Character Input with `cin` and `cin.getline` and Character Output with `cout`

Defensive Programming is also covered

---

The most common way for a C++ program to input from a keyboard and output to the display is to use `cin` and `cout` which stand for console-in and console-out.

```cpp
int a;
cout << "Enter a number: ";
cin >> a;
cout << "The number squared is " << a*a << endl;
```

Use `endl` in C++ to move the cursor to the next line on the display. The word `endl` means 'end line'. The last character of `endl` is a small-L not the number-1.

**NOTE:** Operating systems like Linux can use pipes and redirection to cause stdin and stdout to use other devices or even other programs.
cin and cout

**cin** stands for Console-Input
**cout** stands for Console-Output

They are pronounced as SEE-in and SEE-out. You could also say console-in and console-out, but never SIN or KOUT, otherwise people will think that you are an uninformed newbie 😖 and be accused of a grievous sin 🧑‍⚕️.

---

**cin and cout** Know the Data Type

```cpp
int a;       // the variable **a** holds an integer
            // it can only hold whole numbers
double b;   // the variable **b** holds a floating-point
            // value with digits past the decimal
char c;     // the variable **c** holds only a single
            // character
string d;   // the variable **d** holds a string of
            // string of multiple characters
```

User defined data types can also be used with cin and cout by implementing a **Friend** function. This will be covered much later.
The INSERTION Operators <<

C++ uses the insertion operators >> and << with cin and cout. Look at the way the arrows are facing with respect to the console and the variable.

```cpp
int x;
cin >> x;  // read the keyboard, put the value into x
cout << x;  // get the value from x, send to the display
```

using namespace std;

It is not required to place using namespace std; near the top of your program. The other option is to identify the namespace before each cin or cout like this:

```cpp
std::cout << "Enter a number ";
std::cin >> x;
```

The double-colon :: is called the scope resolution operator.
This part of the discussion covers C++ Character Input with `cin` and `cin.getline`
Defensive Programming is also covered

---

**How Does **cin** work?**

- **cin** reads a stream of data from the keyboard into a variable
- A sample program demonstrates how **cin** can read three numbers from the keyboard, add them together and display the sum
- **cin** uses Whitespace characters to separate one piece of data from the next piece
Reading one or more pieces of data

`cin` can read one or more pieces of data at a time.

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

int main()
{
    int empID;
    string name;
    cout << "Enter ID# and last name: ";
    cin >> empID;
    cin >> name;
    cout << "Hello " << name << " " << empID;  
    cout << endl;
    return 0;
}
```

Separate `cin` statements are used to read into `empID` and `name`.

One `cin` statement is used to read into `empID` and `name`.

---

Sample Program – Add 3 Numbers

```cpp
#include "stdafx.h"
#include <iostream>
using namespace std;

int main(int argc, char* argv[])
{
    // declare the variables
    int a;
    int b;
    int c;
    int sum;
    
    // input
    cout << "Enter three numbers: ";
    cin >> a >> b >> c;
    // process
    sum = a + b + c;
    // output
    cout << "The sum is " << sum << endl;
    return 0;
}
```

Same result if the numbers were entered on the same line or on different lines.
Whitespace

```cpp
// declare the variables
int a, b, c, sum;

// input
cout << "Enter three numbers: ";
cin >> a >> b >> c;
```

The PROMPT Message

When reading data from the keyboard, it is necessary to display a message on the screen asking for the data and identifying what type of data is expected. This is called a prompt. If no prompt is provided, the cursor on the screen will just blink and the user will have no idea of what to do or may think that the program just crashed.
The PROMPT message
The prompt is part of the input process.

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

const double PERCENT_RAISE = 4.75;

int main()
{
    double currentPay;
    double newPay;
    // input the current pay
    cout << "Enter current pay: ";
    cin >> currentPay;
    // processing
    newPay = currentPay + currentPay*PERCENT_RAISE/100;
    // output
    cout << "The new pay is: " << newPay << endl;
    return 0;
}
```

Unexpected Inputs and Defensive Programming

- What happens if `cin` is expecting one data type and something else is input?
- How do we find out what `cin` is actually reading?
- How do we detect an error from `cin` and what should be done if an error occurs?
### Unexpected Data

```cpp
#include "stdafx.h"
#include <iostream>
using namespace std;

int main(int argc, char* argv[]) {
    // declare the variables
    int a;
    int b;
    int c;
    int sum;

    // input
    cout << "Enter three numbers: ";
    cin >> a >> b >> c;

    // process
    sum = a + b + c;
    // output
    cout << "The sum is " << sum << endl;
    return 0;
}
```

*Where did this come from???

