Unit 3

Constants, Expressions, and Variables

C++ Output (with 'cout')
Unit Objectives

• List the various C data types
• Identify what type a constant is
• Know how to write constants in the appropriate C++ syntax
• Know the C++ operators and their order of operations
• Write basic output statements of text and constants using cout
C/C++ Program Format/Structure

• **Comments**
  - Anywhere in the code
  - C-Style => "/*" and "*/"
  - C++ Style => "//"

• **Compiler Directives**
  - #includes tell compiler what other library functions you plan on using
  - 'using namespace std;' -- Just do it for now!

• **main() function**
  - **Start**ing point of execution for the program
  - All code/statements in C must be inside a function
  - Statements execute one after the next and end with a semicolon (;)
  - Ends with a 'return 0;' statement

• **Other functions**
  - printName() is a function that can be "called"/"invoked" from main or any other function

```cpp
#include <iostream>
#include <cmath>
using namespace std;

// Code is organized into units called functions
void printName()
{
  cout << "Tommy Trojan" << endl;
}

int main()
{
  cout << "Hello: " << endl;
  printName();
  printName();
  return 0;
}
```

Hello:
Tommy Trojan
Tommy Trojan
# Review C Integer Data Types

- Integer Types (signed by default... *unsigned* with optional leading keyword)

<table>
<thead>
<tr>
<th>C Type (Signed)</th>
<th>C Type (Unsigned)</th>
<th>Bytes</th>
<th>Bits</th>
<th>Signed Range</th>
<th>Unsigned Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>unsigned char</td>
<td>1</td>
<td>8</td>
<td>-128 to +127</td>
<td>0 to 255</td>
</tr>
<tr>
<td>short</td>
<td>unsigned short</td>
<td>2</td>
<td>16</td>
<td>-32768 to +32767</td>
<td>0 to 65535</td>
</tr>
<tr>
<td>int</td>
<td>unsigned int</td>
<td>4</td>
<td>32</td>
<td>-2 billion to +2 billion</td>
<td>0 to 4 billion</td>
</tr>
<tr>
<td>long</td>
<td>unsigned long</td>
<td>8</td>
<td>64</td>
<td>-8<em>10^{18} to +8</em>10^{18}</td>
<td>0 to 16*10^{18}</td>
</tr>
</tbody>
</table>

- Text characters are usually represented with some kind of binary code (mapping of character to a binary number such as 'a' = 01100001 bin = 97 dec)

- ASCII = Traditionally an 8-bit code
  - How many combinations (i.e. characters)?
  - English only

- UNICODE = 16-bit code
  - How many combinations?
  - Most languages w/ an alphabet

- In C/C++ a single printing/text character must appear between single-quotes (')
  - Example: 'a', '!', 'Z'

http://www.theasciicode.com.ar/
Review

• Show how "Hi!\n" would be stored in the memory below
  – Use decimal to represent each byte
  – Remember how we terminate a string
What About Rational/Real #’s

- Previous binary system assumed binary point was fixed at the far right of the number, so we can't represent decimals
  - 10010. *(implied binary point)*
- Consider scientific notation:
  - Avogadro’s Number: +6.0247 * 10^{23}
  - Planck’s Constant: +6.6254 * 10^{-27}
- Can one representation scheme represent such a wide range?
  - Yes! **Floating Point**
  - Represents the sign, significant digits (fraction), exponent as separate bit fields
- Decimal: ±D.DDD * 10^{±exp}
- Binary: ±b.bbbbb * 2^{±exp}
C Floating Point Types

- **float** and **double** types:

<table>
<thead>
<tr>
<th>C Type</th>
<th>Bytes</th>
<th>Bits</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>4</td>
<td>32</td>
<td>±7 significant digits * 10^+/−38</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
<td>64</td>
<td>±16 significant digits * 10^+/−308</td>
</tr>
</tbody>
</table>

- Prefer **double** over **float**
  - Many compilers will upgrade floats to doubles anyhow

