Introduction – Part 1

CS A150 - C++ Programming 1

What Is Programming?

- A **program** tells computers the sequence of steps needed to fulfill a task
- Programming — act of designing and implementing programs
- Most ***users*** are **not** ***programmers***
- Programming is an essential skill for a computer scientist
- ***Not*** the only skill required to be a successful computer scientist

Introduction to C++

- C++ Origins
  - **Low-level** languages
    - Machine, assembly
  - **High-level** languages
    - C, C++, ADA, COBOL, FORTRAN
  - **Object-Oriented-Programming** in C++
- C++ Terminology
  - ***Programs*** and ***functions***
  - Basic Input/Output (I/O) with **cin** and **cout**

Machine Instructions

- Extremely primitive:

```
    Move memory contents to a register
    Subtract 100 from a register
    If result is positive, go (jump) to another
      instruction
```

- Encoded as numbers:

```
      161 40000 45 100 127 11280
```

- Thousands of instructions for a simple program
- Each processor has its own set of machine instructions
- Adding numeric codes manually is tedious and error prone

Assembler

- Uses computer to translate
- Assigns short names to commands:

```
mov 40000, %eax
sub 100, %eax
jg 11280
```

- Makes reading easier for humans
- Translated into machine instructions by the assembler
- Can give names to memory locations
- Still processor dependent
- Still many instructions

Higher-level Languages

- Easiest for humans to read and write:

```
    if( intRate > 100 )
         cout << “Interest rate error”;
```

- Translated by **compilers** into **machine instructions**
  - Very sophisticated programs
  - Find **memory locations** for all **variable names**
- **Independent** of the underlying hardware

Programming Languages

- Designed for a variety of purposes
- Machine **independent** (generally)
- Much **easier** to read, but…
  - Much **stricter** than spoken languages
  - Compilers don't like to guess
- There are *thousands* of programming languages
- Differences in programming languages are sometimes slight (Java and C++), other times substantial (C++ and Lisp)

The Evolution of C++

- Many languages are created with a ***specific* purpose**
  - database processing
  - artificial intelligence
  - multimedia processing
- ***General* purpose languages** can be used for *any* task
  - C: developed to be translated efficiently into fast machine code, with minimal housekeeping overhead
  - C++: adds "**object oriented programming**“ to **C**

History of C

- Initially designed in 1972 (Kernighan & Ritchie)
- Features were added in response to perceived shortcomings
- Resulted in different dialects
  - Bad for portability
- 1989 — **ANSI** standard of **C** completed
  - **A**merican **N**ational **S**tandards **I**nstitute
    - The institute has been coordinated the private sector standardization system since 1918. It is a non-profit organization formed by engineering societies and government agencies.

History of C++ (cont.)

- 1979 — **Bjarne Stroustrup** of AT&T adds **object oriented** features to C, called ***C with Classes***
- 1985 — rename to ***C++***
- … Other features added over the years
- 1998 — **ISO C++** standard published
- 2003 — minor revision to the ISO standard

- Stroustrup's home page

C++

- Can you learn everything about C++?

Writing a Simple Program

- C++ is **case sensitive**
- C++ has free-form layout
  - **BUT** follow standard program layout for **readability**

    and of course, to get a good grade!
- Do write a **name header** at the beginning of your file (as indicated on the syllabus)

```
/*
  Firstname Lastname      // If a group, write all names
  CS A150
  Date                    // Month must be written in letters

  Exercise/Exam #
*/
```

Writing a Simple Program (cont.)

- `#include<iostream>`
  - Read the **iostream** (input/output) header file

- `using namespace std;`
  - All **names** in the program belong to the "standard namespace"
  - This is to avoid writing **std::cout** and instead simply write **cout**

- `int main () { ... }`
  - Defines the **main** function
  - ***All*** C/C++ programs have a **main()** function
    - It is the **entry way** to your program

