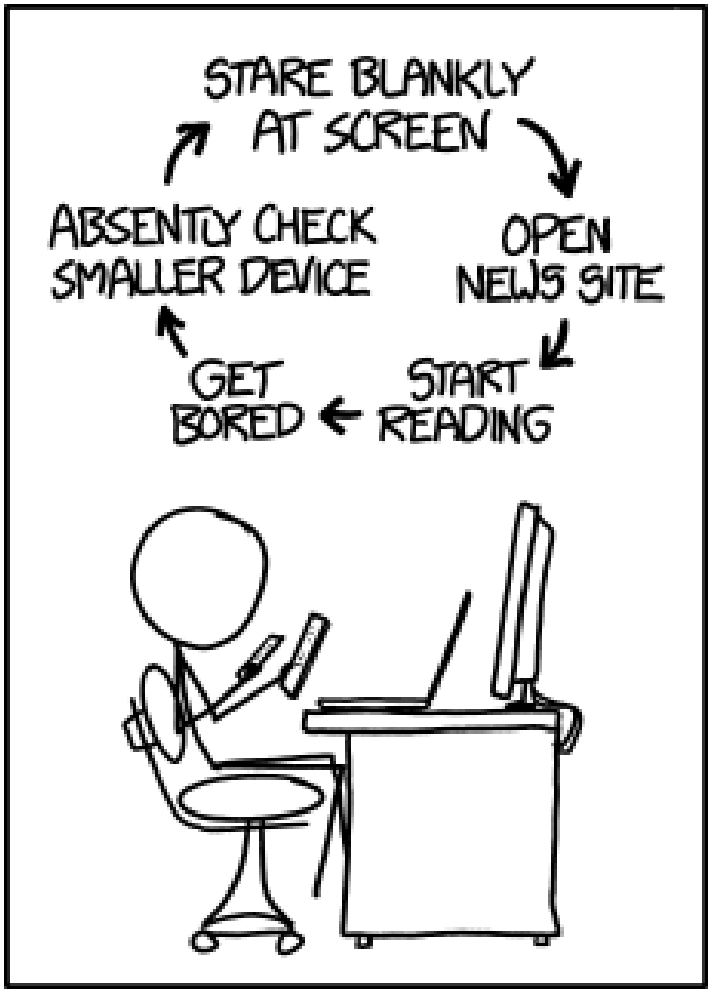
Annotated slides from 9am class

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)

	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 start this week**. Aim to attend any two of

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

Outline

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 1.7977×10^{308}.
  Its **machine epsilon** is $2^{-53} \approx 1.1102 \times 10^{-16}$.

Recall from Week 2

An important example:

00Rounding.cpp

```
9    int i, n;
10   float x=0.0, increment;

12   std::cout << "Enter a (natural) number, n: ";
     std::cin >> n;
14   increment = 1/( (float) n);

16   for (i=0; i<n; i++)
        x+=increment;

     std::cout << "Difference between x and 1: " << x-1
20             << std::endl;
```

Define two ints, i & n

Floats x, increment

Takes input from keyboard. & store in n.

change n to a float

increment = $\frac{1}{n}$

What this does: we add $\frac{1}{n}$ to x n times.

So we set $x = n\left(\frac{1}{n}\right) = 1$ (right?)

Recall from Week 2

- If we input $n = 8$, we get: $1 - 8\left(\frac{1}{8}\right) = 0$
- If we input $n = 10$, we get: $1 - 10\left(\frac{1}{10}\right) \approx 1.192016 \times 10^{-7}$

Other Results

n	$1 - n\left(\frac{1}{n}\right)$
50	-4.72×10^{-7}
100	-6.5×10^{-7}
200	-7.7×10^{-7}
3	0
4	
5	
⋮	
9	0

n	$1 - n\left(\frac{1}{n}\right)$
11	1.19×10^{-2}
12	11
13	0
16	0
17	0
18	2.39×10^{-7}

As well as passing variable names and strings to the output stream, we can also pass manipulators to change how variable values are displayed. Some manipulators (e.g., `setw`) require that *`iomanip`* is included.

We've already seen that we can use `std::endl` to print a new line at the end of some output.

