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
• Know how variables are declared and assigned
• Write basic output statements of text and constants using cout
• Use cin statement to get keyboard input from the user
• Predict how cin will treat input with whitespaces and extract data
• Trace the operation of assignment statements, expressions, and cin and cout commands
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
  – Starting 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

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
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 (Integer vs. Double division)</td>
</tr>
<tr>
<td>%</td>
<td>Modulus (remainder) [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

\[(2 * -4) - 3 + (5 \% 2)\]

Operators (grouped by precedence)

<table>
<thead>
<tr>
<th>Operators</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct member operator</td>
<td>name.member</td>
</tr>
<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>indirection 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>

January 2007 v2.2. Copyright © 2007 Joseph H. Silverman
Permission is granted to make and distribute copies of this card provided the copyright notice and this permission notice are preserved on all copies.
Send comments and corrections to J.H. Silverman, Math. Dept., Brown Univ., Providence, RI 02912 USA. hjhs@math.brown.edu
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

```c++
#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;
}
```

```cpp
// 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.
The Need For Variables & Input

- Printing out constants is not very useful (nor exciting)
- In fact, we could just as easily compute the value ourselves in many situations
- The real power of computation comes when we introduce variables and user input
  - Variables provide the ability to remember and name a value for use at a later time
  - User input allows us to write general programs that work for "any" input values
  - Thus, a more powerful program would allow us to enter an arbitrary number and perform conversion to dozens

```cpp
#include <iostream>

using namespace std;

// Execution always starts at the main() function
int main()
{
    cout << "3 dozen is " << 3*12 << " items." << endl;
    // the above results in the same output as below
    cout << "3 dozen is 36 items." << endl;
    return 0;
}
```
C/C++ Variables

- Variables allow us to
  - Store a value until it is needed and change its values potentially many times
  - Associate a descriptive name with a value
- Variables are just memory locations that are reserved to store a piece of data of specific size and type
- Programmer indicates what variables they want when they write their code
  - Difference: C requires declaring all variables at the beginning of a function before any operations. C++ relaxes this requirement.
- The computer will allocate memory for those variables when the code starts to run
- We can provide initial values via '=' or leave them uninitialized

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

int main()
{
    // Sample variable declarations
    char gr = 'A';
    int x;   // uninitialized variables
             // will have a (random) garbage value until we initialize it
    x = 1;   // Initialize x's value to 1
    gr = 'B'; // Change gr's value to 'B'
}
```

Variables are actually allocated in RAM when the program is run.
C/C++ Variables

- Variables have a:
  - type [int, char, unsigned int, float, double, etc.]
  - name/identifier that the programmer will use to reference the value in that memory location [e.g. x, myVariable, num_dozens, etc.]
    - Identifiers must start with [A-Z, a-z, or an underscore '_'] and can then contain any alphanumeric character [0-9, A-Z, a-z, _] (but no punctuation other than underscores)
    - Use descriptive names (e.g. numStudents, doneFlag)
    - Avoid cryptic names (myvar1, a_thing)
  - location [the address in memory where it is allocated]
  - Value

- Reminder: You must declare a variable before using it

```
int quantity = 4;
double cost = 5.75;
cout << quantity*cost << endl;
```

**What's in a name?**

To give descriptive names we often need to use more than 1 word/term. But we can't use spaces in our identifier names. Thus, most programmers use either camel-case or snake-case to write compound names

- **Camel case**: Capitalize the first letter of each word (with the possible exception of the first word)
  - myVariable, isHighEnough
- **Snake case**: Separate each word with an underscore '_'
  - my_variable, is_high_enough
# Know Your Common Variable Types

- Variables are declared by listing their type and providing a name.
- They can be given an initial value using the '=' operator.

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

// Execution always starts at the main() function
int main()
{
    int w = -400;
    double x = 3.7;
    char y = 'a';
    bool z = false;
    cout << w << " " << x << " " << y << " " << z << endl;
    return 0;
}
```

<table>
<thead>
<tr>
<th>C Type</th>
<th>Usage</th>
<th>Bytes</th>
<th>Bits</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>Text character</td>
<td>1</td>
<td>8</td>
<td>ASCII characters -128 to +127</td>
</tr>
<tr>
<td></td>
<td>Small integral value</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bool</td>
<td>True/False value</td>
<td>1</td>
<td>8</td>
<td>true / false</td>
</tr>
<tr>
<td>int</td>
<td>Integer values</td>
<td>4</td>
<td>32</td>
<td>-2 billion to +2 billion</td>
</tr>
<tr>
<td>unsigned int</td>
<td></td>
<td></td>
<td></td>
<td>0 to +4 billion</td>
</tr>
<tr>
<td>double</td>
<td>Rational/real values</td>
<td>8</td>
<td>64</td>
<td>±16 significant digits</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>* 10^+/-308</td>
</tr>
<tr>
<td>string</td>
<td>Arbitrary text</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
When Do We Need Variables?

• When a value will be supplied and/or change at run-time (as the program executes)

• When a value is computed/updated at one time and used (many times) later

• To make the code more readable by another human

double area = (56+34) * (81*6.25);
// readability of above vs. below
double height = 56 + 34;
double width = 81 * 6.25;
double area = height * width;
What Variables Might Be Needed

• Calculator App
  – Current number input, current result

• Video playback (YouTube player)
  – Current URL, full screen, volume level
Assignment (=) Operator

• To update or change a value in a variable we use the assignment operator (=)

• Syntax:
  – variable = expression;  
    (Left-Side)  (Right-side)

• Semantics:
  – Place the resulting value of 'expression' in the memory location associated with 'variable'
  – Does not mean "compare for equality" (e.g. is w equal to 300?)
    • That is performed by the == operator

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

// Execution always starts at the main() function
int main()
{
    int w;  // variables don't have to be initialized when declared
    char x;  // be initialized when declared

    w = 300;  
    x = 'a';
    cout << w << " " << x << endl;

    w = -75;  
    x = '!';
    cout << w << " " << x << endl;
    return 0;
}
```

