BASICS I

ADAM SWEENEY
CS 211

INTRODUCTION

- A little history, a peek behind the curtain, and some foundational C++

AGENDA

- A brief history of C++
- How the sausage gets made
- Variables & Expressions
- Input & Output

A BRIEF HISTORY OF C++

C WITH CLASSES

- Started in 1979 by Bjarne Stroustrup
- Originally a superset of C, called C with Classes
- Name changed in 1983 to C++
- First standardized version of C++ released in 1998
  - Many shortcomings
- Problems addressed in new standard known as C++03
- Further evolution proposed in 2005
- Finally released in 2011 as C++11
- New version released has since been released every 3 years

HOW THE SAUSAGE GETS MADE

HOW DOES OUR CODE ACTUALLY WORK?

- What do computers understand?
  - 1's and 0's only
- How do we go from `std::cout << “Hello world!\n”;` to 1's and 0's?
- Short answer: a compiler turns our code into something the computer understands (g++, clang++, msvc)
- Longer answer: 5 distinct phases to go from C++ to machine code
  - We'll take a quick look at them
  - Taken from clang documentation

PREPROCESSOR

- File gets read, preprocessor directives get expanded (#include expansion, macro expansion)
  - g++|clang++ -E <source file>

PARSING AND SEMANTIC ANALYSIS

- Code is checked for proper syntax and well-formed code
- This stage will generate most errors seen in this course
  - g++|clang++ -fsyntax-only <source file>

CODE GENERATION & OPTIMIZATION

- Code translation occurs at this stage
- Compiler optimizations occur here, as well as target-specific code generation

ASSEMBLER

- Assembler translates compiler output into what's typically called an object file
- Stopping at this point is important for larger projects (CS 311+)
  - Controlling compilation becomes important to manage compilation of large projects where compile time gets in the way of work
  - g++|clang++ -c <source file>

LINKER

- Merge multiple object files into an executable
  - g++|clang++ <source file>

VARIABLES & EXPRESSIONS

WE NEED DATA

- Variables are how we store data in a program
- 3 aspects to a variable
  - Type
    - Is it an integer, a string, a double, character, etc?
  - Name
    - We name our variables to easily tell what kind of information it contains
  - Value
    - The actual data held

NAMING VARIABLES

- How can we name variables?
  - Must start with a letter or underscore
  - Rest of name can contain letters, numbers, or underscores
- How <u>should</u> we name variables?
  - Use a consistent naming scheme (camelCase)
  - Be descriptive and succinct
  - Ideal for us to always start variable names with a lowercase letter
    - Helps readability in later courses
    - Classes, structures typically start with an uppercase letter

QUICK NOTE

- C++ is case-sensitive
    - account, ACCOUNT, Account, aCcOuNt are all different names

DECLARING VARIABLES

- Computers don’t like surprises
- Before we can use a variable, we must declare its existence
- `TYPE NAME [= INITIAL_VALUE];`
  - `int numBoxes;`
  - `int numBoxes = 0;`
  - `double averageBoxes;`

VARIABLE TYPES

- Integer
  - Whole numbers {…, -1, 0, 1, …}
- Double-precision floating point
  - Can represent decimal values (3.14, etc.)
- Character
  - Single character, needs single quotes (‘A’, ‘2’, ‘#’, etc.)

- Boolean
  - true or false (1 or 0)
- String
  - Two ways to declare in C++
  - char greet[] = “Hello”;
    - From C
  - std::string greet = “Hello”;
    - Not a native type, but part of C++ Standard Library

EXPRESS YOURSELF

- Expression: a sequence of operator and their operands, that specifies a computation
  - Given 2 + 2, the operator is ‘+’, and the operands are 2 and 2
- Three main types of expressions
  - Arithmetic
  - Comparative
  - Logical

ARITHMETIC EXPRESSIONS

Operator	Action Taken
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo (remainder after division)

SHORTHAND ARITHMETIC

Operator	Equivalent to (Given int x)
++	x = x + 1
--	x = x - 1
+=	x += 2 → x = x + 2
-=	x -= 2 → x = x - 2
*=	x *= 2 → x = x * 2
/=	x /= 2 → x = x / 2
%=	x %= 2 → x = x % 2

COMPARATIVE EXPRESSIONS

Operator	Comparison Made
==	Equality
!=	Non-Equality
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

LOGICAL EXPRESSIONS

Operator	Definition
!a	NOT a
a && b	a AND b
a || b	a OR b

- Used in flow control

QUICK EXPLANATIONS OF LOGICAL OPERATORS

A	!A
0	1
1	0

<table>
<tr><th>A</th><th>B</th><th>A && B</th><th>A || B</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td></tr>
</table>

INPUT & OUTPUT

STREAMS

- Input and output are treated as streams
- We can place one type of information in the stream at a time
- The operators show us where the information is flowing
  - Insertion operator: `std::cout << “Hello World!\n”;`
  - Extraction operator: `std::cin >> x;`

SPECIAL CHARACTERS

- Special characters allows customizing of an output stream
- Special characters start with a backslash, called an escape character

Special Character	Interpreted As
\n	New line
\t	Horizontal tab
\\	Backslash
\”	Double quote