CC Class Notes

Followup class notes for CSC100 Intro to Programming in C/C++ Spring 2022

Table of Contents

1 README

Instead of bugging you with emails, I opt to summarize my course observations regarding content, process, in this file. These often contain additional links, articles, and musings.

I usually update it after each class - it also contains the homework (if any). The first point of call for any questions should be the FAQ. There are two FAQs - a general one (for all my courses), and a FAQ for CSC 100.

You find the whiteboard photos here in GDrive.

The companion file to this file, with the agenda and much of the course content, is the agenda.org file.

Please note that all inline references like (King, 2011) are linked within the file but will not render in GitHub (i.e. you'll end up on a 404 page). You find all inline references completed at the end of the file.

2 Welcome to the course - w1s1 (01/12/22)

2.1 Homework (until Friday, 14 Jan)

(See Schoology assignment)

- Create a GitHub account
- Complete the GitHub Hello World exercise
- Submit an issue to the `cc100` GitHub repo to confirm completion.

Let me know if you encounter any problems. This shouldn't take more than 20 minutes of your time. Bring any questions that you may have with you to class!

If you are in another one of my courses, you don't have to do the exercise again, but you should submit an issue for every course that you attend. When I receive the issue, I will close it and give you the points!

Purpose of this exercise:

- Engage with the GitHub software development platform
- Understand how to submit issues (aka corrections, suggestions etc.)
- Understand how software developers work in the cloud using the Git version control software

2.2 Stuff

2.2.1 Finding your operating system

Victor: On Windows, search for `dxdiag` ("DirectX Diagnostic Tool"). This is actually quite a large hammer for a small nail. On any computer, you should be able to enter `About` in the search field and get at least some information about your operating system (OS).

The operating system is software on your computer that enables it to run other programs. You may imagine it as a system of roads to get you where you want to go.

2.2.2 Books

I showed a few books that explain different aspects of C - because so many devices contain C! All of these are interesting, none are necessary (or even recommended) for beginners, and all are brand new - demonstrating the continued interest in C.

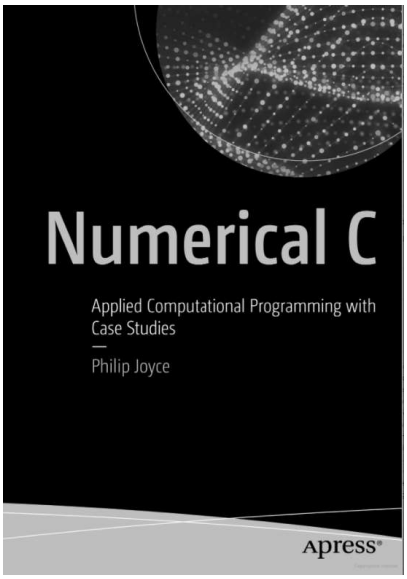

Figure 1: Numerical C by Philip Joyce (Apress, 2019)

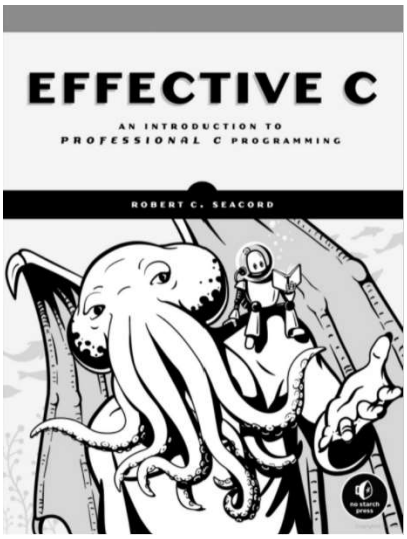

Figure 2: Effective C by Robert C Seacord (NoStarch, 2020)

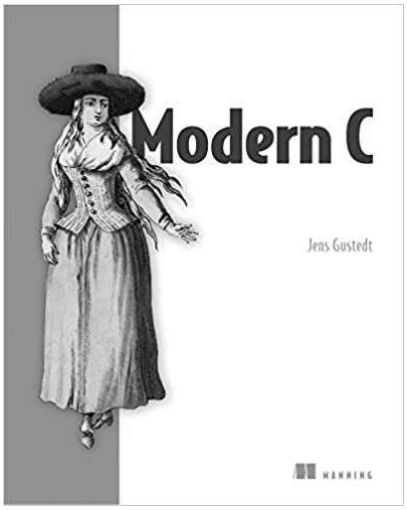

Figure 3: Modern C by Jens Gustedt (Manning, 2020)