- Don't use floating-point if you don't need to
  - It suffers from rounding error
  - Some additional time overhead to perform arithmetic operations
C CONSTANTS & DATA TYPES
Constants

• Integer: 496, 10005, -234
• Double: 12.0, -16., 0.23, -2.5E-1, 4e-2
• Characters (char type): enclosed in single quotes
  - Printing characters: 'a', '5', 'B', '!
  - Non-printing special characters use "escape" sequence (i.e. preceded by a \):
    '\n' (newline/enter), '\t' (tab), '\\' (slash), '"' (apostrophe)
• C-Strings
  - 0 or more characters between double quotes
    "hi1\n", "12345", "b", "\tAns. is %d"
  - Ends with a '\0'=NULL character added as the last byte/character to allow code to delimit the end of the string
• Boolean (C++ only): true, false
  - Physical representation: 0 = false, (Non-zero) = true

String Example (Memory Layout)
## You're Just My Type

- Indicate which constants are matched with the correct type.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Right / Wrong</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.0</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>'a'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>&quot;abc&quot;</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>char</td>
<td></td>
</tr>
<tr>
<td>&quot;5.0&quot;</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>'5'</td>
<td>int</td>
<td></td>
</tr>
</tbody>
</table>

Solutions are provided at the end of the slide packet.
EXPRESSIONS
Arithmetic Operators

- Addition, subtraction, multiplication work as expected for both integer and floating point types
- Modulus is only defined for integers

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td></td>
<td>(Integer vs. Double division)</td>
</tr>
<tr>
<td>%</td>
<td>Modulus (remainder)</td>
</tr>
<tr>
<td></td>
<td>[for integers only]</td>
</tr>
</tbody>
</table>

\[
10 \% 3 = __ \\
17 \% 10 = __
\]
Precedence

• Order of operations/evaluation of an expression

• Top Priority = highest (done first)

• Notice operations with the same level or precedence usually are evaluated left to right (explained at bottom)

• Evaluate:
  – 2* -4 - 3 + 5 % 2;

• Tips:
  – Use parenthesis to add clarity
  – Add a space between literals

Operators (grouped by precedence)

<table>
<thead>
<tr>
<th>struct member operator</th>
<th>name.member</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct member through pointer</td>
<td>pointer-&gt;member</td>
</tr>
<tr>
<td>increment, decrement</td>
<td>++, --</td>
</tr>
<tr>
<td>plus, minus, logical not, bitwise not</td>
<td>+, -, !, ~</td>
</tr>
<tr>
<td>indirect via pointer, address of object</td>
<td>*pointer, &amp;name</td>
</tr>
<tr>
<td>cast expression to type</td>
<td>(type) expr</td>
</tr>
<tr>
<td>size of an object</td>
<td>sizeof</td>
</tr>
<tr>
<td>multiply, divide, modulus (remainder)</td>
<td>*, /, %</td>
</tr>
<tr>
<td>add, subtract</td>
<td>+, -</td>
</tr>
<tr>
<td>left, right shift [bit ops]</td>
<td>&lt;&lt;, &gt;&gt;</td>
</tr>
<tr>
<td>relational comparisons</td>
<td>&gt;, &gt;=, &lt;, &lt;=</td>
</tr>
<tr>
<td>equality comparisons</td>
<td>==, !=</td>
</tr>
<tr>
<td>and [bit op]</td>
<td>&amp;</td>
</tr>
<tr>
<td>exclusive or [bit op]</td>
<td>^</td>
</tr>
<tr>
<td>or (inclusive) [bit op]</td>
<td></td>
</tr>
<tr>
<td>logical and</td>
<td>&amp;&amp;</td>
</tr>
<tr>
<td>logical or</td>
<td></td>
</tr>
<tr>
<td>conditional expression</td>
<td>expr1 ? expr2 : expr3</td>
</tr>
<tr>
<td>assignment operators</td>
<td>+=, -=, *=, ...</td>
</tr>
<tr>
<td>expression evaluation separator</td>
<td>,</td>
</tr>
</tbody>
</table>

