Introduction to Computing
IIb - Control Structures and I/O in C

Jonathan Mascie-Taylor
(Slides originally by Quentin CAUDRON)

Centre for Complexity Science,
University of Warwick

Outline

IDE

Do you have an IDE installed ?

Suggestion : www.codeblocks.org

Downloads → Binary Release → WITH MingW

Hello World

```c
#include <stdio.h>

int main()
{
    // Print something to the screen
    printf("Hello, Complexity !\n");

    // Tell the OS that the code was successful
    return 0;
}
```

#include <stdio.h>

This tells the preprocessor to include the Standard In/Out Library.

A library is a group of predefined functions that you have access to once you've included them. Here, we want to print to screen, using `printf`, so we need to include `stdio.h` because it contains the definition for this function.

Any line beginning with # is a preprocessor directive. These commands are run before compilation process begins. `#include <header.h>` tells the compiler to copy the entire of the `header.h` file into your code, at the location where you `#included` it.

int main()

This notation means that there exists something called `main`, that is a function (because it has round brackets after its name), and that it returns an integer.

The word `main` is reserved, which means when you use it, it has a set behaviour. In the case of `main`, it is the function that any executable tries to run. If you called this anything else, your code would do nothing.

The reason `main` returns an integer is to tell the operating system how the code ran. It will return zero if it ran successfully, and a value larger than zero if there were errors.

```
{
```

Curly brackets are used to define sections of code. In this case, given directly after a function, it defines what the function does. Here, the code enclosed by the brackets makes up the function `main`, which gets called when you try running your program.

Remember to close all brackets, be they round, curly, square or angular.

// Print something to screen

Any line that begins with // is a comment. Anything following the double slash is completely ignored by the compiler.

```
/* You can also make a large block of text into a comment
by using this kind of notation. Anything in between the
slash-star and its ending star-slash is not compiled. */
```

Try getting into the practice of commenting your code, in order to improve its legibility.

printf

The `prinft` function takes in many combinations of arguments (it is *overloaded*), but in this example, it takes an argument in the form of a `string` of text.

```
printf("Hello, Complexity !\n");
```

This will write the data to the standard output, which is the screen.

printf

The `prinft` function takes in many combinations of arguments (it is *overloaded*), but in this example, it takes an argument in the form of a `string` of text.

```
printf("Hello, Complexity !\n");
```

This will write the data to the standard output, which is the screen.

The `\` is called an *escape character*. The character that follows it has a different interpretation - it won't be printed.

`\n` is interpreted as a *new line*. Thus, this will appear on two lines :

```
prinft("Here is some text.\nHere is more.");
```

Result :

```
Here is some text.
Here is more.
```

return 0;

`return` is a *keyword*. You'll notice that the line that really starts the programme is `int main()`, which is a *function* of type *int*. Therefore, when the function finishes, it expects to *return* an integer to whatever called it.

Because `main` is the special function that is called by the operating system when you run the program, the return value goes to the operating system, and in this way, it can track if the code executed correctly. A return value of 0 means the code exited successfully.

;

Semi-colons are used after every instruction to tell the compiler that a particular line of code is finished.

Note that instructions to the preprocessor (those that start with #) don't need semi-colons.

Brackets aren't instructions, so don't need them either.

`int main()` is a declaration of a function, and so doesn't require a semi-colon, because the instruction isn't finished - you first need to tell it what that function does. This goes for all functions.

Hello World

In your IDE, create a new, blank `.c` file. Write some code for Hello World, and then compile it by clicking on Build → Build and Run, or just hitting F9.

A console window should open, with the output you wrote in your code. CodeBlocks also adds a line telling you what the process returned (you should have 0), and how long it took to execute.