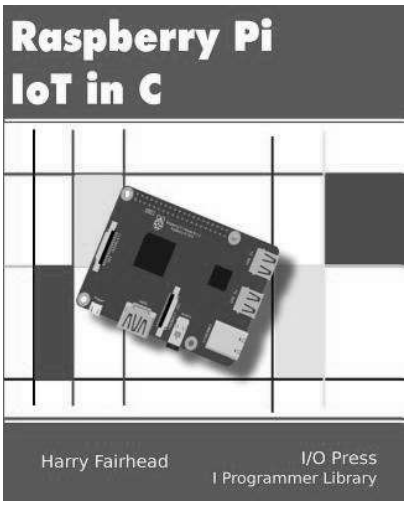

Figure 4: Raspberry Pi IoT in C by Henry Fairhead (I/O Press, 2016)

2.2.3 Wrapper

This term refers to the fact that not all software runs on all operating systems (OS). One way of running "alien" software on your OS is to "wrap" it - same thing that a (bacterial) vaccine does: your OS then thinks it deals with something it knows.

My examples (for Windows only) were:

1. "cygwin", which is a bundle of Linux services, wrapped up to run under Windows. Cool app, actually. See here for the easy download and installation (just download and run the file `setup-x86_64.exe`. It's one quick way of getting a C compiler - a program that can translate your C programs so that they run on your computer.

2. "ubuntu linux app" - This is a wrapper for a complete Linux system for Windows. See this FAQ for how to install it on your computer (worth doing). It's another way to get a C compiler.

2.2.4 (Software) library of functions

Figure 5: Library (Source: Unsplash)

2.2.5 Command-line interface

Aka CLI, terminal, command line, or shell. Linux nerds and hackers thrive on the command line, and so will you, via this course.

Some live "entirely in a terminal" (Lunduke, 2019).

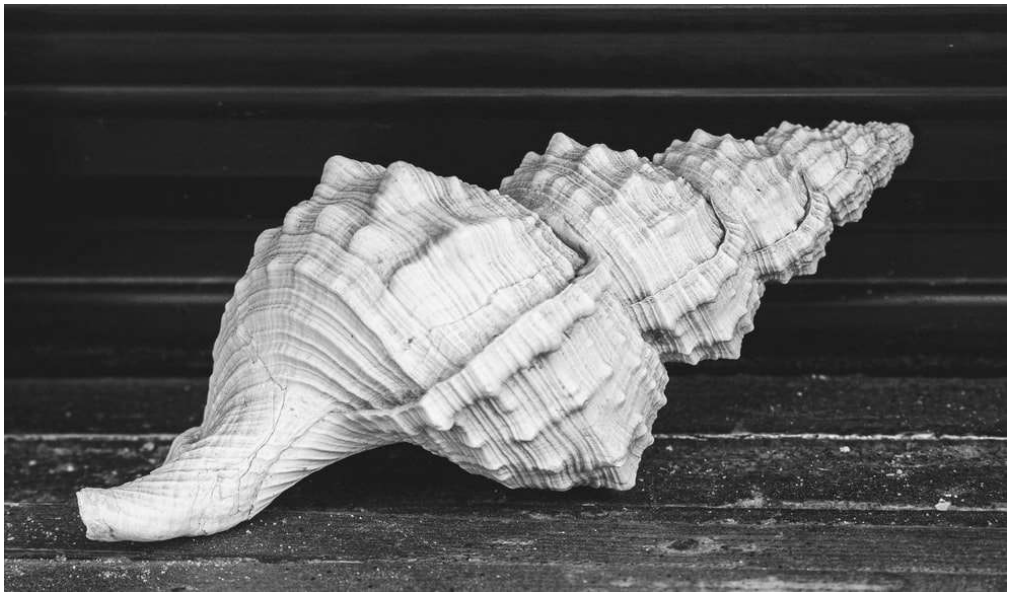

Figure 6: the other kind of shell (Source: Unsplash)

2.2.6 Literate programming = doc + code + output

There isn't that much more to it. We'll use GNU Emacs Org-mode as interactive notebook environment, and also to develop, run and debug C code.

You can find more information about literate programming, and plenty of links at Wikipedia.

I have just made a screencast to show the difference between shell and notebooks, not using C but the database query language SQLite instead (Birkenkrahe, 2022). Check it out.

