

0. Annotated slides from Wednesday

CS319: Scientific Computing

I/O, flow, loops, and functions

Dr Niall Madden

Week 3: 29 and 31 January,
2026



Source: xkcd (1411)

Slides and examples: <https://www.niallmadden.ie/2526-CS319>

0. Inputs for this weeks classes:

- 1 Preview of Lab 1
- 2 Recall from Week 2
- 3 Basic Output
- 4 Output Manipulators
- 5 Input
- 6 Flow of control – if-blocks
- 7 Loops
- 8 Functions

Slides and examples:
niallmadden.ie/2526-CS319



1. Preview of Lab 1

1. Labs start this week.
2. Attend (at least) one hour Thurs 9-10 or Friday 12-1 in AdB-G021.
3. Lab 1 is concerned with program structure, conditionals, and loops. And a little about numbers in C++
4. Submit your C++ file as it is on Friday. This is just to test that you upload the correct file, and to verify participation. So long as you upload a C++ file, you'll get the mark.

2. Recall from Week 2

In Week 2 we studied how numbers are represented in C++.

We learned that all are represented in binary, and that, for example,

- ▶ An `int` is a whole number, stored in 32 bytes. It is in the range $-2,147,483,648$ to $2,147,483,647$.
- ▶ A `float` is a number with a fractional part, and is also stored in 32 bits.
A positive `float` is in the range 1.1755×10^{-38} to 3.4028×10^{38} .
Its **machine epsilon** is $2^{-23} \approx 1.192 \times 10^{-7}$.
- ▶ A `double` is also number with a fractional part, but is stored in 64 bits.
A positive `double` is in the range 2.2251×10^{-308} to 31.7977×10^{308} .
Its **machine epsilon** is $2^{-53} \approx 1.1102 \times 10^{-16}$.

3. Basic Output

Last week we had this example: *To output a line of text in C++:*

```
#include <iostream>
int main() {
    std::cout << "Howya World.\n";
    return(0);
}
```

'\n' ~ new line.

Better: std::cout << "x=" << x ;
std::cout << "x=" << x << "y=" << y;

- ▶ the identifier **cout** is the name of the **Standard Output Stream** – usually the terminal window. In the programme above, it is prefixed by **std::** because it belongs to the *standard namespace*...
- ▶ The operator **<<** is the **put to** operator and sends the text to the *Standard Output Stream*.
- ▶ As we will see **<<** can be used on several times on one lines.
E.g.

```
std::cout << "Howya World." << "\n";
```

4. Output Manipulators

endl

As well as passing variable names and string literals to the output stream, we can also pass **manipulators** to change how the output is displayed.

For example, we can use `std::endl` to print a new line at the end of some output.

In the following example, we'll display some Fibonacci numbers. These are defined by the recurrence: $f_0 = 1$, $f_1 = 1$, and, for $i > 1$, $f_i = f_{i-1} + f_{i-2}$.

We'll use the `for` construct, which will be explained later in this class.

4. Output Manipulators

endl

01Manipulators.cpp

```
4 #include <iostream>          f0 = 1      f1 = 1
5 #include <string>           f2 = f0+f1 = 2
6 #include <iomanip>          f3 = f2+f1 = 3      f4 = f3+f2 = 5
7 int main()                  f5 = f4+f3 = 8
8 {                           ...
9     int i, fib[16];          ...
10    fib[0]=1; fib[1]=1;       ...
11
12   std::cout << "Without setw manipulator" << std::endl;
13   for (i=0; i<=12; i++)
14   {
15     if( i >= 2)
16       fib[i] = fib[i-1] + fib[i-2];
17     std::cout << "The " << i << "th " <<
18     "Fibonacci Number is " << fib[i] << std::endl;
19 }
```

- ▶ `std::setw(n)` sets the width of a field to `n`. Useful for tabulating data.

01Manipulators.cpp

```
22 std::cout << "With the setw  manipulator" << std::endl;
23 for (i=0; i<=12; i++)
24 {
25     if( i >= 2)
26         fib[i] = fib[i-1] + fib[i-2];
27     std::cout
28         << "The " << std::setw(2) << i << "th "
29         << "Fibonacci Number is "
30         << std::setw(3) <<    fib[i] << std::endl;
```

4. Output Manipulators

setw

Other useful manipulators:

- ▶ `setfill` → the character to use when filling.
- ▶ `setprecision`
- ▶ `fixed` and `scientific`
- ▶ `dec`, `hex`, `oct`

Number of decimal places to represent. etc.

5. Input

In C++, the object `cin` is used to take input from the standard input stream (usually, this is the keyboard). It is a name for the **Console INput**.

In conjunction with the operator `>>` (called the **get from** or **extraction** operator), it assigns data from input stream to the named variable.

(In fact, `cin` is an **object**, with more sophisticated uses/methods than will be shown here).

5. Input

02Input.cpp

```
4 #include <iostream>
5 #include <iomanip> // needed for setprecision
6 int main()
7 {
8     const double StirlingToEuro=1.19099; // Correct 28/01/2026
9     double Stirling;
10    std::cout << "Input amount in Stirling: ";
11    std::cin >> Stirling;
12    std::cout << "That is worth "
13        << Stirling*StirlingToEuro << " Euros\n";
14    std::cout << "That is worth " << std::fixed
15        << std::setprecision(2) << "\u20AC"
16        << Stirling*StirlingToEuro << std::endl;
17    return(0);
18 }
```

6. Flow of control – if-blocks

`if` statements are used to conditionally execute part of your code.

