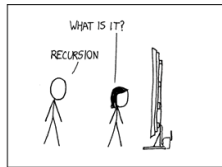
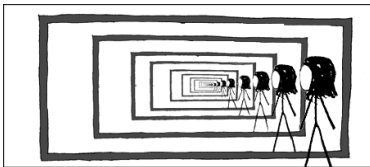


CS319: Scientific Computing

# Quadrature, and Functions in C++

Dr Niall Madden

Week 4 **9am** and 4pm, 31 January, 2024



Slides and examples: <https://www.niallmadden.ie/2324-CS319>

	Mon	Tue	Wed	Thu	Fri
9 – 10			✓	LAB	
10 – 11					
11 – 12					LAB
12 – 1					LAB
1 – 2					
2 – 3					
3 – 4					
4 – 5			✓		

Reminder: **labs again this week** (and every week).

- ▶ Thursday 9-10
- ▶ Friday 11-12
- ▶ Friday 12-1.

For more, see <https://www.niallmadden.ie/2324-CS319/#labs>

## 9am: Section 1, 2, and 3.1

**1** Overview of this week's classes

- Why quadrature?

**2** Functions (again)

- Header
- Function definition
- E.g, Prime?
- void functions

**3** Numerical Integration

- The basic idea

- The code

Start here 4pm

- Trapezium Rule as a function

**4** Functions as arguments to functions**5** Functions with default arguments**6** Pass-by-value**7** Function overloading**8** Detailed example

Slides and examples:


<https://www.niallmadden.ie/2324-CS319>

# Overview of this week's classes

This week, we will study the use of functions in C++, which we started at the end of Week 3.

However, we'll motivate some of this study with a key topic in Scientific Computing: **Quadrature**, which is also known as **Numerical Integration**.

{ Later, we'll use this as an opportunity to study the idea of  
( **experimental analysis** of algorithms.  
next week.

- 
- ▶ A **Quadrature** method, in one dimension, is a method 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.
  - ▶ They are methods for estimating integrals of functions. So this gives us two reasons to code functions:
    - (i) As the functions we want to integrate;
    - (ii) As the algorithms for doing the integration.

But before we get on to actual methods, we'll review some notes from last week, and examples I didn't get to.

*Simplest common method: Trapezium Rule.*

In Week 3 we studied how to write functions in C++.

- ▶ Each function consists of two main parts: a **header** (or “prototype”) and the function definition.
- ▶ The “header” is a single line of code that appears (usually) before the `main()` function. It 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 parameter list can include variable names, but they are treated as comments.
- ▶ The header line ends with a semicolon.

### Syntax for function header:

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

**Examples:**

A function that takes an int as argument and returns of float; and is called  
Convert:

```
float Convert(int i);
```

---

A function that returns nothing, but takes as inputs a double, a char, and a string:

```
void DoSomething(double x, char c, string s);
```

---

However, a function can't return more than one value:

```
float, float = ComputeTwoThings(int x)
```

is not allowed! But we can return arrays ( next week).

- ▶ 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.

Since the variable names in the header are optional or, at best, place-holders, they can be different in the function definition.



**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
  - ▶ 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 **{...}**.

## 00IsComposite.cpp (header)

```
2 // 00IsComposite.cpp
// An example of a simple function.
// Author: Niall Madden
4 // Date: 31 Jan 2024
// Week 4: CS319 - Scientific Computing

#include <iostream>

bool IsComposite(int i); } function header.
```



Return type is "bool" which can be "true" or "false".

Takes a single argument (which is an int).

This will return "true" if the input is composite, and false otherwise (i.e., is not composite/prime).

## 00IsComposite.cpp (main)

```
12 int main(void )  
13 {  
14     int i;  
  
16     std::cout << "Enter a natural number: ";  
17     std::cin >> i;  
  
18     std::cout << i << " is a " <<  
19         (IsComposite(i) ? "composite" : "prime") << " number."  
20         << std::endl;  
  
22     return (0);  
23 }
```

Here we use the ?: operator.

