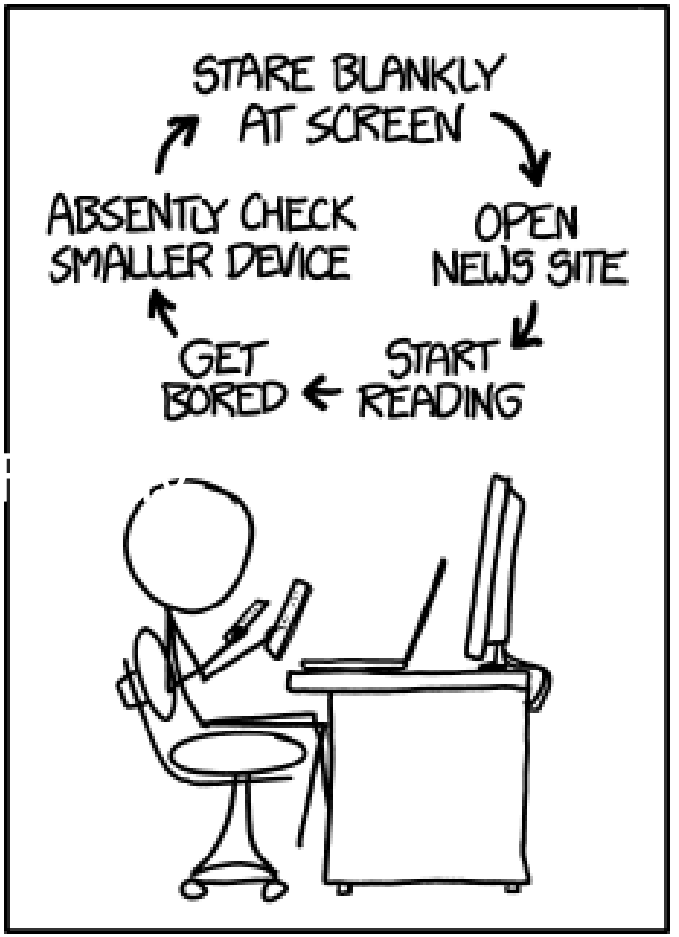
**CS319: Scientific Computing**

# I/O, flow, loops, and functions

Annotated slides fr

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 week's classes:

1. Preview of Lab 1
2. Recall from Week 2
3. Basic Output
4. Output Manipulators
   - endl
   - setw
5. Input
6. `if`-blocks
7. Loops
8. Functions
   - Function Header
   - Function definition
   - A mathematical function
   - E.g, Prime?
   - `void` functions
9. Numerical Integration
   - The basic idea
   - The code
   - Trapezium Rule as a function

Slides and examples:
niallmadden.ie/2526-CS319

# 8. Functions

A good understanding of **functions**, and their uses, is of prime importance.

Some functions return/compute a single value. However, many important functions return more than one value, or modify one of its own arguments.

For that reason, we need to understand the difference between **call-by-value** and **call-by-reference** (⟵ later).

**Warning:** In Scientific Computing, we use the term "**function**" in two different, but related ways:

- A mathematical function, such as $f(x) = e^{-x}$ or $u(x, y) = \sin(\pi x) \cos y$.
- A function we code to preform a task, such a determining if an integer is prime.

And often we'll combine both ideas!

# 8. Functions

Every C++ program has at least one function: `main()`

## Example

```
#include <iostream>
int main(void )
{
   /* Stuff goes here */
   return(0);
}
```

same as `int main()`

Each function consists of two main parts: Header/Prototype and Body/Definition.

1. Header

The Function "header" or **prototype** gives the function's

- return value data type, or `void` if there is none, and
- parameter list data types or `void` if there are none.
- The header line ends with a semicolon.

The prototype is often given near the start of the file, before the **main()** section.

