don't forget to save, then build before you run.

Some Explanation

- **`#include`** is a preprocessor directive that is used to include external code into your program.
- **`iostream`** is a library of I/O streams (makes sense!).
- A **`namespace`** is a collection of name definitions. (The same name can be used in separate namespaces). **`using namespace std;`** says that your program is using names from the **`std`** (or "standard") namespace.
- Another way to do this is to use the **`::`** "scope resolution operator", as in **`std::cout`** (but this gets to be a pain of you are using a lot of names…)

Some Explanation, Cont.

- **`int main()`** is a function header.
- Code within the function is contained between **`{ }`**
- **`cout`** is an object from the **`iostream`** library that represents the console output stream.
- **`<<`** is the "insertion operator" and is used to insert the supplied string literal into the **`cout`** object.
- **`endl`** stands in for a line feed character.
- In ISO C++ the **`return 0;`** is supposed to be optional, but some older compilers still require it.

Some More Explanation…

- Note that older style C libraries still use the ".h" extension, as in **`stdio.h`**.
- The newer library uses **`cstdio`** instead.
- Sometimes you just don't know, so you have to experiment. For example, from the STL, you could try:
  - **`Vector`**
  - **`vector.h`**
  - **`vector`**
- One of them should work!
- I will always try to use the latest library names.

Aside:How to Run C++11 Code in Eclipse

- Assuming the MinGW (or GCC) compiler.
- To change for just a project:
- C/C++ Build -> Settings -> Tool Settings -> GCC C++ Compiler -> Miscellaneous -> Other Flags. Put "-std=c++11" at the end (after a space).
- Or choose "Dialect" and the C++11 option.
- (*Don't do both of the above.*)
- Click on "Apply" after the change.

History of C++

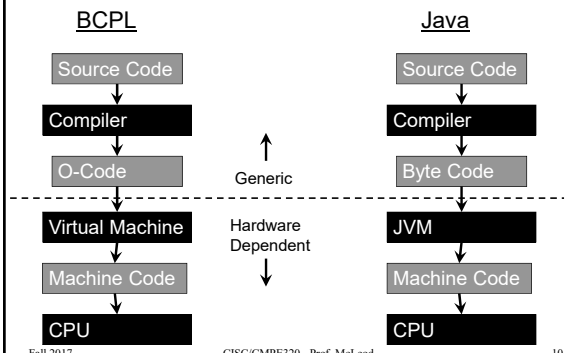
In the beginning there was C…

Not really, but let's not go back too far…

History of C++ - BCPL

- 1966 is before you were born, so let's start there:
- BCPL or "Basic Computer Programming Language" designed by Martin Richards of the University of Cambridge.
- Used to write compilers.
- It was the first language that created "O-code" before executable code:

Aside – "O-Code"



"O-Code", Cont.

- Before BCPL, when they wanted to run on a new platform, the entire compiler program had to be re-written (took 5 man-months…).
- Now only the part (about 1/5 of the total code) that generated the machine code – the "Virtual Machine" had to be changed, the part that generated the generic O-Code did not have to change.
- Radical Idea!
- (Also the first language to use { } and // for comments.)

B

- Written at Bell Labs by Ken Thompson with contributions by Dennis Ritchie. First appeared in 1969.
- Stripped down version of BCPL to run on "minicomputers" like the DEC PDP-11.
- Had a single data type, the "word" that was an integer most of the time, but could also be a memory reference.
- Gradually evolved into C in the early 70's.

Thompson & Ritchie

- Ken Thompson (seated) and Dennis Ritchie working on a DEC PDP11 in 1970:
- They also invented the UNIX OS.
- (The PDP11 was a 16 bit machine.)

C

- Dennis Ritchie, of Bell Labs, is given credit for C in 1972.
- A procedural language used for systems programming (compilers for example) and for embedded systems.
- Gives low level access to memory and efficient translation to machine code.
- No longer have to code in Assembly language!
- Used on any kind of computer during the late 70's and early 80's including the IBM-PC.
- In 1983 the language was standardized by ANSI, giving birth to "ANSI C", also called "C89".

C With Classes

- Was developed by Bjarne Stroustrup at Bell Labs starting in 1979, as an enhancement to C.
- Added to C:
  - classes
  - virtual functions
  - operator overloading
  - multiple inheritance
  - templates
  - exception handling, etc.
- In 1983 the name of the language was changed to C++ (the increment operator…).

C++ Standards

- It took a long time, but the language was first standardized in 1998 by ISO.
- The standard was updated in 2003 and amended in 2005 (310 pages, without the library!).
- Another standard was completed in 2011: "C++11" (August 12, 2011, 1338 pages!).
- C++14 came out on August 18, 2014. A smaller set of changes from C++11.
- Work is underway on C++17.

C++ Standards, Cont.

- But:
- It takes a while (and a considerable amount of work) to update compilers to the new standards.
- See:

http://en.cppreference.com/w/cpp/compiler_support for a summary of how various compilers are doing.

- GCC version 4.8.1 and later is now up-to-date with C++11. We are using version 6.3.0
- Other big compilers like Visual Studio and Clang are also now up-to-date with C++11.

C++ Standards, Cont.

- Compilation is *not* standardized, just the syntax.
- How does Java compare, for example?

The "Philosophy" of C++

- From "The Design and Evolution of C++" by (*who else!*) Bjarne Stroustrup, 1994:
- C++ is designed to be a statically typed, general-purpose language that is as efficient and portable as C.
- C++ is designed to directly and comprehensively support multiple programming styles (procedural programming, data abstraction, object-oriented programming, and generic programming).
- C++ is designed to give the programmer choice, even if this makes it possible for the programmer to choose incorrectly.

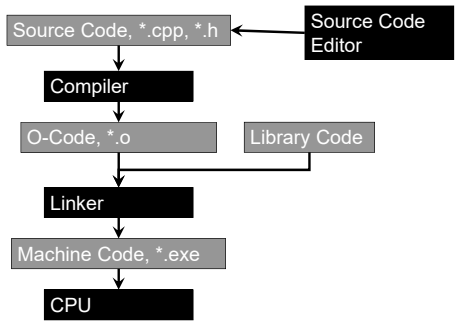
The "Philosophy" of C++, Cont.

- C++ is designed to be as compatible with C as possible, therefore providing a smooth transition from C.
- C++ avoids features that are platform specific or not general purpose.
- C++ does not incur overhead for features that are not used (the "zero-overhead principle").
- C++ is designed to function without a sophisticated programming environment.

ISO C++ and C

- The C++ standard library has incorporated the standard C library with just a few optimizations.
- C++ also has the "Standard Template Library", or "STL", that contains many pre-defined data structures, for example.
- Generally, well-written C code will run in C++.
- But some C "things" will not work in C++ (for example, in C you can use **`new`** or **`class`** as variable names, since they are not C keywords.)

Running a C++ Program



Running a C++ Program, Cont.

- Thankfully, an IDE simplifies this process for you.
- A project (a set of source code and resource files) is "built" – which consists of both compilation and linking.
- MinGW ("Minimalist GNU for Windows"), for example uses the g++.exe program for this. The IDE makes up the command line calls to this program.
- Building creates a *.o file and then a *.exe file.
- Then you run the *.exe program.
- Each stage of this process can unearth errors.

Who Reports the Errors?

- An Eclipse pre-compiler does a pretty good (but not perfect!) job of reporting syntax errors and warnings. Errors may prevent building. You can (but should not!) ignore warnings.
- Build errors are reported by the compiler and are seen in the console window.
- Run-time errors are reported by the OS, which is running the *.exe file.