01Manipulators.cpp

```
#include <iomanip>
int main()
{
  int i, fib[16];
  fib[0]=1; fib[1]=1;

  std::cout << "Without setw  manipulator" << std::endl;
  for (i=0; i<=12; i++)
  {
    if( i >= 2)
      fib[i] = fib[i-1] + fib[i-2];
    std::cout << "The " << i << "th " <<
      "Fibonacci Number is " <<  fib[i] << std::endl;
  }
```

will explain Syntax

like for i in range(13):

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

01Manipulators.cpp

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

Output Manipulators

Other useful manipulators:

- `setfill` —> by default it is a space. Sometimes e.g, 0 is better.
- `setprecision` —> number of decimal places.
- `fixed` and `scientific`
- `dec`, `hex`, `oct`

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 ***C*onsole *IN*put**.

later, this will be a file.

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).

Input

02Input.cpp

```
#include <iostream>
#include <iomanip> // needed for setprecision
int main()
{
  const double StirlingToEuro=1.16541; // Correct 17/01/2024
  double Stirling;
  std::cout << "Input amount in Stirling: ";
  std::cin >> Stirling;
  std::cout << "That is worth "
            << Stirling*StirlingToEuro << " Euros\n";
  std::cout << "That is worth " << std::fixed
            << std::setprecision(2) << "\u20AC"
            << Stirling*StirlingToEuro << std::endl;
  return(0);
}
```

Euro symbol.

output to 2 decimal places

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   (this is optional : but a good idea!)
{
   statements if exprn evaluates as 0, ie
}          exprn is false.
```

Flow of control – `if`-blocks

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

Example:

```
int a = 2, b = 3;

if (a < b)
  {
    cout << "a is less than b";
  }
```

Finished here

10 cm

Same as

```
if (a < b)
    cout << "a is less than b";
```

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** `||`.

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);
}
```

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.
}
```

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;
```

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.

Exercise 2.2

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

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);
   }
```

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:

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:

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.

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);
}
```

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
}
```

- `ReturnType` is the data type of the data returned by the function.
- `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)
30 {
     int k;
32   for (k=2; k<i; k++)
       if ( (i%k) == 0)
34       return(true);

36   // If we get to here, then i has no divisors between 2 and i-1
     return(false);
38 }
```

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 }
```

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

07Kth.cpp

```
10 void Kth(int i);

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;
```

07Kth.cpp

```
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;
     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";
   }
```

Pass-by-value

In C++ we need to distinguish between

- a variable's (unique) memory address
- a variable's identifier (might not be unique) item the value stored in the variable.

The classic example is function that

- takes two `int`eger inputs, `a` and `b`;
- after calling the function, the values of `a` and `b` are swapped.

Pass-by-value

To understand this example, it is important to understand the difference between a

1. **local variable**, which belongs only to the function (or block) in which it is defined;
2. **global variable**, which belongs to the whole programme, and can be accessed in any function (or block).

(Global variables are very uncommon, but we'll have a look at them in some lab exercises).

Pass-by-value

08SwapByValue.cpp

```
#include <iostream>
void Swap(int a, int b);

int main(void )
{
  int a, b;

  std::cout << "Enter two integers: ";
  std::cin >> a >> b;

  std::cout << "Before Swap: a=" << a << ", b=" << b
            << std::endl;
  Swap(a,b);
  std::cout << "After Swap: a=" << a << ", b=" << b
            << std::endl;

  return(0);
}
```

Pass-by-value

```
void Swap(int x, int y)
{
  int tmp;

  tmp=x;
  x=y;
  y=tmp;
}
```

This won't work.

We have passed only the *values stored in the variables a and b*. In the `swap` function these values are copied to local variables x and y. Although the local variables are swapped, they remained unchanged in the calling function.

What we really wanted to do here was to use **Pass-By-Reference** where we modify the contents of the memory space referred to by `a` and `b`. This is easily done...

Pass-by-value

...we just change the declaration and prototype from

```
void Swap(int x, int y)  // Pass by value
```

to

```
void Swap(int &x, int &y)  // Pass by Reference
```

the pass-by-reference is used.