Chapter 5 | Simple I/O

CS185

Overview of Formatted Input/Output

Basics:

- The standard way of displaying output in C++ is through `cout`.
- `cout` is one of the most complex things in C++.
- `cout` is not part of the language. It is part of the C++ standard library (in the **`std`** namespace).
- `cout` is not a function. It is an *object*. (For now, think of it as kind of a **`struct`**.)
- As a **`struct`**, `cout` has many members that you can access.
  - Some of the members are variables of built-in data types (**`int`**, **`bool`**, etc.)
  - Some members are functions (or more exact, *pointers to functions*)

To use `cout`, you must include the appropriate header file:

```
#include <iostream> // No .h extension
```

The `iostream` header actually contains definitions for two types: `istream` and `ostream`.

- `istream` is an input stream (for reading input)
- `ostream` is an output stream (for writing output)
- `cout` is of type `ostream`, since it's purpose is for outputting.
- The standard input `object is cin`, and it's used for reading input.

The stream objects use special operators to perform their "magic". The `ostream` object uses the *insertion operator* or casually called the *output operator*:

Code	`std::cout << 15;    // output the integer 15`
	`std::cout << 3.14;  // output the double 3.14`
	`std::cout << "foo"; // output the NULL terminated string foo`

Output	`153.14foo`

Broken down into its parts (tokens):

namespace	scope-resolution-operator	object	insertion-operator	value	statement-terminator
std	::	cout	<<	15	;

You can chain the operations together in a single statement:

```
// All in one statement
std::cout << 15 << 3.14 << "foo";
```

Obvious questions

- How does `cout` know what type to print?
- What happens if `cout` doesn't know how to print a certain type? (Maybe a user defined type like **`struct TIME`**). For example:

```
struct TIME
{
  int hours;
  int minutes;
  int seconds;
};
```

```
struct TIME t1 = {9, 45, 30};

printf("%XXX", t1); // What format specifier is XXX? (Hint: There is none.)
std::cout << t1;    // Does this work? What will be printed?
```

Changing The Default Precision

Use the *precision* member to change the number of significant digits displayed for floating-point values.

```
cout.precision(value);
```

All of the subsequent floating-point values will be displayed to *value* significant digits.

Example:

Code

```
#include <iostream> // cout, endl

int main()
{
	float f = 123.4567F;
	double d = 3.14159265358979323846264338327795;

	// Modified precision (3, 6, 9, 12, etc.)
	for (int i = 3; i <= 21; i += 3)
	{
		std::cout << "precision is " << i << std::endl;
		std::cout.precision(i);
		std::cout << "f is |" << f << "|" << std::endl;
		std::cout << "d is |" << d << "|" << std::endl;
		std::cout << std::endl;
	}

	return 0;
}
```

Output

```
precision is 3
f is |123|
d is |3.14|

precision is 6
f is |123.457|
d is |3.14159|
```

```
precision is 9
f is |123.456703|
d is |3.14159265|

precision is 12
f is |123.456703186|
d is |3.14159265359|

precision is 15
f is |123.456703186035|
d is |3.14159265358979|

precision is 18
f is |123.456703186035156|
d is |3.14159265358979312|

precision is 21
f is |123.45670318603515625|
d is |3.141592653589793116|
```

Changing the Field Width

Use the **width** member of the `cout` object to modify the width:

```
cout.width(value);
```

The next value displayed will require at least *value* characters in the output. (Like `printf`, if the size of *value* is smaller than the number of characters required to display the value, the output will not be truncated.

Unlike *precision*, the new width specification will only be applied to the next output operation before being reset. (Meaning that it will only apply to one << operation.)

Default right justified:

Code

```
#include <iostream> // cout, endl

int main()
{
	int i = 42;
	float f = 1.23456789F;
	double d = 3.14159265358979323846433832795;

	std::cout << "i is |";
	std::cout.width(12);
	std::cout << i;
	std::cout << "|" << std::endl;

	std::cout << "f is |";
	std::cout.width(12);
	std::cout << f;
	std::cout << "|" << std::endl;
```

```
        std::cout << "d is |";
        std::cout.width(12);
        std::cout << d;
        std::cout << "|" << std::endl;

        system("pause");
        return 0;
}
```

Output

```
i is |           42|
f is |      1.23457|
d is |      3.14159|
```

Another example:

Code

```
#include <iostream> // cout, endl

int main()
{
        // Set width to 10
        std::cout << "|";
        std::cout.width(10);
        std::cout << "setw(10)";
        std::cout << "|" << std::endl;

        // Set width to 15
        std::cout << "|";
        std::cout.width(15);
        std::cout << "setw(15)";
        std::cout << "|" << std::endl;

        // Set width to 20
        std::cout << "|";
        std::cout.width(20);
        std::cout << "setw(20)";
        std::cout << "|" << std::endl;

        system("pause");
        return 0;
}
```

Output

```
|  setw(10)|
|       setw(15)|
|            setw(20)|
```

Padding With Characters Other Than A Space

Use the *fill* member of the `cout` object to modify the fill character:

```
cout.fill(fill-char);
```

Any padding that is added will use *fill-char* instead of the default space.

Executing this once before using `cout`:

```
std::cout.fill('*');
```

This will output something different:

Code

```
#include <iostream> // cout, endl

int main()
{
	//Padding with stars
	std::cout.fill('*');

	// Set width to 10
	std::cout << "|";
	std::cout.width(10);
	std::cout << "setw(10)";
	std::cout << "|" << std::endl;

	// Set width to 15
	std::cout << "|";
	std::cout.width(15);
	std::cout << "setw(15)";
	std::cout << "|" << std::endl;

	// Set width to 20
	std::cout << "|";
	std::cout.width(20);
	std::cout << "setw(20)";
	std::cout << "|" << std::endl;

	system("pause");
	return 0;
}
```

Output

```
|**setw(10)|
|*******setw(15)|
|************setw(20)|
```

Using `setf` For More Control

The *setf* member controls many aspects of the format.

