Input and Output (and predefined functions)

1E3
Topic 4

Objectives

- Learn what a stream is and examine input and output streams
- Explore how to read data from the standard input device
- Learn how to write data to the standard output device
- Learn to use predefined functions.
- Discover how to use manipulators in a program to format output
This topic is based on Chapter 3 of the textbook.
You should read Chapter 3 now.
- Skip Putback & Peek; Input Failure; The Clear function; and File Input /Output
- These notes don’t cover everything.

What does this pseudocode compute?

Input a number-list.
Set x to 0.
FOR EACH number on number-list
    IF number > x
        THEN set x to number.
Output x.

A. Sum of numbers
B. Max of numbers
C. Min of numbers
D. None of the above
Introduction

- We’ve seen
  - `cin >> n;`
  - Assuming n is declared as an integer, this reads a number from the keyboard and stores it in the variable n.
  - `cout << "The answer is " << sum;`  
  - Writes the string in the “ “ followed by the value of the variable sum to the screen.
- Here we look more closely at this kind of thing.

Input/Output Streams

- **I/O**: sequence of bytes (stream of bytes) from source to destination
- Bytes are usually characters, unless program requires other types of information
- **Stream**: sequence of characters from source to destination
- **Input Stream**: sequence of characters from an input device to the computer
- **Output Stream**: sequence of characters from the computer to an output device
Standard I/O Devices

- Use `iostream` to extract (receive) data from keyboard and send output to the screen
- `iostream` contains definitions of two types
  - `istream` - input stream
  - `ostream` - output stream
- `iostream` has two variables
  - `cin` - stands for common input
  - `cout` - stands for common output

Using `cin` and `cout`

- To use `cin` and `cout`, the preprocessor directive `#include <iostream>` must be used
- The declaration is similar to the following C++ statements:
  ```
  istream cin;
  ostream cout;
  ```
cin and the Extraction Operator $\gg$

- The syntax of an input statement using `cin` and the extraction operator $\gg$ is

  ```cpp
  cin >> variable >> variable...;
  ```

- The extraction operator $\gg$ is binary
- The left-hand operand is an input stream variable such as `cin`
- The right-hand operand is a variable of a `simple` data type

Standard Input

- Every occurrence of $\gg$ extracts the next data item from the input stream
  - `cin >> diam >> price;`
- is equivalent to
  - `cin >> diam; cin >> price;`
- $\gg$ skips all `whitespace`
  - `Whitespace` characters consist of blanks and certain nonprintable characters
Data Type of Input

- >> distinguishes between character 2 and number 2 by the right hand operand of >>
  - If it is of type char, the 2 is treated as character 2
  - If it is of the type int (or double) the 2 is treated as the number 2

TABLE 3-1  Valid Input for a Variable of the Simple Data Type

<table>
<thead>
<tr>
<th>Data Type of a</th>
<th>Valid Input for a</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>One printable character except the blank</td>
</tr>
<tr>
<td>int</td>
<td>An integer, possibly preceded by a + or − sign</td>
</tr>
<tr>
<td>double</td>
<td>A decimal number, possibly preceded by a + or − sign. If the actual data input is an integer, the input is converted to a decimal number with the zero decimal part.</td>
</tr>
</tbody>
</table>

Reading Data

- When reading data into a char variable
  - Extraction operator >> skips leading whitespace, finds and stores only the next character
  - Reading stops after a single character
Reading Data (Continued)

- To read data into an `int` or `double` variable:
  - Extraction operator `>>` skips leading whitespace, reads plus or minus sign (if any), reads the digits (including decimal)
  - Reading stops on whitespace non-digit character
- Study example 3.1 in the text.

Using Predefined Functions

- A function (subprogram): set of instructions
- When activated, it accomplishes a task
- `main` executes when a program is run
- Other functions execute only when called
- C++ includes a wealth of functions
- Predefined functions are organized as a collection of libraries called header files
Predefined Functions

- Header file may contain several functions
- To use a predefined function, you need the name of the appropriate header file
- You also need to know:
  - Function name
  - Number of parameters required
  - Type of each parameter
  - What the function is going to do

Predefined Function Example

- To compute \( x^y \) use \texttt{pow} (power) function.
- \texttt{pow} is in the \texttt{cmath} library
  - As are \texttt{sqrt}, \texttt{sin}, \texttt{cos}, \texttt{log}, \texttt{log10}, ...
- \texttt{pow} takes two numeric parameters
- \texttt{pow} \textbf{returns} a number.
- E.g., \texttt{x = pow(2,3)};
  \texttt{cout} \texttt{<<} \texttt{pow (x, 3)};
```cpp
#include <iostream>
#include <cmath>
#include <string>
using namespace std;

int main()
{
    double u;
    int v;
    cout << "2 to the power of 6 = ";
    cout << pow(2.0, 6) << endl;
    u = 12.5;
    v = 3;
    cout << u << " to the power of " << v;
    cout << " = " << pow(u, v) << endl;
    u = pow(8.0, 2);
    cout << "u = " << u << endl;
    return 0;
}
```

