C Fundamentals Practice Notebook

Table of Contents

1 README

- This file is a practice file for C fundamentals.
- You will learn how to:
    1. understand and change syntax highlighting
    2. understand and use comments in C
    3. type, declare and initialize variables
    4. format printout and fix formatting errors
    5. read and store input from the command line
    6. name identifiers correctly
    7. structure programs efficiently
- Time: approx. 30-60 min.
- When you're done with a section move the cursor on the section heading and type `S-<right>` (or SHIFT+ <right-arrow>).

2 DONE Identify yourself

- replace the placeholder `[yourName]` in the header of this file by your name and save the file (`C-x C-s`).

3 DONE Syntax highlighting

1. Create a named code block 1 with a program that prints "Hello, world!" on the screen. Use the `<s` template to create the block.

   Tip: the program consists of only one mandatory line!

—— Put code block here ——

—— SOLUTION ——

```
puts("Hello, world!");
```

```
Hello, world!
```

2. Run the program in Emacs to make sure that it works.
3. Open the themes chooser with `M-x custom-themes`. In the buffer, uncheck the current theme and check another. Then return here (`C-x b`) to see the effect on your hello world program.
4. If you like a theme especially well, you save it for future sessions if you like by clicking on `Save Theme Settings`. You get the message `Custom theme saved for future sessions` in the minibuffer.

4 DONE Fix the program

The program 1 should print out the sentence "`I love my dog 'Milly'`". But it doesn't. Something went wrong with the comments. Fix the program and run it again.

```
printf("I "); /****
                    printf("love ");
                    printf("my ");   ****/
printf("dog");
printf(" 'Milly'"); //* I really do! */
```

```
I dog 'Milly'
```

—— Solution ——

```
printf("I "); /****/
                    printf("love ");
                    printf("my ");   /****/
printf("dog");
printf(" 'Milly'"); //* I really do! */
```

```
I love my dog 'Milly'
```

5 DONE Comment program

Write short comments in the program 1 to explain what each part of the program does (guess if you don't know for sure) - comment at least whenever you see `WRITE COMMENT HERE`.

```
#include <stdio.h> // WRITE COMMENT HERE

int main (void) // WRITE COMMENT HERE
{
  /*
    WRITE COMMENT HERE */
```

```
  int i = 1;

  // WRITE COMMENT HERE
  printf("%d\n", i);

  return 0;
}
```

```
1
```

The program contains so-called *inline* comments `//` and *multi-line* comments: `/* ... */`.

—— SOLUTION ——

```
#include <stdio.h> // include file stdio.h

int main (void)  // define main function (no arguments)
{
  /* declare integer variable
     and initialize it with the value 1 */
  int i = 1;

  // print variable value
  printf("%d\n", i);

  return 0;
}
```

```
1
```

6 DONE Variable types and declarations

1. Create a named C code block pgm:declarations 1 below.
2. Declare two *floating-point* variables `fahrenheit` and `celsius`.
3. Use two separate statements.
4. Put `:results silent` in the code block header[1].
5. Run the code block (`C-c C-c`).

—— PUT CODE BLOCK HERE ——

—— SOLUTION ——

```
float fahrenheit;
float celsius;
```

7 DONE Fix the program

1. A couple of things are wrong in the code block 1.
2. You can check that yourself by running it (`C-c C-c`) and reading the compiler messages that open in another buffer. Type `C-x 1` to delete the message buffer.
3. Find and fix the errors, and run the code block to make sure.

```
freezing_point = 32.0f
  float freezing_point;
```

8 DONE Variable assignments

1. Create a code block 1 below.
2. Declare **and** initialize two *floating-point* variables, `freezing` and `factor`, with the values 32 and 5/9, respectively.
3. Declare and initialize these variables in **one** statement only.