Code

```
#include <iostream> // cout, endl

int main()
{
	double cost = 22.5; // $22.50
	bool flag = true;

	// Default formatting for floating point: cost is 22.5
	std::cout << "cost is " << cost << std::endl;

	// Show trailing zeros: cost is 22.5000
	std::cout.setf(std::ios_base::showpoint);
	std::cout << "cost is " << cost << std::endl;

	// Only 4 significant digits:
	// cost is 22.50 (trailing zeros still in effect)
	std::cout.precision(4);
	std::cout << "cost is " << cost << std::endl;

	// Fixed-point, 2 digits to the right: cost is 22.50
	std::cout.setf(std::ios_base::fixed, std::ios::floatfield);
	std::cout.precision(2);
	std::cout << "cost is " << cost << std::endl;

	// Default formatting for boolean: flag is 1
	std::cout << "flag is " << flag << std::endl;

	// Display true/false instead of 1/0: flag is true
	std::cout.setf(std::ios_base::boolalpha);
	std::cout << "flag is " << flag << std::endl;

	system("pause");
	return 0;
}
```

Output

```
cost is 22.5
cost is 22.5000
cost is 22.50
cost is 22.50
flag is 1
flag is true
```

Changing justification

Code

```
#include <iostream> // cout, endl

int main()
{
   int i = 42;

   std::cout << "i is |";
   std::cout.width(12);
   std::cout.setf(std::ios_base::left, std::ios_base::adjustfield);
   std::cout << i;
   std::cout << "|" << std::endl;

   system("pause");
   return 0;
}
```

Output

```
i is |42          |
```

Comments

- What's the meaning of: `std::ios_base::showpoint`?
- What is the signature of the `setf` function?
  1. One parameter: `std::cout.setf(std::ios_base::showpoint);`
  2. Two parameters: `std::cout.setf(std::ios_base::left, std::ios_base::adjustfield);`
- More information on setf and flags.
- There is also a function called unset which is kind of the opposite of setf:

```
std::unset(ios_base::boolalpha);
```

The above would turn off the *boolalpha* flag that was set by *setf*.

Manipulators

Manipulators can be a more convenient way of formatting output. To use them, you must include another file:

```
#include <iomanip> // No .h extension
```

They work much the way some of the members work, including `setf`. Examples:

Code

```
#include <iostream> // cout, endl
#include <iomanip> // setprecision, setw

int main()
{
	float f = 1.23456789F;
	double d = 3.14159265358979323826433832795;

	std::cout << std::setprecision(3) << "f is |";
	std::cout << std::setw(6) << f << "|\n";
	std::cout << std::setprecision(5) << "d is |";
	std::cout << std::setw(8) << d << "|\n";

	system("pause");
	return 0;
}
```

Output

```
f is |  1.23|
d is |  3.1416|
```

The ***cout*** statements can be written in different ways:

```
std::cout << std::setprecision(3) << "f is |"
          << std::setw(6) << f << "|\n"
          << std::setprecision(5) << "d is |"
          << std::setw(8) << d << "|\n";
```

More examples:

Code

```
std::cout << std::setprecision(5)
                  << "d is |"
                     << std::setfill('-')
                     << std::left
                     << std::fixed
                     << std::setw(10)
                     << d
                     << "|\n";
```

Output

```
d is |3.14159---|
```

More information on manipulators.

Overview of Input

- The standard way of reading input in C++ is through ***cin***.
- Also like ***cout***, ***cin*** is not part of the language. It is part of the C++ standard library (in the **std** namespace).
- Like ***cout***, ***cin*** is an object.
- ***cin*** is of type ***istream***, since it's purpose is for input.

To use `cin`, you must include the appropriate header file (same one used for `cout`)

```
#include <iostream> // No .h extension
```

Example:

Code

```
#include <iostream> // cout, endl

int main()
{
	int i;
	float f;
	double d;
	char s[10];

	// Prompt user and read input
	std::cout << "Enter an int: ";
	std::cin >> i;

	std::cout << "Enter a float: ";
	std::cin >> f;

	std::cout << "Enter a double: ";
	std::cin >> d;

	std::cout << "Enter a string: ";
	std::cin >> s;

	// Display the input
	std::cout << i << std::endl;
	std::cout << f << std::endl;
	std::cout << d << std::endl;
	std::cout << s << std::endl;

	system("pause");
	return 0;
}
```

Sample Run 1

```
Enter an int: 45
Enter a float: 3.14
Enter a double: 3.188888889
Enter a string: digipen
45
3.14
3.18889
digipen
```

Sample Run 2	`Enter an int: 10`
	`Enter a float: 23.45`
	`Enter a double: .8798`
	`Enter a string: Supercalifragilistic`
	`10`
	`23.45`
	`0.8798`
	`Supercalifragilistic`
	
	PS: The output is actually undefined and the application might crash since we exceeded the string's size.

Sample Run 3	`Enter an int: 10`
	`Enter a float: 23.45`
	`Enter a double: .87987898`
	`Enter a string: Supercalifragilisticexpilalidocious`
	`10`
	`23.45`
	`6.78872e+199`
	`Supercalifragilisticexpilalidocious`
	
	PS: The output is actually undefined and the application might crash since we exceeded the string's size.

Note:

- ***cin*** can interpret the input based on the type.
- There is no protection when reading strings into character arrays.
- Like **cout**, you can chain all of the input into one statement:

```
// Prompt user and read input
std::cout << "Enter an int, float, double, string: ";
std::cin >> i >> f >> d >> s;
```