Structure (i):

```
if ( exprn )
{
    statements to execute if exprn evaluates as
non-zero
}
else
{
    statements if exprn evaluates as 0
}
```

6. Flow of control – if-blocks

Note: { and } are optional if the block contains a single line.

Example:

```
if (x == 1)  
    x++;
```

is the same as

```
if (x == 1)  
{  
    x++;  
}
```

Python:

```
if (x == 1):  
    x += 1
```

Catch

```
if (x > y)  
    x++;
```

is not the same as

```
if (x > y)  
{  
    x++;  
}
```

6. Flow of control – if-blocks

The argument to `if()` is a **logical expression**.

Example

- ▶ `x == 8`
- ▶ `m == '5'`
- ▶ `y <= 1`
- ▶ `y != x`
- ▶ `y > 0`

More complicated examples can be constructed using

- ▶ **AND** `&&`
and
- ▶ **OR** `||`.

6. Flow of control – if-blocks

v

03EvenOdd.cpp

```
int main(void)
.12 {
    int Number;

    std::cout << "Please enter an integer: ";
16    std::cin >> Number;

    18    if ( (Number%2) == 0)
        std::cout << "That is an even number." << std::endl;
20    else
        std::cout << "That number is odd." << std::endl;
22    return(0);
}
```

$a \% b$ is the remainder on dividing
a by b.

6. Flow of control – if-blocks

More complicated examples are possible:

Structure (ii):

```
if ( exp1 )
{
    statements to execute if exp1 is "true"
}
else if (exp2)
{
    statements run if exp1 is "false" but exp2 is "true"
}
else
{
    "catch all" statements if neither exp1 or exp2 true.
}
```

} can
have
multiple
"if
else".

6. Flow of control – if-blocks

04Grades.cpp

```
12 int NumberGrade;
13 char LetterGrade;
14
15
16     std::cout << "Please enter the grade (percentage): ";
17     std::cin >> NumberGrade;
18     if ( NumberGrade >= 70 )
19         LetterGrade = 'A';
20     else if ( NumberGrade >= 60 )      60 ≤ B ≤ 69
21         LetterGrade = 'B';
22     else if ( NumberGrade >= 50 )      50 ≤ C ≤ 59
23         LetterGrade = 'C';
24     else if ( NumberGrade >= 40 )
25         LetterGrade = 'D';
26     else
27         LetterGrade = 'E';
28
29
30     std::cout << "A score of " << NumberGrade
31             << "% corresponds to a "
32             << LetterGrade << "." << std::endl;
```

6. Flow of control – if-blocks

The other main flow-of-control structures are $(a==b)? a=c: a=d;$

- ▶ the ternary the `?:` operator, which can be useful for formatting output, in particular, and
- ▶ `switch ... case` structures.

if
a != b

Exercise 2.1

Find out how the `?:` operator works, and write a program that uses it.

Hint: See Example 07IsComposite.cpp

Exercise 2.2

Find out how `switch... case` construct works, and write a program that uses it.

Hint: see https://runestone.academy/ns/books/published/cpp4python/Control_Structures/conditionals.html

We meet a **for**-loop briefly in the Fibonacci example. The most commonly used loop structure is **for**

```
for (initial value; test condition; step)  
{  
    // code to execute inside loop  
}
```

or
increment.

Example: 05CountDown.cpp

```
10 int main(void)  
{  
12     int i;  
13     for (i=10; i>=1; i--)  
14         std::cout << i << "... ";  
15     std::cout << "Zero!\n";  
16     return(0);  
}
```

i--
i is the same
as $i = i - 1$;
which is the same
as $i = 1$;

1. The syntax of `for` is a little unusual, particularly the use of semicolons to separate the “arguments”.
2. All three arguments are optional, and can be left blank.
Example:

E.g., `for (i=0; i<4; i++)`
`{ /* do stuff */ }`

is the same as

```
i=0;  
for ( ; i<4 ; i++)  
{ /* do stuff */ }
```

3. But it is not good practice to omit any of them, and very bad practice to leave out the middle one (test condition).

7. Loops

for loops

- It is very common to define the increment variable within the for statement, in which case it is “local” to the loop. Example:

```
int i;  
for(i=0; i<10; i++)  
{ -- }
```

} for (int i=0; i<10; i++)
 { -- } .

- As usual, if the body of the loop has only one line, then the “curly braces”, { and }, are optional.
- There is no semicolon at the end of the for line.

```
for (int i=0; i<10; i++)  
/* do stuff */
```

} wrong!

Finished here, but do read through the next slide before the lab

The other two common forms of loop in C++ are

- ▶ `while` loops
- ▶ `do ... while` loops

Exercise 2.3

Find out how to write a `while` and `do ... while` loops. For example, see

https://runestone.academy/ns/books/published/cpp4python/Control_Structures/while_loop.html

Rewrite the **count down** example above using a

1. `while` loop.
2. `do ... while` loop.