day-1-basic-program-structure-exercises

September 6, 2023

1 Day 1: Basic program structure

1.0.1 Topics

- A basic C program
  - The `main` function
  - Header files
- Standard headers
- Basic output
- The `#include` directive

In the introduction you saw the following code when learning how to use these notebooks:

```c
#include <stdio.h>

int main(void) {
    printf("hello, world!\n");
    return 0;
}
```

These 6 lines demonstrate the basic structure of a C program. This notebook describes that structure as well as some of the fundamentals of programming in C.

1.0.2 The `main` function

Every C and C++ program must define a `main` function which is where the operating system will start the program. It is the job of the programmer to define (write) `main` and to return an integer value representing the success (or failure) of the program.

Conventionally, a return value from `main` of `0` means the program was successful and anything else means there was a failure (the exact mode of failure depends on the nature of the program).

This line indicates the start of `main`:

```c
int main(void) {
```

The `int` keyword (for integer) indicates that the `main` function returns an integer value, while the `(void)` indicates that the function accepts zero arguments.

The left brace `{` indicates the start of the function itself. All code upto the matching right brace `}` constitutes the body of the function.

The line:

```
4     printf("hello, world!\n");
```

Is a call to the `printf` function which prints its arguments. Its argument `"hello, world!\n"` is a string ending with `'\n'` which is a so-called ``escape sequence'' indicating the end of a line. The whole statement is terminated with a semi-colon `;`.

The final line of `main`:

```
5     return 0;
```

Indicates the end of the program, and returns a value of `0` to the operating system.

In the code cell below, change the program so that it greets you with your own name:

[ ]:
```
//// hello-me.c
#include <stdio.h>

int main(void) {
    printf("hello, world!\n");
    return 0;
}
```

For detailed reference information on `main`, see:

- The `main` function in C
- The `main` function in C++

1.0.3 Header files

In order to use the `printf` function, the code must first include some information about the function. This information is stored in the file `stdio.h` (a ``header'' file, which is known to your compiler), and is provided with the line:

```
1 #include <stdio.h>
```

Lines beginning with `#` are not code but rather **pre-processor directives** (instructions to the compiler to modify the program before compilation actually takes place). This line tells the compiler to substitute in-place the contents of the specified file. This ensures that all information needed to use the features provided by the header is made available during compilation.

The C standard defines a set of standard header files, a useful reference on them is available here.

1.0.4 Standard headers

Like any other language, C has a standard library (defined in the C standard and implemented by compiler developers). To access this functionality, we need to use the necessary header files in our source code (e.g. `#include <stdio.h>`).

This page lists the C standard headers: **C Standard Library header files**. Familiarise yourself with the available headers and what they are each for.

In which headers would you find the following:

1. Functions for printing formatted output (to `stdout`, a file or a buffer).
2. A function for comparing two strings.
3. The base-10 logarithm function `log10`.
4. A function for getting the imaginary part of a complex number.
5. Functions for opening and closing files.
6. Functions for managing memory (allocating and deallocating memory).

1.0.5 Variable declarations & definitions

1. In the cell below, declare variables `b` and `c` with the values -4 and -12 respectively. What should their type be?

[ ]:
```c
//// variables.c
#include <stdio.h>

int main(void) {
    int a = 1;

    // add b and c here...
    ...

    printf("a=%d, b=%d, c=%d\n", a, b, c);
    return 0;
}
```

2. If `a`, `b` and `c` from the cell above are the coefficients of a quadratic polynomial, define a `double` called `disc` equal to the discriminant of this polynomial (*hint*: the discriminant of the polynomial $ax^2 + bx + c$ is $b^2 - 4ac$):

[ ]:
```c
//// variables2.c
#include <stdio.h>

int main(void) {
    int a = 1;

    // add b and c here...
    ...

    printf("a=%d, b=%d, c=%d\n", a, b, c);

    // calculate disc here...
    double disc = ...
```

```
    printf("disc=%f\n", disc);
    return 0;
}
```

3. Write a program which calculates the weight of a cylinder of height \$1.25 m \$, radius $0.355m$ and density \$1.25 kg / m^3 \$ and stores the result in a variable `weight` (*hint*: the weight of an objest is equal to $W = mg$ where m is the mass and g is the acceleration due to gravity, $9.81m/s^2$)

[ ]:
```
//// cylinder-weight.c
#include <stdio.h>

int main(void) {
    ...
    return 0;
}
```

4. Suppose a bank issues a loan which earns 1.25% interest per month (i.e. an initial loan of £1000 will be worth £1012.50 after 1 month). Complete the program below to calculate:

- `interest_after_3_years`: the total interest earned if a loan is taken out over 3 years.
- `monthly_payment_GBP`: the monthly repayments needed to pay off a loan of £7,500 over 5 years.

