Input, Output and Strings

Due this week

- Homework 1
  - Submit pdf file on Canvas. PDF
  - Check the due date! No late submissions!!
- Start going through the textbook readings and watch the videos
  - Take Quiz 2.
  - Check the due date! No late submissions!!
- 3-2-1
- Practice Set

Today

- Casts
- Mathematical Functions
- Console input
- Formatted output
- Strings

Casts

- Occasionally, you need to store a value into a variable of a different type, or print it in a different way
- A cast is a conversion from one type (e.g., int) to another type (e.g., double)
- Example: How can we print or capture the exact quotient from two int variables?

```
int x= 25;
int y = 10;
cout << "The quotient is " << x / y;
//gives int quotient of 2; not what we want
```

Casts

- The cast conversion syntax:

```
static_cast<newtype>( data_to_convert)
```

- Example, to get an exact quotient, we cast one of the int variables to a double before dividing:

```
int x= 25;
int y = 10;
cout << x / static_cast<double>(y);
//gives double quotient of 2.5
```

Combining Assignment and Arithmetic

In C++, you can combine arithmetic and assignments. For example, the statement

```
total += cans * CAN_VOLUME;
```

is a shortcut for

```
total = total + cans * CAN_VOLUME;
```

Similarly,

```
total *= 2;
```

is another way of writing

```
total = total * 2;
```

Powers and Roots

In C++, there are no symbols for powers and roots.

To compute them, you must call functions. Don’t forget to include the cmath library

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

Example of `pow()` function call

The `pow()` function has two arguments:

Base

exponent

```
pow(base, exponent)
```

Using the pow function:

```
double balance = b * pow(2, n);
```

Other Mathematical Functions (from `<cmath>`)

Example:

```
double population = 73693997551.0;
double decimal_log = log10(population);
```

Common Error – Unintended Integer Division

If both arguments of `/` are integers, the remainder is discarded:

7 / 3 is 2, not 2.5

but..

7.0 / 4.0, 7 / 4.0, and 7.0 / 4.0 all yield 1.75

Remember: if at least one of the operands is a double, then the result will be a double.

Common Error – Unintended Integer Division

- It is unfortunate that C++ uses the same symbol `/` for both integer and floating-point division.
- It is a common error to use integer division by accident.
  Consider this segment that computes the average of three integers:

```
int score1 = 2
int score2 = 3
int score3 = 5
double average = (score1 + score2 + score3) / 3;
cout << “Your average score is ” << average << endl;
```

Common Error – Unintended Integer Division

Here, however, the `/` denotes integer division because both (score1 + score2 + score3) and 3 are integers.

FIX: make the numerator or denominator into a floating-point number:

```
double total = score1 + score2 + score3;
double average = total / 3;
```

or

```
double average = (score1 + score2 + score3) / 3.0;
```

Common Error – Unbalanced Parentheses

Consider the expression

```
(-(b * b - 4 * a * c) / (2 * a)
```

- What is wrong with it?
  - the parentheses are unbalanced
  - very common with complicated expressions

Spaces in Expressions

It is easier to read

```
x1 = (-b + sqrt(b * b - 4 * a * c)) / (2 * a);
```

than

```
x1=(-b+sqrt(b*b-4*a*c))/(2*a);

Itreallyiseasiertoreadwithspaces!
```

So always use spaces around all operators: `+ - * / % =`

Spaces in Expressions

- Unary minus: A minus sign - used to negate a single quantity like: -b
- Binary minus: A minus sign taking the difference between two quantities: a - b
- We do not put a space after a unary minus. Helps distinguish it from a binary one.
- It is customary not to put a space between a function name and the parentheses.
  Write: `sqrt(x)`
  not `sqrt (x)`

Input and Output

Input

- Sometimes the programmer does not know what value should be stored in a variable – but the user does.
- The programmer must get the input value from the user
  - Users need to be prompted -- how else would they know they need to type something?
  - Prompts are done in output statements
- The keyboard needs to be read from
  - This is done with an input statement