**Example continued**

- The output of the previous program is

  2 to the power of 6 = 64
  12.5 to the power of 3 = 1953.12
  u = 64

  The example shows how to call a function, pass parameters to it and use the value that it returns (if it returns one).
cin and the get Function

- `>>` can’t be used to read blanks or to notice new lines.
  - because `>>` skips blanks and newlines.
- The get function
  - Inputs next character (including whitespace)
  - Stores character at location indicated by its argument
- The syntax of cin and the get function:
  ```cpp
  cin.get(varChar);
  ```

Example

- Suppose I want to read two characters, either of which may be a blank, followed by an integer:
  ```cpp
  cin.get(ch1);
  cin.get(ch2);
  cin >> num;
  ```
- If I type “A 34”,
  - ch1 will be ‘A’, ch2 will be ‘ ‘ and num will be 34.
Other `istream` functions

- Note the sections on `ignore`, `putback` and `peek` for future reference.
- These are functions which allow greater control over the input stream.
- For example
  ```cpp
cin.ignore (100, '.');
```
- Skips up to 100 characters until a full stop is read. The ‘.’ would be discarded too.

Input Failure

- Things can go wrong during execution
- If input data does not match the corresponding variables, the program may run into problems
- Trying to read a letter into an `int` or `double` variable would result in an input failure
- If an error occurs when reading data
  - Input stream enters the fail state
Input Failure (continued)

- Once in a fail state, all further I/O statements using that stream are ignored
- The program continues to execute with whatever values are stored in variables
- This causes incorrect results
- The `clear` function restores input stream to a working state

```
istreamVar.clear();
```

Output

- The syntax of `cout` and `<<` is:

```
cout << expression or manipulator
<< expression or manipulator
<< ...;
```
- Called an output (`cout`) statement
- The `<<` operator is called the insertion operator or the stream insertion operator
- Expression evaluated and its value is printed at the current cursor position on the screen
Output (continued)

- **Manipulator**: alters output
- **endl**: the simplest manipulator
  - Causes cursor to move to beginning of the next line
  - `cout << u << endl << name;`
  - where `u=2034` and name is "Mary" would produce
    
    2034
    Mary

The New Line Character

- Another way of moving to a new line is to include ‘\n’ in an output string.
- ‘\n’ is the new line character.
- Tells the output to go to the next line
  - `cout << "Line 1\nLine 2\n" << name;`
  - where name is "Mary" would produce
    
    Line 1
    Line 2
    Mary
### Formatting Output

- **As well as the `endl` manipulator**
  - `setprecision(n)` outputs decimal numbers with up to `n` decimal places
  - `fixed` outputs floating-point numbers in a fixed decimal format
  - `showpoint` forces output to show the decimal point and trailing zeros

---

<table>
<thead>
<tr>
<th>Escape Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\n</td>
<td>Newline, cursor moves to the beginning of the next line</td>
</tr>
<tr>
<td>\t</td>
<td>Tab, cursor moves to the next tab stop</td>
</tr>
<tr>
<td>\b</td>
<td>Backspace, cursor moves one space to the left</td>
</tr>
<tr>
<td>\r</td>
<td>Return, cursor moves to the beginning of the current line (not the next line)</td>
</tr>
<tr>
<td>&quot;</td>
<td>Backslash, backslash is printed</td>
</tr>
<tr>
<td>'</td>
<td>Single quotation, single quotation mark is printed</td>
</tr>
<tr>
<td>&quot;</td>
<td>Double quotation, double quotation mark is printed</td>
</tr>
</tbody>
</table>
The `setw` Manipulator

- `setw` outputs the value of an expression in specific columns
- If the number of columns exceeds the number of columns required by the expression
  - Output of the expression is right-justified
  - Unused columns to the left are filled with spaces

Types of Manipulators

- Two types of manipulators:
  - With parameters
  - Without parameters
- Parameterized: require `iomanip` header
  - `setprecision`, `setw`, and `setfill`
- Nonparameterized: require `iostream` header
  - `endl`, `fixed`, `showpoint`, `left`, and `flush`
I/O and the string Type

- An input stream variable (cin) and extraction operator >> can read a string into a variable of the data type string
- Extraction operator
  - Skips any leading whitespace characters and reading stops at a whitespace character
  - Can not be used to read strings with blanks
- The function getline
  - Reads until end of the current line
  - Should be used to read strings with blanks

Highlights

- cin and cout are the common input and output streams.
- >> extracts data from an input stream into a variable.
- << pushes data to an output stream
- get and getline allow finer level control
- iomanip library provides manipulators for formatting output.