3 Git, GitHub, History of C - w1s2 (01/14/22)

- TIOBE = "The Importance of Being Earnest" - play by Oscar Wilde (1895).
- Weekly quiz: review of last week's content. After the first play, the quiz will be opened for unlimited playing and can be used to prepare for the final exam (whose questions are sourced from the quiz).
- Algorithms vs heuristics - heuristics is a human behavioral domain and leads to more resilient processes.
- The "kernel" is the core of an operating system. E.g. the Linux kernel.
- COBOL is a compiled language used mainly in finance and business since the late 1950s. It mainly runs on mainframes (very large computers). To be able to understand, and fix COBOL programs has become a rare skill since the language is rarely taught. Here is a free tutorial.

4 Introduction to C and C++ - w2s3 (01/19/22)

4.1 Quiz feedback discussion

- Timing - 15 OK
- Content - no questions
- Schedule - weekly quiz as a recap of last week
- Grading - unlimited attempts after the first play (in class)

4.2 Captain's Log

- If you miss a class and would like it recorded, notify me beforehand. I will do my best to create a recording, which will then be uploaded to GDrive (link in Schoology)
- Whiteboard screenshots taken after class will be uploaded in GDrive as well (link in Schoology). You can submit GitHub issues if you have questions, additions, etc
- If you miss a class, check content planned/covered through the files `agenda.org` and `notes.org` in the GitHub repo.
- You can add your GitHub skills to your resume and it might help you when you try to get a job or an internship!
- This class will be offered in the Summer I program (June 2022). Same content and format, and likely better than this first version (because I'll have learnt how to do it better).
- My challenge: why should a liberal arts major (e.g. political science) learn to program, and especially in C?
    - "Meet The Newest Liberal Art: Coding" (Koenig, 2020)
    - Greater employability is a general, traditional reason
    - Coding can be fun. Kids know this.

- Data and computer literacy is a life and a professional skill. Especially for political science/economics, see the Economist newletter "Off the charts" (Segger, 2022).

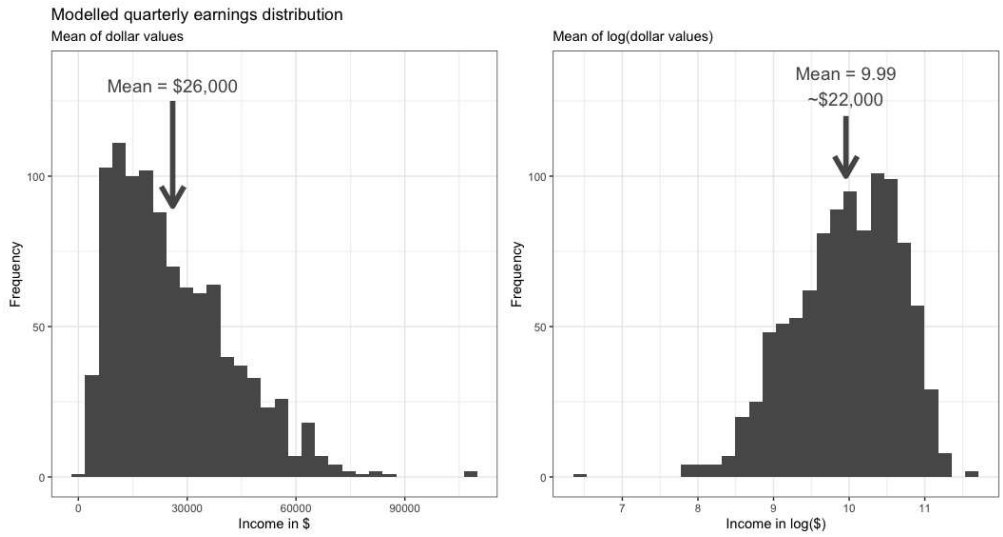

Figure 7: Plots from The Economist

- Object-Oriented Programming (OOP) is a specialty of C++ over C. Other modern languages share this paradigm (which is conceptually quite difficult): Java, Python, R, are examples. OOP is useful in the world especially for **reuse** of code, and it makes maintaining large code repositories easier. Our introduction to C++ at the end of the introductory class will include a short tour of OOP, too.
- Glossaries are like dictionaries of important words. Knowing words and definitions can help you learn new things much more quickly and easily. I know whereof I speak - at this stage of my life, I learn new things really quickly because I have such a large - not necessarily deep - foundation of technical terms (and thereby, concepts). Don't neglect your learning of (new) words!

5 Install C compiler, set PATH - w2s4 (01/21/22)

Figure 8: GCC logo

- Installed GCC, the compiler. For instructions, see here.
- After the installation, you have to set the `PATH` to the compiler so that it can be found.
- You can test the successful installation in the terminal (Windows CMD prompt) with the command `gcc --version`.
- The documentation to this important, large (with > 10 mio lines of source code) software program is massive (PDF - Stallman 2022), though there are also shorter introductions, if you're interested, like this one (Gough, 2014).