—— PUT CODE BLOCK HERE ——

—— SOLUTION ——

```
float freezing = 32.0f, factor = 5.0f/9.0f;
```

9 DONE Variable computations

1. The code from 1 and from 1 has been copied into the code block 1 below[2].
2. Complete 1 with two statements:
   - assign the temperature 80 to `fahrenheit`
   - compute `celsius` using 1
3. Run the program to make sure that the answer is correct for 80 degrees Fahrenheit (equivalent to 26.7 degrees Celsius).

```
celsius = (fahrenheit - freezing) * factor
```

```
float fahrenheit;
float celsius;
float freezing = 32.0f, factor = 5.0f/9.0f;
...
...
printf("Fahrenheit: %g\nCelsius equivalent: %.1f\n",
       fahrenheit, celsius);
```

—— SOLUTION ——

```
float fahrenheit;
float celsius;
float freezing = 32.0f, factor = 5.0f/9.0f;
fahrenheit = 80.f;
celsius = (fahrenheit - freezing) * factor;
printf("Fahrenheit: %g\nCelsius equivalent: %.1f\n",
       fahrenheit, celsius);
```

```
Fahrenheit: 80
Celsius equivalent: 26.7
```

10 DONE Fix the program

The program 1 declares and initializes the variable `i` with the value `0`. After assigning `1` to `i`, it should print out `1` but it prints `0` instead.

Fix the error and then run the block with `C-c C-c` to check.

```
int i = 0;
i == 1;
printf("%d\n", i);
```

```
0
```

```
0
```

—— SOLUTION ——

```
int i = 0;
i = 1;
printf("%d\n", i);
```

```
1
```

11 DONE Formatting printout

1. Define and initialize three variables in a code block named 1:
   - an integer variable `foo` with value 100
   - a floating-point variable `bar` with value 100
   - a character variable `baz` with value A
2. Print the three variables so that the output looks like shown below.
3. Use
   - `puts` for the headline "Three variables",
   - `printf` to print `foo` and `bar`, and
   - `putchar` to print `baz`.

   Tip: The final program 1 has 7 lines.

   Output:

   ```
   Three variables:
   foo = 100
   bar = 100.01
   baz = A
   ```

—— PUT CODE BLOCK HERE ——

—— SOLUTION ——

```
int foo   = 100;
float bar = 100.01f;
char baz  = 'A';

puts("Three variables:");
printf("foo = %d\nbar = %.2f\n", foo, bar);
printf("baz = ");
putchar(baz);
```

```
Three variables:
foo = 100
bar = 100.01
baz = A
```

```
Three variables:
foo = 100
bar = 100.01
baz = A
```

12 DONE Fix the program

The program 1 should print out

```
Speed of light (m/s): c = 299792458
Euler number: e = 2.7183
```

But instead it print out this:

```
Speed of light (m/s): c = 14.985029
Euler number: e = 0
```

Fix the program to get the right output!

```
int c = 299792458;
float e = 2.718282f;

printf("Speed of light (m/s): c = %f\n", c);
printf("Euler number: e = %d\n", e);
```

```
Speed of light (m/s): c = 0.000000
Euler number: e = -1610612736
```

—— SOLUTION ——

```
int c = 299792458;
float e = 2.718282f;
```

```
printf("Speed of light (m/s): c = %d\n", c);
printf("Euler number: e = %.4f\n", e);
```

```
Speed of light (m/s): c = 299792458
Euler number: e = 2.7183
```

13 DONE Constants

1. Create a C code block named 1 with three different constant definitions.
2. Define the Arkansas sales tax rate (6.5%) as `SALES_TAX_AR` using the `#define` pre-processor macro.
3. Define the Euler number using `M_E` in `math.h`, and call it `EULER`.
4. Define the speed of light as `SPEED_OF_LIGHT` using `const`.

5. Print all three definitions to get the output:

```
The Euler number is: e = 2.7182818285
The AR sales tax is: 6.5%
The speed of light is: 299792458 m/s
```

—— PUT CODE BLOCK HERE ——

—— SOLUTION ——

```
// Included libraries
#include <math.h>

// Constant declarations
#define EULER M_E // Euler number
#define SALES_TAX_AR 6.5f // AR sales tax
const int SPEED_OF_LIGHT = 299792458; // speed of light

// Print out
printf("The Euler number is: e = %.10f\n", EULER);
printf("The AR sales tax is: %.1f%%\n",     SALES_TAX_AR);
printf("The speed of light is: %d m/s\n",  SPEED_OF_LIGHT);
```

```
The Euler number is: e = 2.7182818285
The AR sales tax is: 6.5
The speed of light is: 299792458 m/s
```

```
The Euler number is: e = 2.7182818285
The AR sales tax is: 6.5%
The speed of light is: 299792458 m/s
```

14 DONE Standard math library

Open the file `/usr/include/math.h` and search for the definition of `M_PI`. What is the last non-zero digit?