Order of evaluation:   right to left

```
Output:  
300  a  
-75  !
```

Assignment is one of the most common operations in programs
Assignment & Expressions

- Variables can be used in expressions and be operands for arithmetic and logic
- See inset below on how to interpret a variable's usage based on which side of the assignment operator it is used

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

// Execution always starts at the main() function
int main()
{
    int dozens = 3;
    double gpa = 2.0;
    int num = 12 * dozens;
    gpa = (2 * 4.0) + (4 * 3.7); // gpa updated to 22.8
    gpa = gpa / 6; // integer or double division?
    cout << dozens << " dozen is " << num << " items." << endl;
    cout << "Your gpa is " << gpa << endl;
    return 0;
}
```

Order of evaluation: right to left

```cpp
int x = 0;
x = x + 3;
```

Semantics of variable usage:
- Right-side of assignment: Substitute/use the current value stored in the variable
- Left-side of assignment: variable is the destination location where the result of the right side will be stored
Exercises

• What is printed by the following two programs?

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

int main()
{
    int value = 1;
    value = (value + 5) * (value - 3);
    cout << value << endl;

    double amount = 2.5;
    value = 7;
    amount = value + 6 / amount;
    cout << amount << endl;
    cout << value % 3 << endl;
    return 0;
}
```

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

int main()
{
    int x = 5;
    int y = 3;
    double z = x % y * 6 + x / y;
    cout << z << endl;
    z = 1.0 / 4 * (z - x) + y;
    cout << z << endl;
    return 0;
}
```
RECEIVING INPUT WITH CIN
Keyboard Input

- In C++, the 'cin' object is in charge of receiving input from the keyboard.
- Keyboard input is captured and stored by the OS (in an "input stream") until cin is called upon to "extract" info into a variable.
- 'cin' converts text input to desired format (e.g. integer, double, etc.)

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

int main()
{
    int dozens;
    cout << "Enter number of dozen: "
         << endl;
    cin >> dozens;
    cout << 12 * dozens << " eggs" << endl;
    return 0;
}
```
Dealing With Whitespace

- **Whitespace (def.):**
  - Characters that represent horizontal or vertical blank space. Examples: newline ('\n'), TAB ('\t'), spacebar (' ')  

- cin sequentially scans the input stream for actual characters, discarding leading whitespace characters  

- Once cin finds data to convert it will STOP at the first trailing whitespace and await the next cin command

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

int main()
{
    int dozens;

    cout << "Enter number of dozen: "
         << endl;
    cin >> dozens;

    cout << dozens << " dozen "
         << " is " << 12*dozens
         << "items." << endl;
    return 0;
}
```

**Main Take-away:**

- cin SKIPS leading whitespace
- cin STOPS on the first trailing whitespace
Timing of Execution

- When execution hits a 'cin' statement it will:
  - Wait for input if nothing is available in the input stream
    - OS will capture what is typed until the next 'Enter' key is hit
    - User can type as little or much as desired until Enter (\n)
  - Immediately extract input from the input stream if some text is available and convert it to the desired type of data

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

int main()
{
  int dozens;
  cout << "Enter number of dozen: "
       << endl;
  cin >> dozens; // input stream empty
          // so wait for input
  cout << 12*dozens << " eggs" << endl;
  double gpa;
  cout << "What is your gpa?" << endl;
  cin >> gpa; // input stream has text
          // so do not wait...
          // just use next text
  cout << "GPA = " << gpa << endl;
  return 0;
}
```
Exercises

• cpp/cin/building_floor
SOLUTIONS
## 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:
  – $25 / 3 = 8$
  – $20 - 12 / 4 * 2 = 14$
  – $3 - 15 \% 7 = 2$
  – $18.0 / 4 = 2.5$
  – $28 - 5 / 2.0 = 25.5$

Exercises

• What is printed by the following two programs?

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

int main()
{
    int value = 1;
    value = (value + 5) * (value - 3);
    cout << value << endl;

double amount = 2.5;
    value = 7;
    amount = value + 6 / amount;
    cout << amount << endl;

    cout << value % 3 << endl;
    return 0;
}
```

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

int main()
{
    int x = 5;
    int y = 3;
    double z = x % y * 6 + x / y;
    cout << z << endl;

    z = 1.0 / 4 * (z - x) + y;
    cout << z << endl;

    return 0;
}
```

-12
9.4
1

13  // or 13.0
5   // or 5.0