Hints:

- if the loan earns 1.25% interest per month, by what factor f is it multiplied per month?
- what is the value of this multiplier after n months?

[ ]:
```
//// loan-amount.c
// this option allows you to use math.h if you want to:
//% LDFLAGS -lm

#include <stdio.h>
#include <math.h>

int main(void) {
    double monthly_interest_rate = 1.25;  // percent
    double interest_after_3_years = 0.00; // percent

    printf("1-month interest rate:    %6.2f%\n", monthly_interest_rate);
    printf("3-year  interest accrued: %6.2f%\n", interest_after_3_years);

    double initial_amount_GBP = 7500.00; // loan amount, GBP
    double monthly_payment_GBP = 0.00;   // monthly repayments (including␣
 ↪interest) over 5 years, GBP

    printf("monthly repayments on £%.2f loan: £%.2f\n", initial_amount_GBP,␣
 ↪monthly_payment_GBP);
```

```
    return 0;
}
```

1.0.6 Basic output

You've seen some examples above of how to print variables using `printf`, as well as how to control the format of the output. `printf` uses **conversion specifiers** (placeholders like `%d`) to describe how to interpret the variables to print.

You can read more about `printf` conversion specifiers here: `printf` conversion specifiers.

Conversion specifiers can control:

- the width of a printed value
- the alignment (left/right)
- padding (e.g. with zeros)
- precision (e.g. decimal places)

Questions:

1. What types are the `%d`, `%f`, `%s` and `%e` used to print?
2. What does the format specifier `%6d` mean?
3. What does the format specifier `%2.6f` mean?
4. In the code cell below, write code which prints the value of the `double` variable `x` in the following formats:
    1. in decimal notation, right-justified with a field width of 11 and 2 decimal places.
    2. in decimal notation, left-justified with a field width of 9 and 3 decimal places.
    3. in scientific notation, right-justified with a field width of 16 and 4 decimal places.
    4. in scientific notation, right-justified with a field width of 12, 2 decimal places and padded with leading zeros.

[ ]:
```c
//// print-formats.c
#include <stdio.h>

int main(void) {
    double x = 1234.0123456789;
    return 0;
}
```

5. Take a look at the family of printf-like functions in the standard library and consider these questions:
    1. Where does `printf` send its results to be printed?
    2. How does `fprintf` determine where to send its results to be printed?
    3. What happens when you use `fprintf` with the `stderr` output stream?

You might want to use the code cell below to experiment with `printf` and `fprintf`:

[ ]:
```c
//// experiment-with-printing.c
//% CFLAGS -Wall -Werror

int main(void) {
    return 0;
}
```

1.0.7 The `#include` directive

Lines beginning with `#` are instructions (directives) to the pre-processor. The pre-processor is a part of the compiler which executes before any source code is actually compiled and it provides what are essentially sophisticated text substitution & replacement capabilities.

For more information on the pre-processor, see: The Pre-processor

`#include` This directive searches for and includes a specified file, e.g. a standard header file with `#include <stdio.h>`.

This directive is somewhat analogous to the `import` statement in Python.

Using `<>` indicates a search among files under the control of the compiler, typically among a set of standard include directories, also housing the C standard library, along with include directories for system-wide 3rd-party software.

Using `""` indicates a search for files not necessarily belonging to the compiler, typically starting with the directory containing the current file.

1. Execute the code cell below and examine the output. What is the reason for this output? What is missing from the code?

[ ]:
```c
//// fix-me.c
//% LDFLAGS -lm

int main(void) {
    double alpha = 3.14159;
    printf("sin(%.2f)=%.2f\n", alpha, sin(alpha));
}
```

2. Execute this script to create an empty file called `stdio.h` in the notebook's directory. What does the pre-processor then do with the following code? What mistake has been made in this code?

[ ]:
```
%%file stdio.h
// this is an empty file in the current directory called "stdio.h"
```

[ ]:
```c
//// include.c
//% CFLAGS -Wall -Werror
#include "stdio.h"

```

```c
int main(void) {
    printf("hello!\n");
    return 0;
}
```