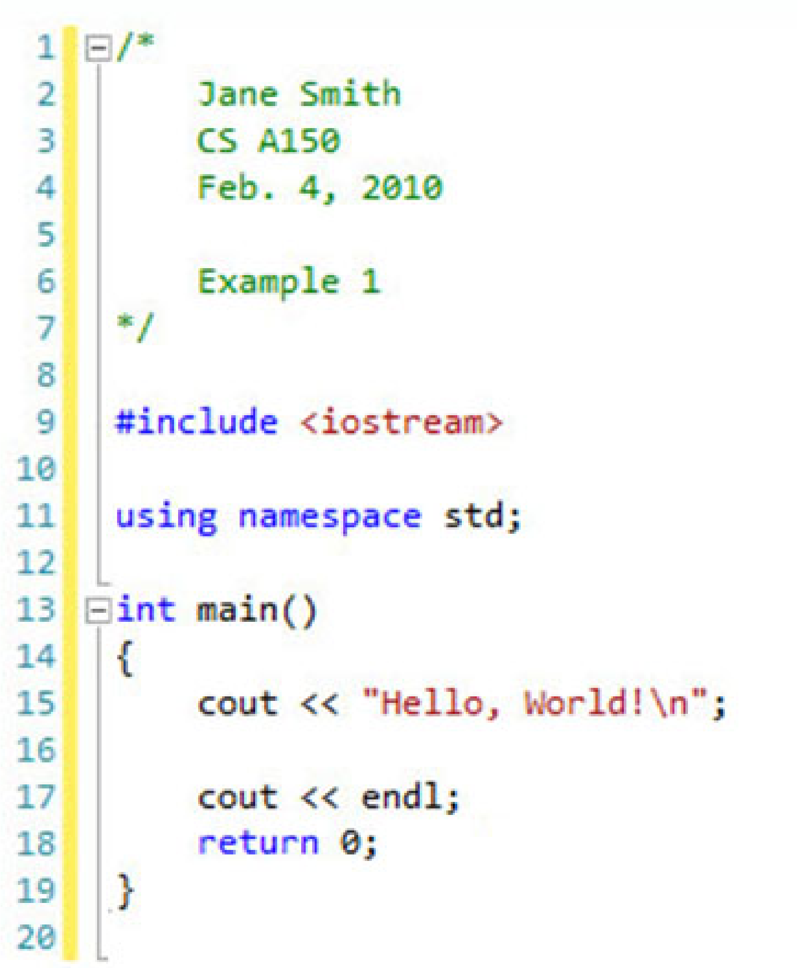
Writing a Simple Program (cont.)

- statements (instructions) are written inside the ***body*** of the main function:
  - Executed one by one, in order
  - Each statement ends with a semicolon (**;**)

- `cout << endl;`
  - outputs a line before ending the program

- `return 0;`
  - The end of the **main()** function
  - The **0** value signals that the program ran successfully

Example 1



```
/*
    Jane Smith
    CS A150
    Feb. 4, 2010

    Example 1
*/

#include <iostream>

using namespace std;

int main()
{
    cout << "Hello, World!\n";

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

Book Example

- Display 1.1 – A Sample C++ Program

C++ Variables

- C++ **Identifiers**
  - **Keywords**/**reserved** words vs. identifiers
  - The name we assign to variables, constants, functions, classes, etc.
  - Give a meaningful names!
- **Variables**
  - A memory location to store data for a program
  - Must declare all data before using it in a program

Simple Data Types

- Integer numbers
  - **`short`** (*typically* 2 bytes)
  - **`unsigned short`** (*typically* 2 bytes)
  - **`int`** (*typically* 4 bytes)
  - **`unsigned int`** (*typically* 4 bytes)
- Floating-point numbers (*decimals*):
  - **`float`** (*typically* 4 bytes)
  - **`long double`** (*typically* 8 bytes)
  - **`double`** (*typically* 8 bytes)

SIMPLE DATA TYPES (CONT.)

- Other types:
  - `char` (1 byte)
    - Represents all ASCII characters
    - Can be used as an integer type, but ***not*** recommended
  - `bool` (1 byte)
    - Represents either **true** or **false**

Boolean Type

- C++ has the boolean type: **bool**
  - Can be true or false
- C does ***not*** have a bool type
  - we used any *integer* type (char, short, int, etc.)
- Note:
  - For compatibility reasons,
    in C++ any ***non-zero*** value evaluates to ***true***

IDENTIFIERS

- **Identifiers** are symbolic names assigned to locations to store values
  - Make programs *easier to read* and *easier to update*
- We will use the following conventions for naming variables:
  - Start with a letter
  - Use **descriptive** names
  - Use **camelCase** format when more than one word (for example, **`compoundInterestRate`**)
  - **No** special characters
  - **No** reserved words
- Remember that C++ is ***case sensitive***

String Type

- C++ has a data type of “**string**” to store sequence of characters
  - It is not a primitive data type

    (we will make the distinction later)
  - Must include at the top of the program

    ```
    #include <string>
    ```

  - The “**+**” (plus) operator on strings concatenates two strings together

Good Programming Practice

- **Avoid** identifiers that begin with the underscore character, as these can lead to linker errors, since many code **libraries** use names that begin with underscores.

- To improve **readability**:
  - **Avoid** using abbreviations in identifiers.

ASSIGNING DATA

- Initializing data in **declaration** statement
  - Results "**undefined**" if you don't!

```
int myValue = 0;
```

- Assigning data during execution
  - **Lvalues** (left-side) & **Rvalues** (right-side)
    - Lvalues must be variables
    - Rvalues can be any expression
    - Example:

```
distance = rate * time;
```

Lvalue ➔ "distance"
Rvalue ➔ "rate * time"

Assignment Operator

- Assigns a value to a variable =

- Can use different formats

```
int pennies;
pennies = 1;

int nickels = 5;

int dimes = 10,
    quarters = 25;

int halfDollar (50), dollar (100);
```

Note the commas

Combining Assignment and Arithmetic

- Use shorthand