**cin** is expecting to input integers

**cin** fails with wrong data

```cpp
// declare the variables
int a, b, c, sum;

// input
cout << "Enter three numbers: ";
cin >> a >> b >> c;

READ INTO a
Ignore leading space, read the 5, stop at the space and put 5 into a leave the space in the input buffer

5 6 . 2 \n
READ INTO b
Ignore leading space, read the 6, stop at the period and put 6 into b leave the period in the input buffer

READ INTO c
**cin** is expecting to read into an integer and sees the period that was left in the input buffer. **cin** can't put anything into c. Whatever garbage was in memory is still in c
```
a) Use code to display the values

```c++
int main(int argc, char* argv[]) {
    // declare the variables
    int a, b, c;
    // input
    cin >> a >> b >> c;
    // output
    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    cout << "c = " << c << endl;
    return 0;
}
```

`cin` read the 5 into the variable `a`, then `cin` read the 6 into the variable `b`, then when `cin` went to read into the variable `c` it saw the decimal point. Integers are whole numbers. No decimals allowed. Since the variable `c` was not initialized, whatever garbage was in its memory location is what was used.

b) Use Debug to display the values

When using Microsoft Visual Studio:
1) Click in the gray bar on the left to set a breakpoint
2) Use Debug/Start (F5) to run the program
3) Enter the numbers. The program will pause at the breakpoint
4) Hover the mouse over each of the variables to display their values
Possible Solutions

Solution #1 – Initialize the variables to 0 to prevent weird numbers from showing, but this does not stop wrong answers from being displayed. **BAD Solution**

Solution #2 – change the definition of the variables from type `int` to type `double`. This will allow the 6.2 to be read without an error, but the program will still fail if the user inputs a non-numeric character such as `X`. This is NOT a complete solution. **Poor Solution**

Solution #3 – Use `cin.fail()` to determine if cin was not able to input all the data that was entered. **GOOD**

Solution #4 – Use a `Try...Catch` block to detect and process an input error. **GOOD**

Solution #1 - Initialize the Variables

```cpp
#include "stdafx.h"
#include <iostream>
using namespace std;

int main(int argc, char* argv[]) {
    // declare the variables
    int a = 0;
    int b = 0;
    int c = 0;
    int sum = 0;

    // input
    cout << "Enter three numbers: 
";
    cin >> a >> b >> c;
    // process
    sum = a + b + c;
    // output
    cout << "The sum is " << sum << endl;
    return 0;
}
```

The program ran and produced an answer, but the answer is **WRONG**! It only added 5 + 6 because `c` was not read. `cin` stopped trying to read `c` when it saw the decimal point.

`cin` is expecting to input integers

Not a good solution
Solution #2 – **double** Data Type

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

int main(int argc, char* argv[]) {
    // declare the vars
    double a;
    double b;
    double c;
    double sum;

    // input
    cin >> a >> b >> c;
    // process
    sum = a + b + c;
    // output
    cout << "The sum is " << sum << endl;
    return 0;
}
```

- `cin` is expecting to input floating point numbers.
- It works! All input was numeric.
- Not a good solution.

Solution #3 – `cin.fail()` detects an error

```cpp
// input
cout << "Enter three numbers: ";
cin >> a >> b >> c;
// process
if (cin.fail())
{
    cout << "Error reading data" << endl;
    return 1;
}
else
{
    sum = a + b + c;
    // output
    cout << "The sum is " << sum << endl;
}
return 0;
```

- The error was detected and it was properly processed by the program.
- This is a good solution.
Solution #4 – Use a **Try...Catch** block

```cpp
try
{
    // input
    cin >> a >> b >> c;
    // process
    sum = a + b + c;
    // output
    cout << "The sum is " << sum << endl;
} catch (int errID)
{
    cout << "Error reading data" << endl;
    return 1;
} return 0;
```

This is a good solution

The input, normal processing and output is placed in a *try* block. Any errors are processed in the *catch* block.

**cin.getline(...)**

Use **cin.getline(...)** to read a full line of text without stopping each time some whitespace is detected.
cin stops at whitespace

**cin.getline** does not

Anything after a space is lost by cin. Many last names have spaces. Sometimes McElroy has a space and becomes Mc Elroy so letters get addressed to Mr. Elroy instead of Mr. Mc Elroy.

By using **cin.getline** all of the characters up to the Enter key are read into the character array **fullName**. **cin.getline** needs to know the size of the array.
cout

`cout` can output one or more pieces of data to the display console. Each piece of data is separated by the `<<` insertion operator. The `endl` is used to move the cursor to the next line on the screen. NOTE: the last character in `endl` is a small-L, not the number-1. `endl` stands for END-Line.
### cout Example – Display 4 Things

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

int main()
{
    char name[] = "Fred Flintstone";
    double pay = 425.73;

    cout << name << " your pay is \$" << pay << endl;
    return 0;
}
```