Input with cin >>

The input statement

- To read values from the keyboard, you input them from an object called cin.
- The *double greater than* operator `>>` denotes the *send to* command.

  cin >> bottles;
  is an input statement.

  ```
  Of course, the variable bottles must be defined earlier.
  ```

Input with cin >> to multiple variables

You can read more than one value in a single input statement:

```
cout << "Enter the number of bottles and cans: ";
cin >> bottles >> cans;
```

The user can supply both inputs on the same line:

```
Enter the number of bottles and cans: 2 6
```

Alternatively, the user can press the Enter key or tab key after each input, as cin treats all blank spaces the same

Formatted Output

- When you print an amount in dollars and cents, you want it to be rounded to two significant digits.
- You learned earlier how to round off and store a value but, for output, we want to round off only for display.
- A **manipulator** is something that is sent to `cout` to specify how values should be formatted.
- To use manipulators, you must include the `iomanip` header in your program:

  #include <iomanip>

and of course `using namespace std;` is also needed

Formatted Output for Dollars and Cents: setprecision()

Which do you think the user prefers to see on her gas bill?

```
Price per liter: $1.22
```

or

```
Price per liter: $1.21997
```

Formatted Output Examples

By default, a number is printed with 6 significant digits.

```
cout << 12.345678;
```

outputs

```
12.3457
```

Formatted Output Examples 2

The fixed and setprecision manipulators control the number of digits after the decimal point.

```
cout << fixed << setprecision(2) << 12.3;
```

outputs

```
12.30
```

Formatted Output Examples 3

Four spaces are printed before the number, for a total width of 6 characters.

```
cout << ":" << setw(6) << 12;
```

outputs

```
:    12
```

Formatted Output Examples 4

If the width not sufficient, it is ignored.

```
cout << ":" << setw(2) << 123;
```

outputs

```
:123
```

The width only refers to the next item. Here, the : is preceded by five spaces.

```
cout << setw(6) << ":" << 12;
```

outputs

```
     :12
```

Formatted Output, Dollars and Cents

You can combine manipulators and values to be displayed into a single statement:

```
price_per_liter = 1.21997;
cout << fixed << setprecision(2)
    << "Price per liter: $"
    << price_per_liter << endl;
```

This code produces this output:

```
Price per liter: $1.22
```

Formatted Output with `setw()` to Align Columns

- Use the `setw` manipulator to set the width of the next output field.
- The width is the total number of characters, including digits, the decimal point, and spaces.
- If you want aligned columns of certain widths, use the `setw()` manipulator.
- For example, if you want a number to be printed, right justified, in a column that is eight characters wide, you use

  << setw(8)

  before EVERY COLUMN's DATA.

Exercise: Formatting Examples

Given

```
int quantity = 10;  double price = 19.95;
```

What do the following statements print?

```
cout << "Quantity:" << setw(4) << quantity;
cout << "Price:" << fixed << setw(8) << setprecision(2) << price;
cout << "Price:" << fixed << setprecision(2) << price;
cout << fixed << setprecision(3) << price;
cout << fixed << setprecision(1) << price;
```

Formatted Output, Another Example

```
price_per_ounce_1 = 10.2372;
price_per_ounce_2 = 117.2;
price_per_ounce_3 = 6.9923435;
cout << setprecision(2);
cout << setw(8) << price_per_ounce_1;
cout << setw(8) << price_per_ounce_2;
cout << setw(8) << price_per_ounce_3;
cout << "--------" << endl;
```

produces this output:

```
   10.24
   117.20
     6.99
--------
```

`setprecision` versus `setw:Persistence`

There is a notable difference between the setprecision and setw manipulators.

Once you set the precision, that precision is used for all floating-point numbers until the next time you set the precision.

But `setw` affects only the next value.

Subsequent values are formatted without added spaces.