6 Install GNU Emacs - w3s5 (01/24/22)

What you learnt using the instructions from tutor.org:

- Open, shut or suspend Emacs
- Open the Emacs tutorial

- Read a file into Emacs

  These are the skills you need to complete the assignment (for extra credit until Wednesday 11 AM - for all others we'll do this in class).

  Here is the Emacs documentation on the initialization file `.emacs` in the GNU Emacs manual: "How Emacs finds your init file".

7 Create GNU Emacs Org-mode file - w3s6 (01/26/22)

- Fixed the `.emacs` problem - this file must be located in the directory that you find in Emacs with `~/`. (See instruction.)
- Started the Org-mode exercise. Will finish it on Friday. Then you can run C code inside Emacs as an Integrated Development Environment (IDE).

8 GNU Emacs initialization file - w3s7 (01/28/22)

- Explanation of the Org-mode C code block header arguments:

HEADER ARGUMENT	MEANING
`:main yes`	wrap `puts` statement in `int main(void) { }` function
`:includes stdio.h`	include standard input/output header file
`:tangle first.c`	export source code as C file `first.c` ("tangle")
`:exports both`	both result and source code will be exported
`:comments both`	link source code and org files, add comments to source
`:results raw`	insert output directly in org format into org file

- Something you may have noticed in the GitHub issue submissions of your resulting Org-mode file: the tilde (~) around a word strikes it through on GitHub. On GitHub, `…` is the syntax for code (in HTML, this is <code> … </code>.

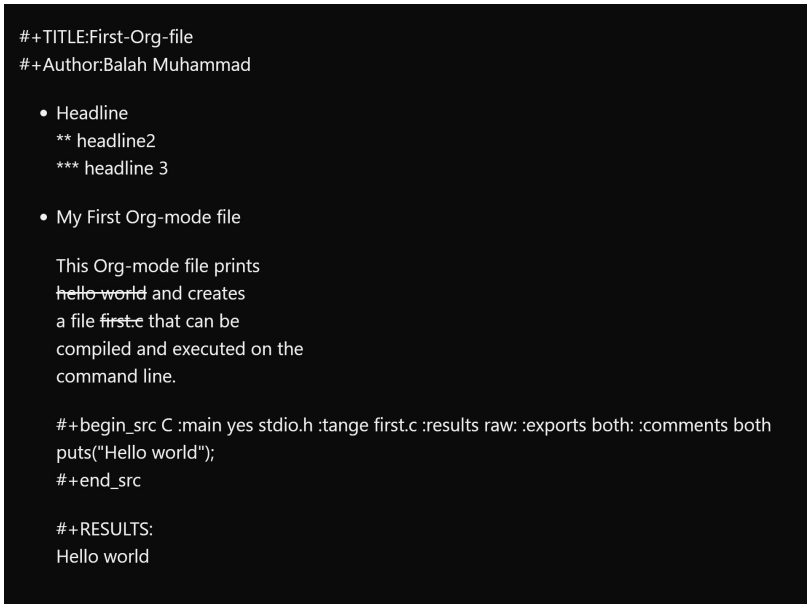

Figure 9: screenshot from a GitHub Org-mode file submission

- Feel free to bring your own laptop to future sessions. If you want me to check installation because something did not work, come a little earlier or stay a little later.
- This concludes our "Emacs week". To get more practice in GNU Emacs, complete the onboard tutorial (`C-h t`).
- Solutions to the Org-mode assignment are posted here on GitHub. Note that submissions of programs as Org-mode files should always also be accompanied by references and sources.
- I told you an inaccuracy in class: when rendering the Org-mode file on GitHub, the `#+TITLE` meta information is displayed as the title of the file. If no such header is present, only the `README` file is displayed (with the file name as title).

9 GCC lab session - w4s8 (01/31/22)

Lab session - see whiteboards.