1. Provide a space between the name and the paycheck
2. Move the cursor to the next line

### cout Example – Prompt Message

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

int main()
{
    int score;
    const int PASSING_SCORE = 70;

    cout << "Enter your score: ";
    cin >> score;
    if (score >= PASSING_SCORE)
        cout << "Good job" << endl;
    else
        cout << "Try again" << endl;
    return 0;
}
```

1. A space is provided before the closing quote " so that the user won't be typing right next to the prompt message.
2. The endl was not included on this line so that the user's input would be on the same line as the prompt message.
Formatted Output with `cout`

Use the methods that are provided by `iomanip` to format output using `cout`. Depending on the data type, the following things can be done:

- Set field width
- Set fill character
- Left-justify
- Right-justify
- Center
- Set base
- Set number of digits past the decimal

```
#include <iomanip>
```

### Field modifiers

<table>
<thead>
<tr>
<th>Field modifiers</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set the width of the field</td>
<td><code>&lt;&lt; setw(10)</code></td>
</tr>
<tr>
<td>Left justify</td>
<td><code>&lt;&lt; left</code></td>
</tr>
<tr>
<td>Right justify (default)</td>
<td><code>&lt;&lt; right</code></td>
</tr>
<tr>
<td>Fill characters</td>
<td><code>&lt;&lt; setfill('*')</code></td>
</tr>
</tbody>
</table>

### Display integers

<table>
<thead>
<tr>
<th>Display integers</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display as decimal (default)</td>
<td><code>&lt;&lt; dec</code></td>
</tr>
<tr>
<td>Display as octal</td>
<td><code>&lt;&lt; oct</code></td>
</tr>
<tr>
<td>Display as hex</td>
<td><code>&lt;&lt; hex</code></td>
</tr>
<tr>
<td>Set to any base</td>
<td><code>&lt;&lt; setbase(8)</code></td>
</tr>
<tr>
<td>Display A-F instead of a-f</td>
<td><code>&lt;&lt; uppercase</code></td>
</tr>
</tbody>
</table>

### Display float numbers

<table>
<thead>
<tr>
<th>Display float numbers</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of digits past decimal</td>
<td><code>&lt;&lt; setprecision(2)</code></td>
</tr>
<tr>
<td>Fixed point notation (default)</td>
<td><code>&lt;&lt; fixed</code></td>
</tr>
<tr>
<td>Scientific notation</td>
<td><code>&lt;&lt; scientific</code></td>
</tr>
<tr>
<td>Show the decimal point even if the digits past the decimal are all zeros</td>
<td><code>&lt;&lt; showpoint</code></td>
</tr>
</tbody>
</table>
cout Formatting Example

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

int main()
{
    int ID = 2873;
    double pay = 475.739;
    char name[] = "Fred Flintstone";

    cout << setw(5) << ID << " " // space before name
         << setw(20) << left << name
         << setw(10) << setfill('*') << fixed << right
         << setprecision(2) << showpoint << '$' << pay
         << endl << endl;
}
```

#include <iomanip> is required to do formatting with cout

There are multiple lines in this cout statement. The cout statement does not end until the semicolon ; is reached

### cout Formatting Example

The ID is 5 columns wide and right-justified. The name is 20 columns wide and left-justified.

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

double pay = 475.739;
char name[] = "Fred Flintstone";

cout << setw(5) << ID << " " // space before name
     << setw(20) << left << name
     << setw(10) << setfill('*') << fixed << right
     << setprecision(2) << showpoint << pay
     << endl << endl;
}
```
cout Formatting Example

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

int main() {
    cout << setw(5) << ID << " " // space before name
        << setw(20) << left << name
        << setw(10) << setfill('*') << fixed << right
        << setprecision(2) << showpoint << '$' << pay
        << endl << endl;
    return 0;
}
```

The pay is 10 columns wide, with 2 digits past the decimal and right-justified. Asterisks * fill any unused spaces. A dollar-sign $ is placed before pay is displayed.

Creating a blank line - Example

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

int main() {
    char name[] = "Fred Flintstone";
    cout << setw(5) << ID << " " // space before name
        << setw(20) << left << name
        << setw(10) << setfill('*') << fixed << right
        << setprecision(2) << showpoint << '$' << pay
        << endl << endl;
    return 0;
}
```

The first endl moves the cursor off the line with the ID, name and pay to the next line. The second endl creates a blank line.