```
total += 3.25;
```

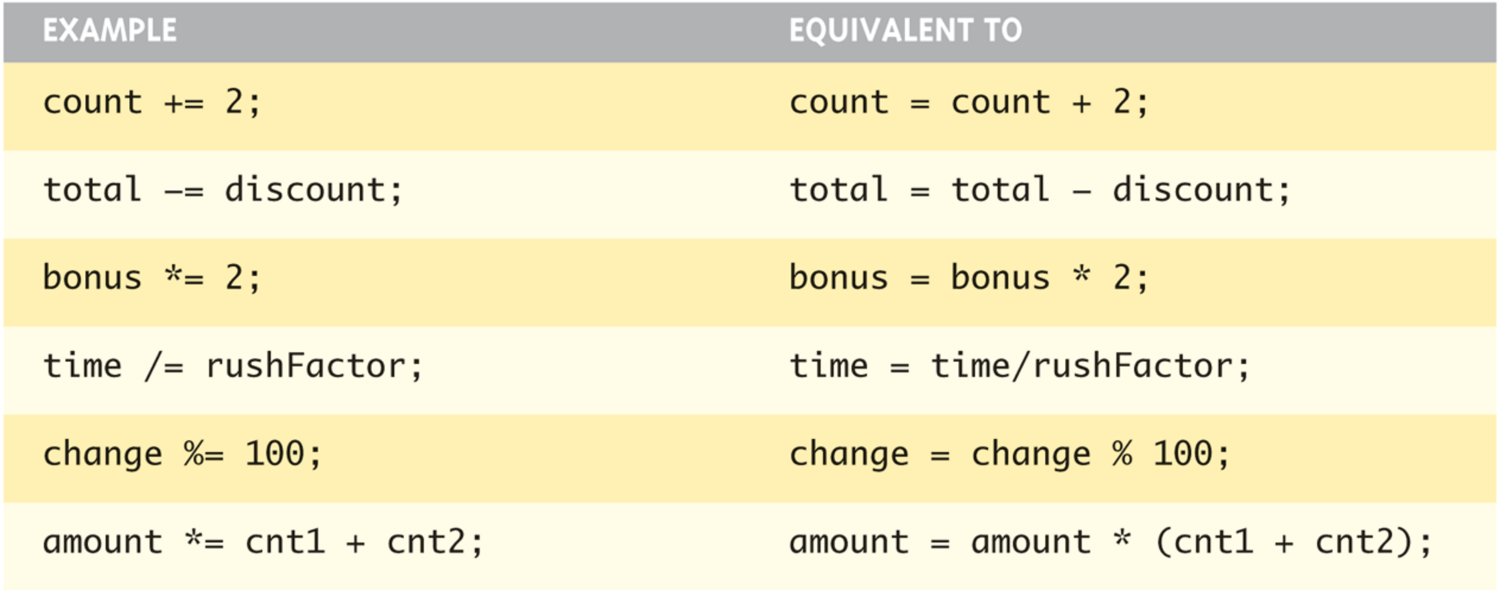
- Is the same as:

```
total = total + 3.25;
```

- Same as other operations:

```
-=  *=  /=  %=
```

Assigning Data: Shorthand Notations



EXAMPLE	EQUIVALENT TO
`count += 2;`	`count = count + 2;`
`total –= discount;`	`total = total – discount;`
`bonus *= 2;`	`bonus = bonus * 2;`
`time /= rushFactor;`	`time = time/rushFactor;`
`change %= 100;`	`change = change % 100;`
`amount *= cnt1 + cnt2;`	`amount = amount * (cnt1 + cnt2);`

Assignment Compatibility

- Type mismatches
  - **General Rule:** Cannot place value of one type into variable of another type

```
int intVar = 2.99; // 2 is assigned to intVar!
```

  - Only integer part "fits“ in the variable
  - Called "**implicit**" or "**automatic type conversion**"

Literals

- Example of **literals** are:

```
2                // Literal constant int
5.75             // Literal constant double
"Z"              // Literal constant char
"Hello World"    // Literal constant string
```

- **Cannot** change values during execution
- Called "literals" because you "literally typed" them in your program!

Escape Sequences

- "Extend" character set
- Backslash \ preceding a character
  - Instructs compiler that a special "escape character" is coming
  - Following character is treated as "escape sequence char"

(Display on next slide)

Common Escape Sequences

SEQUENCE	MEANING
\n	New line
\t	(Horizontal) Tab – Advances the cursor to the next tab stop.
\\	Backlash – Allows you to place a backlash in a quoted expression.
\″	Double quote – Mostly used to place a double quote inside a quoted string.

Constants

- Literal constants provide little meaning

```
double newBalance = balance + (balance * 0.75);
```

  0.75 tells nothing about what it represents

- Use named constants instead

```
const double INTEREST_RATE = 2.5;
```

- Declared w/the keyword **const**
- Can ***never*** change value assigned
- Easier to modify/maintain code
- Must be initialized when declared
- Identifier should be **all capital letters**
  - Words should be separated by an underscore
- **Constants** are usually declared **globally**

Book Example

- Display 1.4 – Named Constant

QUESTIONS?

(Introdunction – Part 1)