10 Structure of a C program - w4s9 (02/02/22)

10.1 Compiler flags

- Compilers come with a lot of options (check `gcc --help`)
- For the assignment, add `:flags -Wall` to the header (`gcc` option)[1]

10.2 The `pun.c` program

Check the 10.4 for details. The output, by the way, does not look good on GitHub.

```
printf("To C, or not to C, that is the question.\n");
```

To C, or not to C, that is the question. To C, or not to C, that is the question.

```
#include <stdio.h>

int main(void)
{
  printf("To C, or not to C, that is the question.\n");
  return 0;
}
```

To C, or not to C, that is the question. To C, or not to C, that is the question.

```
printf("To C, or not to C,\n that is the question.\n");
```

To C, or not to C, that is the question. To C, or not to C, that is the question.

```
printf("O         O\n");
printf("     | \n");
printf("\n");
printf(" *       * \n");
printf("   * * * \n");
```

O O

- *
    - * *

O O

- *
    - * *

10.3 Comments

```
/* Comment on a line */
// Another type of comment on a line
printf("Nothing to see here.\n");  // on a statement line

/* You can also comment
   over several lines */
```

Nothing to see here.

10.4 Glossary

TERM	EXPLANATION
C program	Directive + `main` function + statements
Directive	Pre-processor, e.g. `#include` + header
Header	Standard library functions, e.g. `stdio.h`

TERM	EXPLANATION
Function	Named series of statements for execution
Library function	Pre-defined C function, e.g. for input/output
Return value	Returned value computed by a function
`main` function	Mandatory wrapper function, called automatically
Status code	Indicates program termination status, e.g. `0`
String literal	List of characters in quotes, e.g. `"Hello world"`
End symbol	Statements must have it at the end, in C: `;`
New-line character	`\n` moves the display by one line

11 Variables and assignments - w5s10 (02/07/22)

- Difference between the output functions `puts` ("put string") and `printf` ("print file"): `puts` automatically adds a newline character (`\n`) at the end of the string - for `printf`, you have to add the character, if you want to move to the next line.
- Purpose of the Org-mode header arguments? Emacs automatically inserts
  - `main` function (`:main yes`)
  - Standard library files as directives (`#inlude <stdio.h>`)
  - Compiler flags (`-Wall` for all warnings)
- Why do types have to be declared? So that the computer can manage its memory better - volatile memory (RAM) is expensive and small (compared to non-volatile memory, like a hard disk)

12 C constants, naming conventions - w6s13 (02/16/22)

- "Warming Up: be the compiler!" - The solution to fix the output was to change the format specifiers for the floating point variables from `%d` to `%f`, and to change the type declaration for `tax` from `int` to `float` (since it's coerced into a `float` through the computing statement.
- The follow-up question was: how can we get rid of the 6 default digits and print only two digits. The answer: use `%.2f` with precision `p=2`.
- Computers are unforgiving when it comes to neglect. It's like talking to someone who doesn't understand you at all if you cannot spell perfectly.
- When a program does not run, use classic compsci heuristics:
  1. Try to understand the error message from Emacs or gcc.
  2. If the compiler gcc complains: check the syntax - Emacs helps you with Syntax highlighting (you might want to change the Emacs theme for that with `M-x custom-themes`).
  3. Close the program and reopen it
  4. Close Emacs and reopen it
  5. Shut down the computer and restart it

13 Test review - w6s12 (02/14/22)

13.0.1 Known quiz questions

- WHERE DOES C COME FROM

  UNIX

- WHICH OF THE FOLLOWING ARE STRENGTHS OF THE C PROGRAMMING LANGUAGE?
  - Portability - runs on any computer
  - Speed and efficiency - clever compilers
  - Standard library - like `stdio.h` for input/output

- WHAT DO YOU NEED TO CREATE AND RUN A C PROGRAM?

WHAT	WHY
An editor (like Emacs)	To write the source code (.c)
An operating system (like Windows)	To run the edior, the compiler, and display (or print) the results
A compiler + linker (like GCC)	To compile the source code to object code, and link it to libraries (e.g. stdio.h for input/output) to produce executable machine code
A hosting platform (like GitHub)	Cloud application - nice to have if you collaborate at a distance, not needed
A learning management system (like Schoology)	Only if you want to learn and teach, not for programming

- C IS AN OBJECT-ORIENTED PROGRAMMING LANGUAGE
  - NO
  - C++, C#, Java, R, Python etc. are OOP languages
  - C has no notion of a class that allows to define subclasses, inheriting functions/methods etc.

- KNOW YOUR CODE - MATCH THE TERM AND THE DEFINITION!

