Unit 1a – Basic Output and Input (with 'cout' and 'cin')

Mark Redekopp
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
• Use cin statement to get keyboard input from the user
• Predict how cin will treat input with whitespaces and extract data
• Know how variables are declared and assigned
• Trace the operation of assignment statements, expressions, and cin and cout commands
Unit

• **Unit 1: Scalar processing**
  – aka IPO=Input-Process-Output Programs

• **Unit 2: Linear (1D) Processing**

• **Unit 3: Multidimensional Processing**

• **Unit 4: Divide & Conquer (Functional Decomposition)**
UNIT 1: SCALAR PROCESSING
Scalar Processing Programs

Scalar processing programs follow a simple sequence: Prompt-Input-Process-Output (PIPO)

1. Prompt the user for **1 or more** (some constant amount of) input values
2. Receive the input value(s)
3. Using the input, perform operations (processing) to produce **1 or more** (some constant amount of) desired output values
4. Display the output value(s)

Examples of Input-Process-Output programs.
Where is the processing?
PROCESSING
Basic Processing

• To start, we will write programs to do simple processing similar to what we would use a calculator to perform.
• Let us briefly review the constants we introduced you to in an earlier unit.
Review: Constants (aka Literals)

- Integer: 496, 10005, -234
- Double: 12.0, -16., 0.23, 6.02E23, 4e-2
  - Both very large and very small numbers (i.e. fractions/decimals)
- Characters (char type): enclosed in single quotes (')
  - Printing characters: 'a', '5', 'B', '!'  
  - Each quoted value is converted to appropriate ASCII number (e.g. 'a' => 97)
  - Non-printing special characters use "escape" sequences (i.e. preceded by a \):  
    '\n' (newline/enter), '\t' (tab), '\\' (slash), '"' (apostrophe)
- C-Strings (Note: there is also a C++ string type...)
  - 0 or more characters between double quotes ("")  
    "hi1\n", "12345", "b", \tAns. is %d"
  - Ends with a '\0' = 0 (aka NULL character) added as the last byte/character to allow code to delimit the end of the string
- Boolean (C++ only): false, true
  - Physical representation: 0 = false, Non-zero (1, -5, 300) = 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.
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)</td>
</tr>
</tbody>
</table>

\[
10 \% 3 = \boxed{1} \\
17 \% 10 = \boxed{7}
\]
# Precedence

- Order of operations/evaluation of an expression
- Higher level (level 16 in table) done first
- Notice operations with the same level or precedence usually are evaluated left to right

## Evaluate:
- $2 \times -4 - 3 + 5 / 2$

## Tips:
- Use parenthesis to add clarity
- Add a space between literals

\[(2 \times -4) - 3 + (5 / 2)\]
Review: 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

OUTPUT
C++ Output

- The most basic programming command is usually printing something to the screen
- In C++, this is done with a `cout` (short for character output) command
  - Different constants (and later variables) can be specified for output and `cout` will convert it all to text
  - The text is then handed to the OS to be printed
  - `endl` = `
` (newline)

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

int main()
{
    cout << "5 dozen is: " << 5*12 << endl;
    cout << "The end";
    return 0;
}
```
Output From Your Program

• To see 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 similar to '\n'

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

```cpp
#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;

int main()
{
    cout << 102 1889 << endl;  // Compile Error!
    cout << 102 << 1889 << endl; // Better, but no spaces
    cout << 102 << " " << 1889 << endl; // Best
    cout << "102 1889" << endl; // or as a string

    cout << "There are " << 60*24*365 << " minutes in a year."
        << endl;
    return 0;
}
```

1021889
102 1889
102 1889
There are 525600 minutes in a year.

The '<<' operator has multiple (aka "overloaded") meanings. In C (and still in C++) it is used to shift bits in a variable to the left, but C++ also uses it for output. In that (output) context, it is **NOT** known as the shift operator but the "stream insertion" operator!
Challenge 1

• Write a program that incorporates:
  – Processing
  – Output

• Think of simple converters/calculator operations to work with a fixed input
  – Example: How many hours will it take to drive 850 miles at 110 km/h?
VARIABLES AND RECEIVING INPUT WITH CIN
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 via cin**
  - **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
C/C++ Variables

- A variable is a reserved memory location that
  - Stores a value that can be read (retrieved) or written (changed) as often as desired
  - Associates a descriptive name (e.g. x) the programmer will use with that memory location (aka address) and the value stored in that location
- It's like renting an apartment or storage unit
- You must "declare" (allocate) your variables before using/assigning to them
  - A variable is not allocated (aka "in scope") until the computer executes the line with the declaration
  - The declaration must give the type of the variable and a name/identifier

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

int main()
{
    // Sample variable declarations
    char gr = 'A'; // BAD! must declare first
    int x; // uninitialized variables
    int x; // GOOD! Declared 'gr'
    // 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.

Difference: C required that variables be declared at the beginning of a function before any operations. C++ relaxes this and allows declarations anywhere in the code.
Keyboard Input

- In C++, the 'cin' object is responsible for 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 in your program.
- '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;
}
```

The >> operator also has multiple (aka "overloaded") meanings. In C (and still in C++) it is used to shift bits in a variable to the right, but C++ also uses it for input. In that (input) context, it is known not as the shift operator but the "stream extraction" operator!
Dealing With Whitespace

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

- `cin` sequentially **discards leading whitespace** characters until it hits a non-whitespace.

- `cin` then checks the characters can be converted to the appropriate variable type and keeps scanning for more.

- `cin` will **STOP at the first trailing whitespace** (or on a character unable to be converted to the desired type) 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;
}
```

Suppose at the prompt the user types:

```
\t15
```

**Input stream:**

- Leading whitespace is discarded by `cin`.
- The first non-whitespace character is `1`.
- Then `cin` checks if the input can be converted to an integer and keeps scanning for more.

**Main Take-aways:**
- `cin` **SKIPS** leading whitespace.
- `cin` **STOPS** on the first trailing whitespace.

**Space ≠ Whitespace** (Whitespace = ' ', '	', '
', etc.)
Timing of Execution

- When execution reaches a 'cin' statement, it will either:
  - **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** 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;
}
```
Multiple Inputs and Unexpected Inputs

- Use the '>>' operator to separate any number of variables you want to read
- For now let us make the unreasonable assumption that the user always types in the write "format" of information
  - We'll learn more about how cin handles errors later

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

int main()
{
    int score; double multiplier;
    cin >> score >> multiplier;
    cout << "Your new score is " << score * multiplier << endl;
    return 0;
}
```

Input stream:
```
15 2.4
```

Output:
```
Your new score is 36
```
Challenge 2

• Write a program that incorporates all 3 aspects:
  – Input
  – Processing
  – Output

• Example: Compute and output how far an object with initial downward velocity, $v$, has fallen after $t$ seconds?
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:
  - \( \frac{25}{3} = 8 \)
  - \( 20 - 12 / 4 \times 2 = 14 \)
  - \( 3 - 15 \% 7 = 2 \)
  - \( 18.0 / 4 = 2.5 \)
  - \( 28 - 5 / 2.0 = 25.5 \)