Strings

Strings

- Strings are sequences of characters:

```
"Hello World!"
```

- Include the string header, so you can create variables to hold strings:

```
#include <iostream>
#include <string>
using namespace std;
...
string name = "Harry";       // literal string "Harry" stored
```

String Initializations

- String variables are automatically initialized to the empty string if you don't initialize them:

```
        string response;      // literal string "" stored
```

- `""` is called the empty or null string

Concatenation of Strings

- Use the `+` operator to concatenate strings;
  that is, put them together to yield a longer string.

```
string fname = "Harry";
string lname = "Potter";

string name = fname + lname; //need a space!
cout << name << endl;

name = fname + " " + lname; //got a space
cout << name << endl;
```

- The output will be:

```
HarryPotter
Harry Potter
```

Common Error – Concatenation of literal strings

```
string greeting = "Hello, " + " World!";
        // will not compile
```

Literal strings cannot be concatenated. And it's pointless anyway, just do:

```
string greeting = "Hello World!";
```

String Input

You can read a string from the console:

```
cout << "Please enter your name: ";
string name;
cin >> name;
```

When a string is read with the `>>` operator, only one word is placed into the string variable.

For example, suppose the user types

`Harry Potter`

as the response to the prompt. Only the string `"Harry"` is placed into the variable name.

String Input

You can use another input string to read the second word:

```
        cout << "Please enter your name: ";
        string fname, lname;
        cin >> fname >> lname;

        //fname gets Harry, lname gets Potter
```

String Input

`getline()` function allows us to accepts a full string input

```
        cout << "Please enter your name: ";
        string name;
        getline(cin, name);

        //name gets Harry Potter
```

String Functions

The `length` *member function* yields the number of characters in a string.

Unlike the `sqrt` or `pow` function, the `length()` function is invoked with the dot notation:

```
string name = “Harry”;
int n = name.length();
```

String Data Representation & Character Positions

```
H  e  l  l  o  ,     W  o  r  l  d  !
0  1  2  3  4  5  6  7  8  9  10 11 12
```

- In most computer languages, the starting position 0 means “start at the beginning.”
- The first position in a string is labeled 0, the second 1, and so on. And don’t forget to count the space character after the comma—but the quotation marks are not stored.
- The position number of the last character is always one less than the length of the string.

`substr` Function

Once you have a string, you can extract substrings by using the `substr` *member function*.

`s.substr(start, length)` returns a *string* that is made from the characters in the `string s`, starting at character *start*, and containing *length* characters. (*start* and *length* are integers)

NOTE: the first character has an index of 0, not 1.

```
string greeting = "Hello, World!";
string sub = greeting.substr(0, 2);
    // sub contains "He"
```

Another Example of the substr Function

```
string greeting = "Hello, World!";
string w = greeting.substr(7, 5);  // w contains "World" (not the !)
```

- `"World"` is 5 characters long but...
- Why is 7 the position of the `"W"` in `"World"`?
- Why is the `"W"` not at position 8?
- *Because the first character has an index of 0, not 1.*

String Character Positions

```
 H  e  l  l  o  ,     W  o  r  l  d  !
 0  1  2  3  4  5  6  7  8  9  10 11 12
```

```
string greeting = "Hello, World!";
string w = greeting.substr(7);
// w contains "World!"
```

If you do not specify how many characters should go into the substring, the call to the `substr()` function will return a substring that starts at the *specified index*, and goes until the *end* of the string

String Functions, Complete Program Example

ch02/initials.cpp

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
   cout << "Enter your first name: ";
   string first;
   cin >> first;
   cout << "Enter your significant other's first name: ";
   string second;
   cin >> second;
   string initials = first.substr(0, 1) + "&" + second.substr(0, 1);
   cout << initials << endl;
   return 0;
}
```

Representing Characters: Unicode. ASCII

- Printable characters in a string are stored as bits in a computer, just like int and double variables
- The bit patterns are standardized:
  - ASCII (American Standard Code for Information Interchange) is 7 bits long, specifying 27 = 128 codes:
    - 26 uppercase letters A through Z
    - 26 lowercase letters a through z
    - 10 digits
    - 32 typographical symbols such as +, -, ', ...
    - 34 control characters such as space, newline
    - 32 others for controlling printers and other devices.