QUESTION	ANSWER	EXAMPLE
Source code	Human-readable program	e.g. a program `hello.c`
Machine code	Program ready for execution	after compiling and linking
Object code	Code produced by a compiler	only `main()`, but unable to produce output

- WHAT IS GCC? TIP: WE INSTALLED GCC ON WINDOWS AS MINGW.
  - A bundle of compilers for many different languages
  - Includes `gcc` that we use on Windows, Linux and MacOS[2]

- MATCH THE EMACS ORG-MODE HEADER ARGUMENT AND THE MEANING

  - Emacs Org-mode is an application inside Emacs that allows you to run C programs without having to explicitly invoke GCC
  - To make the magic happen, Org-mode parses the header arguments and substitutes them in the code that GCC gets from Emacs
  - When you tangle a file, you can see the result of this substitution, and the tangled file can be compiled and run on the command line

HEADER-ARGS:C	MEANING
`:main yes`	wrap `puts` statement in `int main(void) { }` function
`:includes stdio.h`	include standard input/output header file
`:tangle first.c`	export source code as C file `first.c` ("tangle")

HEADER-ARGS:C	MEANING
`:exports both`	both result and source code will be exported
`:comments both`	link source code and org files, add comments to source
`:results raw`	insert output directly in org format into org file

```
#+name: C example
#+begin_src [language] [header-args]
  [statements]
#+end_src
```

13.0.2 MATCH PROGRAM ELEMENT AND DEFINITION!

- The directive has to be at the top of the program
- The `main` function has to be there (there can be other functions, but it has to run first)
- The statement is not necessary but it's what is executed

C SOURCE CODE	MEANING
`#include <stdio.h>`	Directive (input/output library)
`int main(void) { ... }`	Main function (without input argument)
`printf("hello\n");`	Program statement (screen display)

- Here is a function definition outside of `main()` that does the same job as our hello world program - print out `hello world`. To demonstrate the relationship between our own function `hello()` and `main()`, I am not using any `header-args` in the block 1.

```
#include <stdio.h>
hello() {
  printf("hello function world\n");
}
int main(void){
  hello();
  return 0;
}
```

```
hello function world
```

- Which of these are NOT valid PATH settings?
- The "binary" of a program (like gcc) is machine code that runs on any type of computer.
- Which symbols mark Org-mode metadata?
  - #+
  - * *
  - //

13.0.3 WHICH OF THESE ARE VALID TYPE DECLARATIONS?

Tip: once a type is declared, it can be used in a program statement to be assigned values.

TRUE:

- `int height;`
- `float profit;`

FALSE:

- `int profit`
- `height = 8;`

13.0.4 The name `main` in a C program is critical - it can't be `MAIN` or `start`

TRUE

Feedback: Computers, and their languages, are very picky. That's why computer languages are called "formal" rather than "natural". Violation of the rules means (usually) instant program failure. The spelling of keywords in C like `main` (or `int`, or `printf`) or the demand for a semicolon after each statement, are examples.

13.0.5 Which of these terminal commands will compile the file `hello.c` and create an executable file named `hello`

TRUE:

- `gcc -o hello hello.c`

FALSE:

- `gcc hello.c`
- `compile hello.c -out hello`
- `gcc -o hello.c hello`

13.0.6 How can you tell the compiler to warn you if something's not quite right with your source code?

TRUE:

- gcc -Wall

FALSE:

- gcc –help
- gcc –version
- gcc –target-help

13.0.7 `:includes` is for Org-mode, `#include` is for the C compiler preprocessor

TRUE

Feedback: :includes and #include both are followed by the library or libraries (usually more than one) that you wish to use in your code. E.g. adding stdio.h enables you to print output and get input.

13.0.8 The C function `puts` needs a newline character `\n` to display the next line

FALSE

Feedback: puts stands for "put string" and that's what this function does. It does include a newline character unlike printf (= formatted print).

13.0.9 If the variable `height` is declared an `int`, what's wrong with the following `printf` statement?

```
printf("My height is %f", height);
```

TRUE:

- The format specifier should be `%d`

FALSE:

- The format specifier should be `.2f`
- There must be a newline character `\n` after `%f`
- The statement does not need a semicolon (;) at the end

13.0.10 When you print a variable with the wrong format, you get unpredictable numerical results

TRUE

Feedback: I demonstrated this in class (in `variables.org`). If you are in doubt, make the test yourself: declare an integer variable and print it with printf using the %f format specifier.

13.0.11 You want to display `x = 234.5895484` with 3-digit accuracy after the decimal point. Which formatting specifier is correct?

TRUE:

- `printf("%.3f", x);`

FALSE

- `printf("%.pf", x);`
- `printf("%.3d", x);`
- `printf("%3f", x);`

Feedback: As Victor noticed, `printf("%.pf", x);` does not result in an error but in a strangely formatted output (without the number).