**Syntax for function header:**

```
ReturnType FnName (type1, type2, ...);
```

**Examples:**

```
int  find_largest_divisor (int);
int  round_to_nearest (float);
int  find_lcm (int, int);
```

```
bool  IsClose(float, float, float);
or
bool IsClose(float x, float y, float epsilon);
   // true if |x-y| <= epsilon
```

Note: you can give variable names, optionally, in the header, but they are just place-holders and ignored by the compiler.

## 2. Function definition

- The **function definition** can be anywhere in the code (after the header).
- First line is the same as the prototype, except variables names need to be included, and that line does not end with a semi-colon.
- That is followed by the body of the function contained within curly brackets.

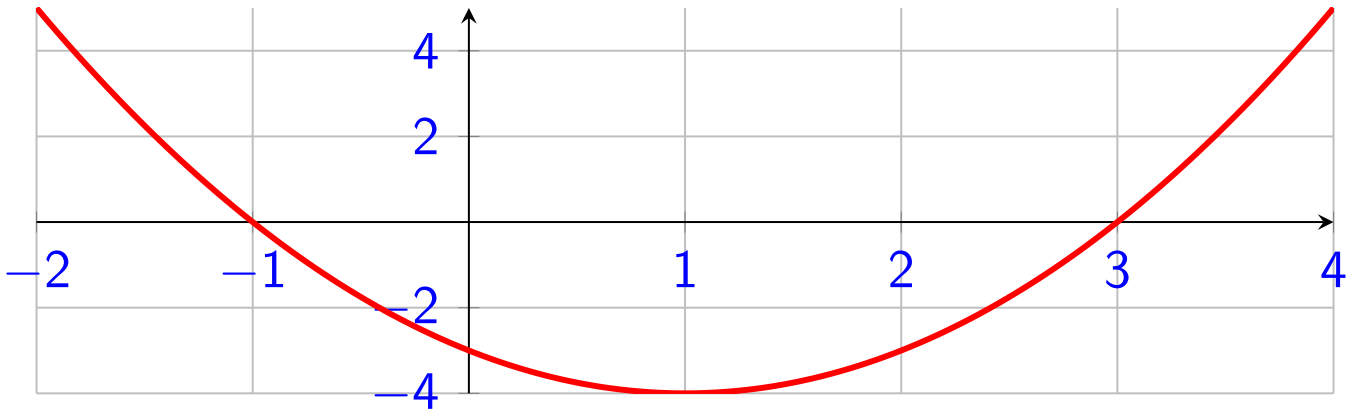
**Syntax:**

```
ReturnType FnName (type1 param1, type2 param2, ...)
{
      statements
}
```

- `ReturnType` is the data type of the data returned by the function.
- `FnName` the identifier by which the function is called.
- `type1 param1, ...` consists of (eg int, float, ...)
  - the data type of the parameter
  - the name of the parameter will have in the function. It acts within the function as a local variable.
- the statements that form the function's body, contained with braces `{...}`.

Since this is a course on scientific computing, we'll often need to define mathematical functions from $\mathbb{R} \to \mathbb{R}$ (more or less), such as $f(x) = e^{-x}$. Typically, such functions map one or more `double`s onto another `double`.

The example we'll look at is $f(x) = x^2 - 2x - 3$.



06MathFunction.cpp

```
#include <iostream>
#include <iomanip>

double f(double x) // x^2 - 2x - 3
{
  return (x*x - 2*x -3);
}

int main(void){
  double x;
  std::cout << std::fixed << std::showpoint;
  std::cout << std::setprecision(2);
  for (int i=0; i<=10; i++)
  {
    x = -1.0 + i*.5;
    std::cout << "f("<< x << ")="<< f(x) << std::endl;
  }
  return(0);
}
```

Note: for simple functions that appear before the main() the function header can be omitted.

In this example, we write a function that takes an non-negative integer input and checks it its a composite (`true`) or prime (`false`).