Unary operators, conditional expression and assignment operators group right to left; all others group left to right.
Division

• Computers perform division differently based on the type of values used as inputs

• **Integer** Division:
  – When dividing two integral values, the result will also be an integer (any remainder/fraction will be dropped)
  – 10 / 4 = 2  
  – 52 / 10 = 5
  – 6 / 7 = 0

• **Floating-point** (Double) & Mixed Division
  – 10.0 / 4.0 = 2.5  
  – 52.0 / 10 = 5.2
  – 6 / 7.0 = 0.8571
  – Note: If one input is a double, the other will be promoted temporarily to compute the result as a double
Exercise Review

• Evaluate the following:

  25 / 3
  20 - 12 / 4 * 2
  3 - 15 % 7
  18.0 / 4
  28 - 5 / 2.0

Using 'cout'...

SIMPLE C++ OUTPUT
Output From Your Program

• To see the output in C++ we need to explicitly tell the computer to output the value using 'cout'
  – So what happens to the result of 12*3 on the first line?
• Note: 'endl' stands for end-line and causes the cursor to move to the next line of the screen

Performing computation is like having a thought. No output is generated unless you explicitly write it down.

To output a result to the screen in C++ (i.e. "write it down") we use the 'cout' command

// iostream allows access to 'cout'
#include <iostream>
using namespace std;

// Execution always starts at the main() function
int main()
{
  12 * 3; // No result printed
  cout << 12 * 3 << endl; // 36 printed
  return 0;
}
Printing Different Values & Types

- 'cout' requires appropriate use of separators between consecutive values or different types of values.
- 'cout' does not add spaces between consecutive values; you must do so explicitly.
  - Since text strings are a different value we must separate it with the '<<<' operator.
- Generally good practice to give some descriptive text with your numeric output.
  - Note: You may divide up output over multiple 'cout' statements. Unless an 'endl' or '\n' is used, the next 'cout' statement will resume where the last one left off.

```cpp
// iostream allows access to 'cout'
#include <iostream>
using namespace std;

// Execution always starts at the main() function
int main()
{
    cout << 345 << 754 << endl; // Bad
    cout << 345 << 754 << endl; // Better, but no spaces
    cout << 345 << " " << 754 << endl; // Best
    return 0;
}

// iostream allows access to 'cout'
#include <iostream>
using namespace std;

// Execution always starts at the main() function
int main()
{
    cout << "3 dozen is " << 3*12 << " items." << endl;
    cout << "There are " << 60*24*365 << " minutes";
    cout << " in a year." << endl;
    return 0;
}
```

Output:
3 dozen is 36 items.
There are 525600 minutes in a year.
**You're Just My Type**

- Indicate which constants are matched with the correct type.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
<th>Right / Wrong</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.0</td>
<td>int</td>
<td>double (.0)</td>
</tr>
<tr>
<td>5</td>
<td>int</td>
<td>int</td>
</tr>
<tr>
<td>'a'</td>
<td>string</td>
<td>char</td>
</tr>
<tr>
<td>&quot;abc&quot;</td>
<td>string</td>
<td>C-string</td>
</tr>
<tr>
<td>5.</td>
<td>double</td>
<td>float/double (. = non-integer)</td>
</tr>
<tr>
<td>5</td>
<td>char</td>
<td>Int...but if you store 5 in a char variable it'd be okay (char = some number that fits in 8-bits/1-byte)</td>
</tr>
<tr>
<td>&quot;5.0&quot;</td>
<td>double</td>
<td>C-string</td>
</tr>
<tr>
<td>'5'</td>
<td>int</td>
<td>char</td>
</tr>
</tbody>
</table>
Exercise Review

• Evaluate the following:
  – 25 / 3 = 8
  – 20 - 12 / 4 * 2 = 14
  – 3 - 15 % 7 = 2
  – 18.0 / 4 = 2.5
  – 28 - 5 / 2.0 = 25.5