14 C constants, naming - w6s15 (02/18/22)

- What's wrong with this program? (It won't even compile.) **ANSWER**: there shouldn't be a semi-colon in the definition of the constant `CONST_` with the preprocessor (using `#define`).

```
#include <stdio.h>
#define CONST_ 100;

int main(void) {
  int value = CONST_, newValue;

  newValue = value + 1;

  printf("%d %d %d\n", CONST_, value, newValue);
```

```
  return 0;
}
```

15 Program assignment review, scanf - w7s16 (02/21/22)

15.1 Compute the volume of a sphere

- This is a fairly long set of solutions - I'm also explaining how to tackle reading input in Org-mode, which you don't need to know.
- Below is also the solution to the extra credit problem (reading the radius input from the commandline).
- Included are also a couple of exhibits with small mistakes - anonymized student submissions.

15.1.1 Problem

- Write a program that computes the volume of a sphere with a 10-meter radius, using the formula $v = 4/3 \times \pi r^3$.
- Write the fraction $4/3$ as `4.0f/3.0f`. (Try writing it as 4/3 and see what happens.)
- Remember that C does not have an exponentiation operator, so you need to write r^3 as `r*r*r`.
- Upload your solution program as a `.c` file or as a `.org` file to Schoology not later than 11 AM on Monday, February 21st. Make sure that your program actually runs without errors!
- Be prepared to present your solution in class.

15.1.2 Solutions and pseudo solutions

- In the first solution, we define the constant as a macro using `#define`.

```
#define PI_CONST 3.141593 // macro declaration

float volume, radius=10.0f; // type declaration

volume = 4.0f/3.0f * PI_CONST * radius * radius * radius; // statement

printf("The volume of a sphere of radius %.2f is %.2f\n",
       radius, volume); // display output
```

```
The volume of a sphere of radius 10.00 is 4188.79
```

- What happens when you use `4/3` instead of `4.0f/3.0f` in the formula?

```
#define PI_CONST 3.141593 // macro declaration

float volume, radius=10.0f; // type declaration

volume = 4/3 * PI_CONST * radius * radius * radius; // statement

printf("The volume of a sphere of radius %.2f is %.2f\n",
       radius, volume); // display output
```

```
The volume of a sphere of radius 10.00 is 3141.59
```

- What happens when you use integers instead of floating point numbers?

```
#define PI_CONST 3.141593 // macro declaration

int volume, radius=10; // type declaration

volume = 4.0f/3.0f * PI_CONST * radius * radius * radius; // statement

printf("The volume of a sphere of radius %d is %d\n",
       radius, volume); // display output
```

```
The volume of a sphere of radius 10 is 4188
```

- What happens when you use a less accurate PI constant?

```
#define PI_CONST 3.14 // macro declaration

int volume, radius=10; // type declaration

volume = 4.0f/3.0f * PI_CONST * radius * radius * radius; // statement

printf("The volume of a sphere of radius %d is %d\n",
       radius, volume); // display output
```

```
The volume of a sphere of radius 10 is 4186
```

- In the second solution, we include the constant using the math library `math.h`.

```
#include <math.h>  // include math library with PI=M_PI

float volume, radius=10.0f; // type declaration

volume = 4.0f/3.0f * M_PI * radius * radius * radius; // statement

printf("The volume of a sphere of radius %.2f is %.2f\n",
       radius, volume); // display output
```

```
The volume of a sphere of radius 10.00 is 4188.79
```

15.1.3 Reading input

- For the expanded problem, we use the input function `scanf` for the `radius`. Unfortunately, Emacs Org-mode will not wait for us to enter the input. You have to tangle the code block, compile and run the C file outside of Emacs, on the CMD shell (terminal).

- In the code below, I'm using a trick - the Org-mode header argument `cmdline` will accept input from a file. This file, `in.txt`, contains my "interactive" input.

```
#include <math.h> // include math library
```

```
float volume, radius; // declare types

printf("Enter radius of sphere: \n");  // ask for input
scanf("%f", &radius);                  // read input

volume = (4.0f / 3.0f) * M_PI * radius * radius * radius; // statement

printf("Volume (cubic meters): %.1f\n", volume); // display result
```

- To pass arguments to source code blocks, use `var` (cp. manual).

```
int radius = INPUT;
printf("Input: %d", radius);
```

```
Input: 10
```

- We can use this for a better solution: this code block can be run again and again like the compiled program, and we enter the variable `radius` as header `:var RADIUS`.

```
#include <math.h> // include math library

float volume, radius; // declare types

radius = RADIUS;

volume = (4.0f / 3.0f) * M_PI * radius * radius * radius; // statement

printf("Volume (cubic meters): %.1f\n", volume); // display result
```

```
Volume (cubic meters): 7238.2
```