Variable Declaration and Types

A variable is declared using this syntax :

```
type name;
name = value;
```

Or, on the same line :

```
type name = value;
```

Variable Declaration and Types

A variable is declared using this syntax :

```
type name;
name = value;
```

Or, on the same line :

```
type name = value;
```

There are many types of variables. Here are the more common ones :

- `int` - an integer
- `char` - a single character
- `float` - a floating-point number
- `double` - a double-precision floating-point number

Arithmetic

Let's declare some integer variables :

```
int a;
int b;
```

Equivalently, we could have used :

```
int a, b;
```

Now that we've *declared* the variables, we can *define* them :

```
a = 12;
b = 17;
```

Arithmetic

Output the sum and the product of the two integers to screen :

```
printf("The sum of the numbers is %i\n", a+b);
printf("Their product is %i\n", a*b);
printf("The original numbers were %i and %i.\n", a, b);
```

You can have as many output variables in a single call to `printf` as you like.

scanf

Hard-coding values into the source code doesn't allow for much flexibility. Another option is to have the user input some variables via keyboard.

The `scanf` function works very much like the `printf` function :

```
double x;
printf("What is the value of x ?\n");
scanf("%f", &x);
```

scanf

Hard-coding values into the source code doesn't allow for much flexibility. Another option is to have the user input some variables via keyboard.

The `scanf` function works very much like the `printf` function :

```
double x;
printf("What is the value of x ?\n");
scanf("%f", &x);
```

The `&` character is a *reference* operator. Instead of telling `scanf` in what variable to store some input, you tell it *where in memory* you want to store it.

If Statements

```
1  if(a == b)
2  {
3    printf("a and b are both equal to %i.", a);
4  }
5
6  else if(a > b)
7  {
8    printf("%i is greater than %i.", a, b);
9  }
10
11 else
12 {
13   printf("%i is greater than %i.", b, a);
14 }
```

For Loops

```
1 int i;
2 for(i = 0; i < 10; i++)
3 {
4   printf("%i, ", i);
5 }
```

Result : `0, 1, 2, 3, 4, 5, 6, 7, 8, 9`

For Loops

```
1 int i;
2 for(i = 0; i < 10; i++)
3 {
4   printf("%i, ", i);
5 }
```

Result : `0, 1, 2, 3, 4, 5, 6, 7, 8, 9`

The structure of the `for` loop is :
`for(initialisation; condition; increment)`

The `initialisation` is performed once. Then, the contents of the loop are executed until the `condition` returns *false*, each time, executing the `increment`.

While Loops: Break and Continue

```
1  int i = 0;
2  while(i < 10)
3  {
4    if ( condition )
5    {
6      // Leave the loop
7      break;
8    }
9
10   if ( another condition )
11   {
12     // Jump to next iteration of the loop
13     continue;
14   }
15
16   // Some other code
17 }
```

Relational Operators

`==`	Equals (comparative)
`!=`	Does not equal
`>`	Greater than
`>=`	Greater than or equal to
`<`	Less than
`<=`	Less than or equal to

A Guessing Game

We're going to write a number guessing game. The idea is to have the user input a number between one and ten until they make the correct guess.

We'll need to think about the structure of the code, and what tools we'll use for what purpose.

A Guessing Game

We're going to write a number guessing game. The idea is to have the user input a number between one and ten until they make the correct guess.

We'll need to think about the structure of the code, and what tools we'll use for what purpose.

Write a guessing game using the following :

- A `while` loop, to continue the game until the user gets the guess right.
- An `if` statement, to check if the user input 0, in which case, exit the loop (use `break;` here)
- `printf` to give the user instructions, and outputting the user's last guess.
- `scanf` to accept user input

The Fibonacci Sequence

Write some code to generate the Fibonacci sequence to n terms :

$$F_n = F_{n-1} + F_{n-2},$$
$$F_0 = 0 \text{ and } F_1 = 1.$$

The Fibonacci Sequence

Write some code to generate the Fibonacci sequence to n terms :

$$F_n = F_{n-1} + F_{n-2},$$
$$F_0 = 0 \text{ and } F_1 = 1.$$

Use the following tools :

- `printf` and `scanf` to ask the user how many terms to generate
- A `for` loop to iterate over terms
- Three variables for F_n, F_{n-1} and F_{n-2}