07IsComposite.cpp (header)

```
// 07IsComposite.cpp
// An example of a simple function, to check if an int is composite
// Author: Niall Madden
// Week 3: 2526-CS319 - Scientific Computing

#include <iostream>

bool IsComposite(int i);
```

## Calling the `IsComposite` function

`07IsComposite.cpp` (main)

```
10 int main(void )
   {
12   int i;

14   std::cout << "Enter a natural number: ";
     std::cin >> i; // Warning: should check this is positive

     std::cout << i << " is a " <<
18     (IsComposite(i) ? "composite":"prime") << " number."
               << std::endl;

     return(0);
22 }
```

## Defining the `IsComposite` function

`01IsComposite.cpp` (function definition)

```
24 // Check if i as a Composite number (i.e., not prime)
   // Return "true" if it is composite.
26 // Return "false" if it is prime.
   bool IsComposite(int i) // should check i > 0
28 {
     int k;
30   for (k=2; k<i; k++)
       if ( (i%k) == 0)
32       return(true); // note: function terminates at first "return()"

34   // If we get to here, then i has no divisors between 2 and i-1
     return(false);
36 }
```

Most functions will return some value (they are sometimes called "*fruitful*" functions). In rare situations, they don't, and so have a `void` return value.

`08Kth.cpp` (header)

```
// 08Kth.cpp:
// An example of a void function.
// CS319, Week 3
// Author: Niall Madden
// Date: Jan 2026
// Week 3: CS319 - Scientific Computing

#include <iostream>

void Kth(int i);
```


Puzzle: What is the next term in the sequence
s, t, n, d, r, d, t, h, t, h, t, h,
st, nd, rd, th, th, ...

02Kth.cpp (main)

```
12 int main(void )
   {
14   int i;

16   std::cout << "Enter a natural number: ";
     std::cin >> i;

     std::cout << "That is the ";
20   Kth(i);
     std::cout << " number." << std::endl;

     return(0);
24 }
```

08Kth.cpp (function definition)

```
26 // FUNCTION KTH
   // ARGUMENT: single integer
28 // RETURN VALUE: void (does not return a value)
   // WHAT: if input is 1, displays 1st, if input is 2, displays 2nd,
30 // etc.
   void  Kth(int i)
32 {
     std::cout << i;
34   i = i%100; //  remainder on dividing i by 100
     if ( ((i%10) == 1) && (i != 11))
36     std::cout << "st";
     else if ( ((i%10) == 2) && (i != 12))
38     std::cout << "nd";
     else if ( ((i%10) == 3) && (i != 13))
40     std::cout << "rd";
     else
42     std::cout << "th";
   }
```

AND.

Recall : OR is ||

# 9. Numerical Integration

A **Numerical Integration** (aka "Quadrature") method is an algorithm for estimating definite integrals. The applications are far too numerous to list, but feature in just about every area of Applied Mathematics, Probability Theory, and Engineering, and even some areas of pure mathematics.