- See also:
  - stackexchange (2018)
  - stackexchange (2015)
  - stackoverflow (2017)

15.1.4 Volume Assignment review (class notes)

- Exhibit 1

```
#include <stdio.h>

int main(void) {

  int radius, volume;
  radius = 10;
  volume = 4.0f/3.0f * 3.14 * radius * radius * radius;

  printf("Dimensions: %d\n", radius);
  printf("Volume (cubic meters): %d\n", volume);

  return 0;
}
```

Dimensions: 10 Volume (cubic meters): 4186 Dimensions: 10 Volume (cubic meters): 4186

- Exhibit 2

```
#define PI 3.14;
int main()
{
  int r = 10;
  int v = 4.0f/3.0f * 3.141592 * r*r*r;

  printf("volume = %d",v);
}
```

```
volume = 4188
```

15.2 Hello world function

- The first solution does not show `main()` because it's taken care of by Emacs. `hello()` is defined inside `main()`.

```
// function definition
int hello(void) {
  puts("hello world");
  return 0;
}
// function call
hello();
```

```
hello world
```

- `[X]`

  The second solution is like the first but does show the envelopping `main()` function. For some reason, it does not print out a result, and the compiler warns: "`hello()` defined but not used." What is that about?

```
#include <stdio.h>

int main(void) {
  int hello(void) {
    puts("hello world");
    return 0;
  }
  return 0;
}
// function call
hello();
```

- The answer to the last question is: the `hello()` function cannot be called outside of `main()`. If you move it back before the `return` statement, it works.

```
#include <stdio.h>

int main(void) {
  int hello(void) {
    puts("hello world");
    return 0;
  }
  // function call
  hello();
  return 0;
}
```

```
hello world
```

- A very short solution just for fun.

```
int hello() {puts("hello world");return 0;} hello();
```

```
hello world
```

15.3 Be the compiler - solution

- `int` is not an allowed name for a variable (protected keyword)
- The string is not followed by a comma but by a semi-colon
- The variables in `printf` are not recognized

```
float int1=1.f, int2=2.f, int3;
printf("%.2f\n", int3 = int1 + int2);
```

```
3.0
```

16 Compiling, LitProg, Program Layout - w7s19 (03/04/22)

- GCC has a long UNIX manual page, which contains all options for the command line.
- You can add packages to Emacs using its package manager with `M-x package-list-packages`. In the `*Packages*` buffer, add a package with `i`, install it with `x`, update packages with `U x`.

- For my notes and presentations, I use `org-beautify-theme` and `org-bullets`, two layout themes. For `org-bullets` to run, you need to put the following code in your `.emacs` initialization file.

```
(require 'org-bullets)
(add-hook 'org-mode-hook (lambda () (org-bullets-mode 1)))
```

17 Switch, case, break, pgm 7, pgm 8 - w11s26 (04/01/22)

17.1 How do you attack a (non-trivial) programming exercise?

Victor's process:

1. What type of variables do I need to declare?
2. Which control flow would I need? (if, switch)
3. Start coding!

Gaven's process:

1. Read problem carefully
2. Show what should come out
3. What is input and what is output
4. Decide variables…and continue with Victor's process

18 References

- Birkenkrahe (Jan 11, 2022). Interactive shell vs. interactive notebook (literate programming demo). URL: youtu.be/8HJGz3IYoHI.
- Gough (2004). An Introduction to GCC. URL: tfetimes.com.
- Koenig (Feb 4, 2020). Meet The Newest Liberal Art: Coding [blog]. URL: www.edsurge.com.
- Lunduke (June 28,2019). Without a GUI–How to Live Entirely in a Terminal [article]. URL: linuxjournal.com.
- Segger(Jan 18, 2022). Off the Charts - The best of our data journalism. URL: economist.com.
- Stallman et al (2022). Using the GNU Compiler Collection. URL: gcc.gnu.org.

Footnotes:

[1] From the GCC reference manual (Stallman et al, 2003):

> `-Wall` enables all the warnings about C constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning).

[2] Suggestion to install GCC under MacOS here - have not tried this yet myself. Perhaps you could, if you have a Mac, and let me know? (1) First, you need to install HomeBrew from https://brew.sh/, (2) Then, run `brew install gcc` in the terminal. (3) Type `g++-11` in the terminal to see if successful. (4) Type `sudo ln -s /usr/local/bin/gcc-11 /usr/local/bin/gcc`.

Author: Marcus Birkenkrahe
Created: 2022-04-25 Mon 09:14