—— SOLUTION ——

The answer is 6.

```
# define M_PI           3.14159265358979323846  /* pi */
```

15 DONE Reading input

1. Copy the code block 1 below into a code block 1
2. Modify 1 so that it reads a floating-point variable `x` instead of an integer variable `i`.
3. The *format specifier* for `float` numbers is `%f`.
4. Create an input file named `finput` in `$PWD` and put the number `3.141593` into it.
5. Run 1

```
int i;
puts("Enter an integer!");
scanf("%d", &i);
printf("You entered %d\n", i);
```

```
Enter an integer!
You entered 5
```

—— SOLUTION ——

```
float x;
puts("Enter a floating-point number!");
scanf("%f", &x);
printf("You entered %f\n", x);
```

```
Enter a floating-point number!
You entered 3.141593
```

```
Enter a floating-point number!
You entered 3.141593
```

16 DONE Naming identifiers

Naming conventions dictate that you should use

- upper case letters for constants
- lower case letters for variables and function names
- separate names with underscore or insert capital letters
- name according to function
- In the code block 1, complete the code according to these rules.
- Run the code block with the additional header-argument `:flags -Wall` to see if you get any warnings.

```
// integer constant for the speed of light
const int ... = 299792458;

// floating-point constant for pi
```

```
#define ... 3.141593f

// integer variable for volume computations
int ...

// character variable for last names
char ...

// function that adds two integers i and j
int ... (i,j) {
  return i + j;
}

// variable whose name contains "my", "next", and "birthday"
int ...
```

—— SOLUTION ——

```
// integer constant for the speed of light
const int SPEED_OF_LIGHT = 299792458;

// floating-point constant for pi
#define PI 3.141593f

// integer variable for volume computations
int volume;

// character variable for last names
char lastName;

// function that adds two integers i and j
int add (int i, int j) {
  return i + j;
}

// variable whose name contains "my", "next", and "birthday"
int my_next_birthday;
```

17 DONE Fix the program

The program statements in 1 contain multiple errors. Find them all and fix them if you can so that the program compiles and runs without errors - without simply commenting out erroneous code.

```
int void = 1;

double 10_times;

float _long = 10.45;

char else;

const int ui-1 = 1;

int bottles100 = 100;
```

—— SOLUTION ——

```
int _void = 1; // contained keyword 'void'

double _10_times;

float _long = 10.45;

char Else; // contained keyword 'else'

const int ui_1 = 1; // contained a dash '-'

int bottles100 = 100;
```

18 DONE Program layout

The program 1 does not accommodate program layout conventions (though it will compile and run). Fix that.

Tip: sort the different parts of the program first. The comments might be helpful for that.

The output looks liks this:

```
I'm gonna print a number now.
The number is 100
100*(-1)=-100
```

```
const X=100.;puts("I'm gonna print a number now.");printf("The number is %d\n", X);

// declarations

// computation
int i=-1;int y; y = X * i;printf( // print result of computation
                          "%d*(%d)=%d\n",X,i,y
                          ); // print constant
```

```
I'm gonna print a number now.
The number is 100
100*(-1)=-100
```

—— SOLUTION ——

```
// declarations
const X=100.;
int y;

// print constant
puts("I'm gonna print a number now.");
printf("The number is %d\n", X);

// computation
int i=-1;
y = X * i;

// print result of computation
printf("%d*(%d)=%d\n",X,i,y);
```

```
I'm gonna print a number now.
The number is 100
100*(-1)=-100
```

19 DONE Fix the program

The program [1] violates layout standards and will not compile. Fix it and run it - the correct output is: `1 is not 2`.

```
#define
ONE 1
#define
TWO 2
printf("%d is not %d\n", ONE, TWO);
```

—— SOLUTION ——

```
#define ONE 1
#define TWO 2
printf("%d is not %d\n", ONE, TWO);
```

```
1 is not 2
```

Footnotes:

[1] With `:results silent` in the header, the Org-mode code block will be executed, but the results will not be printed in the buffer, only in the minibuffer. If there is no printout, the minibuffer shows `""` (empty).

[2] The header argument `:noweb` enables referencing to other code. Setting it to `yes` means that references are expanded when evaluating, tangling, or exporting. You can check that by tangling the source code and looking at the result (more info).

Author: [yourName] (pledged)
Created: 2022-05-27 Fri 10:51