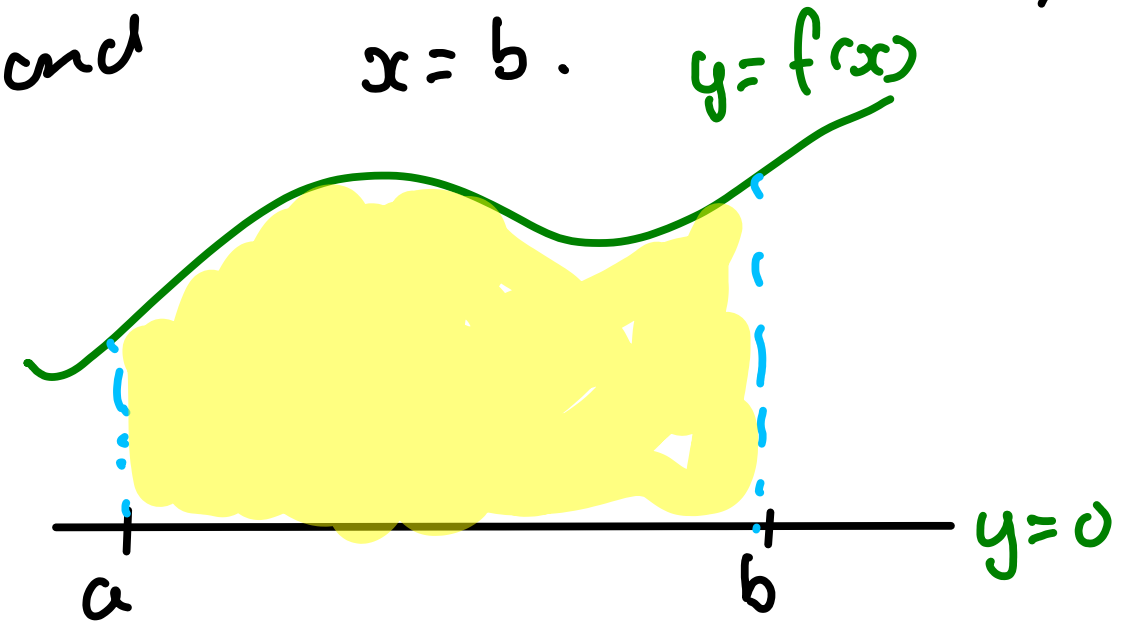
Although the history is ancient, it continues to be a hot topic of research, particularly when computing with high-dimensional data.

In this section, we want to estimate definite integrates of one-dimensional functions:

$$\int_a^b f(x)dx.$$

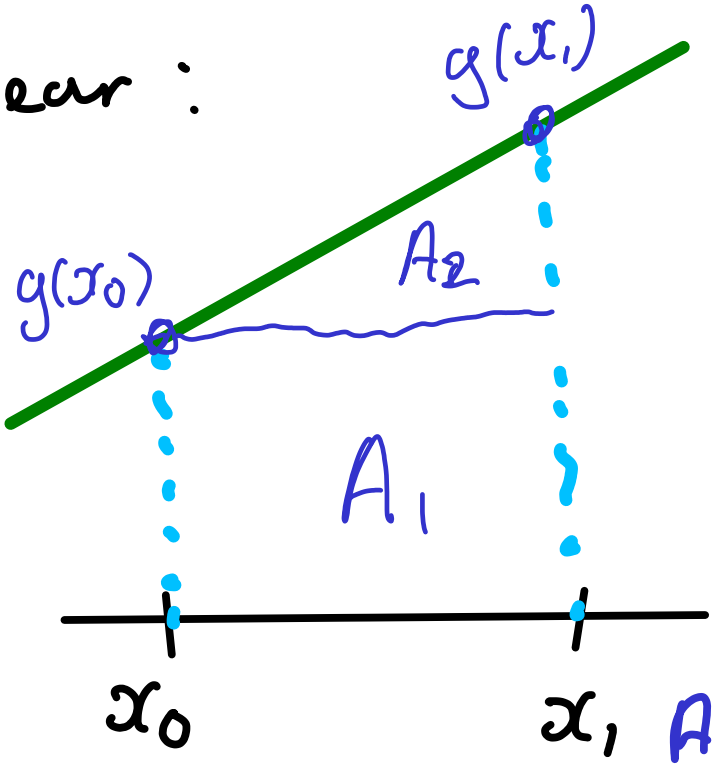
We'll use one of the simplest methods: the Trapezium Rule.

The value of $\int_a^b f(x)\,dx$ is the area between $y = f(x)$, $y = 0$, $x = a$ and $x = b$.




Usually there is no formula for $\int f(x)\,dx$.

But we can write down a formula for $\int_{x_0}^{x_1} g(x)\,dx$ for some simple function $g(x)$ ... e.g. if $g$ is linear:




$A_1$ has area $(x_1 - x_0)\, g(x_0)$

$A_2$ has area $\frac{1}{2}(x_1 - x_0)(g(x_1) - g(x_0))$

So

$$A_1 + A_2 = \frac{1}{2}(x_1 - x_0)\left(g(x_0) + g(x_1)\right)$$