## 00IsComposite.cpp (function definition)

```
28 bool IsComposite(int i)
   {
30     int k;
       for (k=2; k<i; k++)
32         if ( (i%k) == 0) if ("remainder on dividing i by k is zero)
           return(true); returns "true" AND exits function.

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

Most functions will return some value. In rare situations, they don't, and so have a `void` return value

01Kth.cpp (header)

```
2 // 01Kth.cpp:  
2 // Another example of a simple function.  
// Author: Niall Madden  
4 // Date: 31 Jan Feb 2024  
// Week 04: CS319 - Scientific Computing  
6 #include <iostream>  
void Kth(int i); // header
```

*This will return*

Question: what is next in the sequence {s,t,n,d,r,d,t,h,t,h,...}

## 01Kth.cpp (main)

```
10 int main(void )  
11 {  
12     int i;  
  
14     std::cout << "Enter a natural number: ";  
15     std::cin >> i;  
  
16     std::cout << "That is the ";  
17     Kth(i); // outputs "st", "nd", "rd" or "th"  
18     std::cout << " number." << std::endl;  
  
20     return(0);  
21 }
```

## 01Kth.cpp (function definition)

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

first digit is 1  
but second is not 11.

Notice - no return value.

# Numerical Integration

Numerical integration is an important topic in scientific computing. 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 integrals of one-dimensional functions:

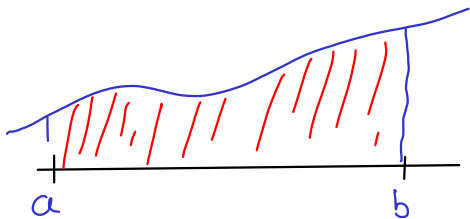
$$\int_a^b f(x) dx.$$

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

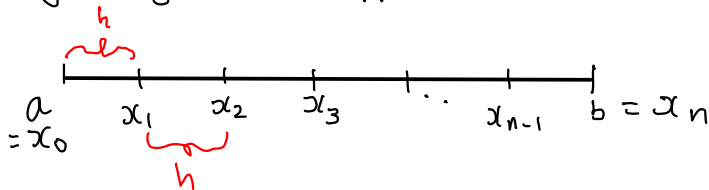


Idea  $\int_a^b f(x) dx$  is the area between  $f(x)$  and the  $x$ -axis, and between  $a$  &  $b$ .

Eg



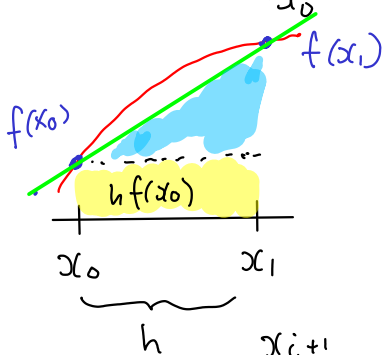
of length  $h = \frac{b-a}{n}$



Idea

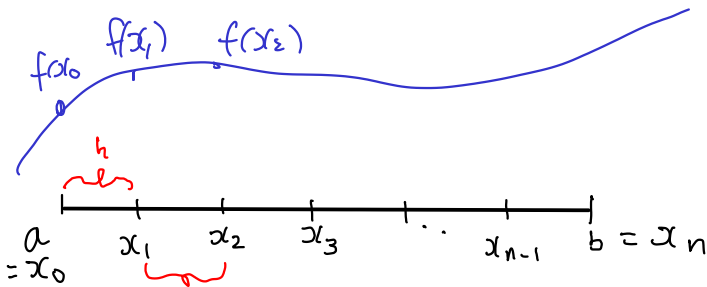
- Choose a natural number,  $n$
- Divide  $[a, b]$  into  $n$  intervals

• Estimate  $\int_{x_0}^{x_1} f(x) dx$  as  $\frac{h}{2} (f(x_0) + f(x_1))$



$$\begin{aligned} \int_{x_0}^{x_1} f(x) dx &\approx h f(x_0) \\ &+ \frac{h}{2} (f(x_1) - f(x_0)) \\ &= \frac{h}{2} (f(x_0) + f(x_1)). \end{aligned}$$

• Indeed  $\int_{x_i}^{x_{i+1}} f(x) dx = \frac{h}{2} (f(x_i) + f(x_{i+1})).$



$$\begin{aligned} \int_a^b f(x) dx &= \int_{x_0}^{x_1} f(x) dx + \int_{x_1}^{x_2} f(x) dx + \dots + \int_{x_{n-1}}^{x_n} f(x) dx \\ &= h \left( \frac{1}{2} f(x_0) + f(x_1) + f(x_2) + \dots + \frac{1}{2} f(x_n) \right) \end{aligned}$$

Finished here at 10am