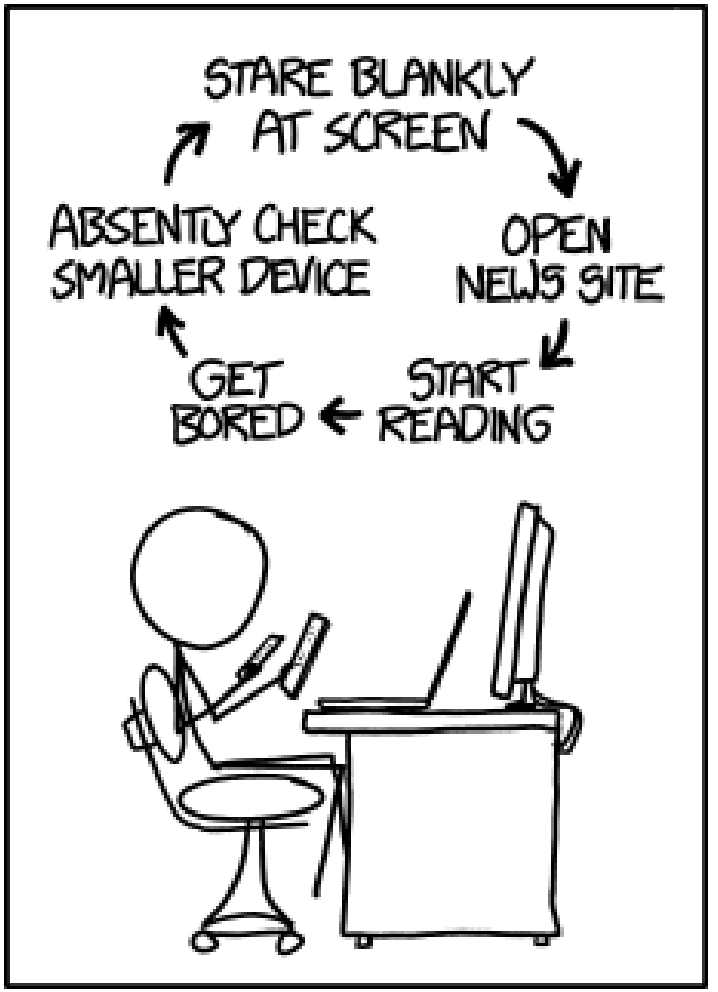
Annotated slides from 4pm class

CS319: Scientific Computing

I/O, flow, loops, and functions in C++

Dr Niall Madden

Week 3: **9am and 4pm**, 24 January, 2024



Source: xkcd (1411)

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

Outline

1. Recall from Week 2
2. Output Manipulators
   - endl
   - setw
3. Input
4. Flow of control – `if`-blocks
5. Loops
6. Functions
   - E.g, Prime?
   - `void` functions
7. Pass-by-value

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

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'` True if m stores the character '5'
- `y <= 1` y less than or equal to 1
- `y != x` y not equal to x.
- `y > 0`

More complicated examples can be constructed using

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

Note: writing
if (x = 8) {.....}
is probably the most common bug in C++ code!

Flow of control – if-blocks

03EvenOdd.cpp

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

     std::cout << "Please enter an integrer: ";
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);
   }
```

x % n is the remainder on dividing x by n.

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.
}
```

One can have many "else if" statements.

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;
```

$60 \leq x < 70$

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.

Exercise 2.1

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

Hint: See Example `06IsComposite.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
}
```

Note
i++ means
i = i + 1

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);
   }
```

Initial value: $i = 10$

Continuation: $i >= 1$

Step : $i--$

means: set $i = i - 1$

at the end of each pass through the loop.

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:
```
i = 10
for ( ; i >= 1 ; i--)
```
   same as
```
for (i=10; i>=1; i++)
```
Similarly
```
for (i = 10; i >= 1;)
  {
  cout << " - " << i;
  i--;
  }
```
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:

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

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

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

```
for (int i=0; i<10; i++) ;
{   ..... }
skips the loop!!!
```

Same for if statements.

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.

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.

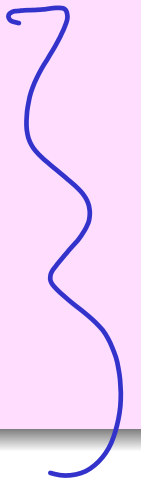
For that reason, we need to understand the difference between **call-by-value** and **call-by-reference** (⟵ later).

Functions

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

Example

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



Return value type

int

function name

main

arg list

void

= empty.

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 names, parameter list and return type, followed by the body of the function contained within curly brackets.

Functions

Syntax:

```
ReturnType FnName ( param1, param2, ...)
{
      statements   // Last statement is usually "return()"
}
```

- `ReturnType` is the data type of the data returned by the function, eg int, float ...
- `FnName` the identifier by which the function is called.
- `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 `{...}`.

06IsComposite.cpp

```
bool IsComposite(int i)
{
  int k;
  for (k=2; k<i; k++)
    if ( (i%k) == 0)
      return(true);

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

data type for orgument, and the name of that within the function.

"bool" is a data type with values true or false.

Calling the `IsComposite` function:

06IsComposite.cpp

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

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

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

     return(0);
24 }
```

? : operator.

Finish here 5 pm.