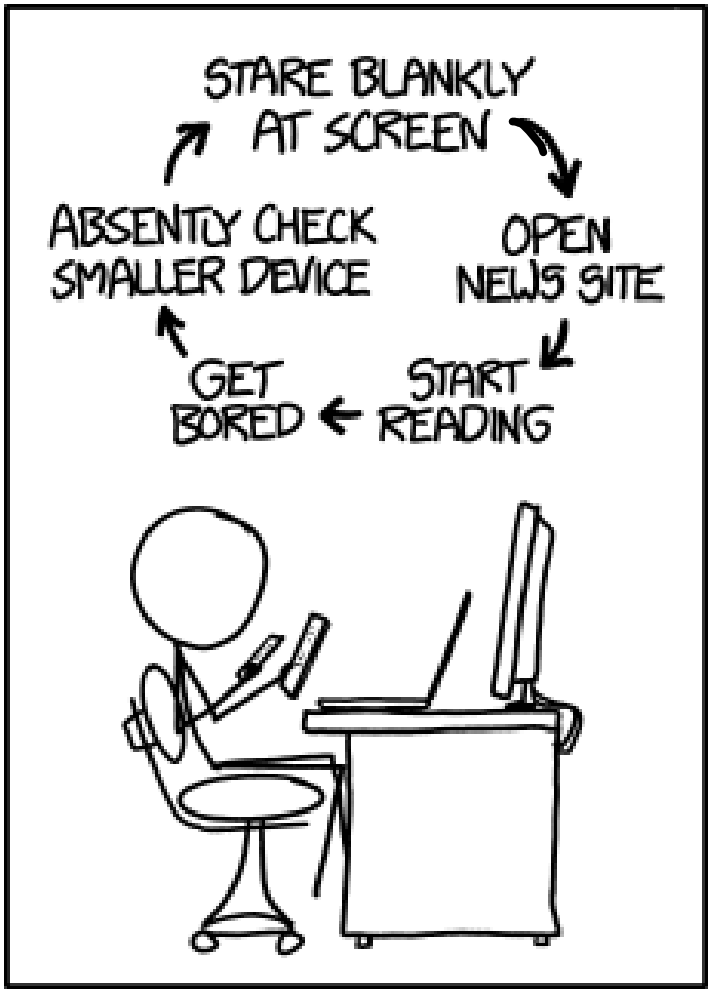
CS319: Scientific Computing

`double`, I/O, flow, loops, and functions

Dr Niall Madden

Week 3: 29 and 31 January, 2025



Source: xkcd (1411)

Slides and examples: https://www.niallmadden.ie/2425-CS319

Outline

Slides and examples:
https://www.niallmadden.ie/2425-CS319

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
}
```

logical expression: Evaluates as true or false.

Note use of { & }. These define a program block, like indentation in Python

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++;  y = x;
```

is not the same as

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

Flow of control – `if`-blocks

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

Example

- `x == 8` → true if 8 is stored in x.
- `m == '5'`
- `y <= 1` → $y \leq 1$
- `y != x` → true if $y \neq x$.
- `y > 0`

More complicated examples can be constructed using

- **AND** `&&` and
- **OR** `||`.

→ `(x == y) && (y == z)` (&& = and)

`(x==y) || (y == z)` (|| = or)

Flow of control – if-blocks

03EvenOdd.cpp

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

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

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

Recall a % b is the remainder on dividing a by b. Eg 23 % 10 is 3. ("modulo operator").

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 else if's

Flow of control – if-blocks

04Grades.cpp

```
12    int NumberGrade;
      char LetterGrade;

      std::cout << "Please enter the grade (percentage): ";
16    std::cin >> NumberGrade;
      if ( NumberGrade >= 70 )
18       LetterGrade = 'A';
      else if ( NumberGrade >= 60 )
20       LetterGrade = 'B';
      else if ( NumberGrade >= 50 )
22       LetterGrade = 'C';
      else if ( NumberGrade >= 40 )
24       LetterGrade = 'D';
      else
26       LetterGrade = 'E';

28    std::cout << "A score of " << NumberGrade
                << "% cooresponds to a "
30              << LetterGrade << "." << std::endl;
```

so between 60 & 69

between 50 & 59 etc.

```
else if (NumGrade == 39)
{
    NumberGrade++;
     LetterGrade = 'D';
}
```

x++; means x = x+1;

Flow of control – `if`-blocks

The other main flow-of-control structures are

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

x ? (op1) : (op2)

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
}
```

Example: 05CountDown.cpp

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

Output:
10... 9... 8... 7... 6... 5...4...
3... 2... 1... Zero!

body of the loop has just 1 line, so { – } not used.

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:

```
if (i=-4; i<=4; i+=2)
{   x=3*i;
}
```

This is the same as:

```
i=-4;
for ( ; i <=4; i+=2)
{   x = 3*i;  }
```

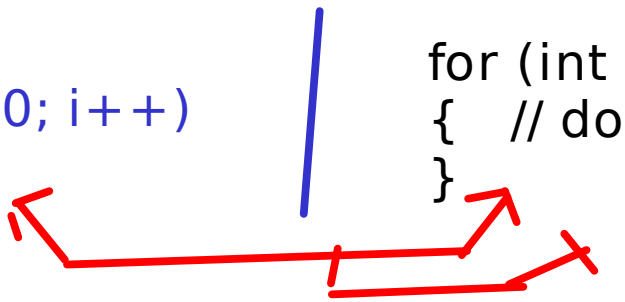
```
for (i=-4; i<=4; )
{ x=x3*i;
  i += 2;   // same as i=i+2;
 }
```

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

4. 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=1; i<10; i++)
{ // do stuff
}
```

```
for (int i=1; i<10; i++)
{   // do stuff
}
```



These are the same, except for scope of i.

5. As usual, if the body of the loop has only one line, then the "curly braces", { and }, are optional.

6. There is no semicolon at the end of the `for` line.

WRONG:

```
for (int i=1; i<10; i++);
{
  // do stuff
}
```

not a syntax error, but a logical one.

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

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

We'll use these frequently – they are important.

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.

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).

Functions

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

Example

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

Functions

Each function consists of two main parts:

- Function "header" or **prototype** which 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 prototype is often given near the start of the file, before the **main()** section.

- **Function definition**. Begins with the function's name (identifier), parameter list and return type, followed by the body of the function contained within curly brackets.

Finished here Friday.