Spring 2022 courses

DONE cc quiz 6

Settings

This quiz covers of our lab practice session material of the past 2 weeks. Rules of the game:

- A question has only ONE right answer unless otherwise noted
- After the first play, the quiz will be opened for unlimited play
- Let me know if you have any comments or corrections

This is the last quiz before the next test.

What is GNU Emacs?

More than one answer is correct.

TRUE:

- Free, open source software
- A self-extensible text editor
- An integrated development environment (IDE)
- An application written in C and (Emacs) Lisp

What is Org-mode?

More than one answer is correct.

TRUE:

- A plain text Emacs environment
- A tool for authoring documents
- Interactive notebook software

FALSE:

- A programming language

"Tangling" means generating source code from an Org-mode file

TRUE

"Weaving" means running code in an HTML browser

FALSE

Feedback: weaving code means extracting a document from an Org-mode file to document a program. HTML comes in because you can weave the document as HTML and display it in a browser (with C-c C-e h o).

What is good program layout practice?

TRUE:

- Space between tokens makes identification easier
- Indentation makes nesting easier to spot
- Blank lines can divide a program into logical units

FALSE:

- Use a smart coloring scheme to highlight syntax

Feedback: the syntax highlighting depends on the IDE and on the person, and it is not part of the layout.

What's wrong with this layout?

The program won't compile. Can you see what the problems are?

```
#include <stdio.h>  #include <math.h>
int main() { int x
    = M_PI; puts("print me");}
```

More than one answer is correct.

TRUE:

- Only one preprocessor directive per line
- The constant `M_PI` was not declared

FALSE:

- The assignment of `M_PI` to `x` should not be split over two lines
- `return 0;` is missing

Order the steps in this workflow

You have some C code in an Org-mode file `file.org`. You want to run it on the command line.

Order the steps below from 1-7 where 1 is the first and 7 is the last step.

1. Write source code in an Org-mode code block (#+begin_src…#+end_src)
2. Put the argument `:tangle file.c` in the header of the code block
3. Tangle the Org-mode code block with `C-c C-v t` to obtain `file.c`
4. Open a command line terminal (aka shell)
5. Find `file.c` on the command line with `DIR` (Windows) or `ls` (MacOS or eshell)
6. Compile `file.c` on the command line with the command: `gcc -o outfile file.c`
7. Run the executable file named `outfile` by entering at the prompt: outfile.exe or outfile (Windows, eshell), or ./outfile (MacOS, eshell)

Match the `scanf` statement and the corresponding input

Remember: The input needs to be identically matched to the expected `scanf` format.

Here, `i`, `j`, `k`, `l`, `m` are declared as `int`, and `x`, `y`, `u`, `v`, `r`, `s`, `t` are declared as `float`.

scanf("%d, %f, %f", &i, &x, &y);	100, 99.99, -1.5
scanf("%d\\%f%\\%f", &j, &u, &v);	1000\3.14\37.5567
scanf("%d (%d%) %d", &k, &l, &m);	222 (-333) 444
scanf("%f %f%.%f", &r, &s, &t);	3.1 3.14.3.141593

```
int i,j,k,l,m;
float x,y,u,v,r,s,t;

scanf( "%d, %f, %f",   &i, &x, &y);
scanf( "%d\\%f%\\%f",  &j, &u, &v);
scanf( "%d (%d%) %d",  &k, &l, &m);
scanf( "%f %f%=%f",    &r, &s, &t);
printf("%d %g %g\n",   i,  x,  y);
printf("%d %g %g\n",   j,  u,  v);
printf("%d %d %d\n",   k,  l,  m);
printf("%g %g %f\n",   r,  s,  t);
```

Fix the Org-mode code block!

The code block below should print out `pi = 3.14159274`.

I have stored this value of `pi` in a file called `pi`.

But instead the result printed is `pi = 0.00000000`. What's wrong?

Tip: I presume that the problem lies with feeding the input to the `scanf` statement.

```
float pi;
scanf("%f",    &pi);
printf("pi = %.8f\n",  pi);
```

```
#+RESULTS: cmdlineQuiz
: pi = 0.00000000
```

TRUE:

- The header argument should be `:cmdline < pi`

FALSE:

- The header argument is `:cmdline < input`
- The `scanf` formatting string should be `"%.8f"`, too
- The precision of 8 decimal digits is too high

```
float pi;
scanf("%f",    &pi);
printf("pi = %.8f\n",  pi);
```

Computers can compensate for descriptive inaccuracies, humans cannot.

'A descriptive inaccuracy" is something like: "To get to school, keep straight on this road" (even though no road is actually straight).

Tip: the answer is why programming languages are based on strict rules

FALSE

Feedback: It's exactly the other way around. Humans CAN compensate for descriptive inaccuracies, machines including computers CANNOT. Humans can use rough instructions and use HEURISTICS, machines require exact instructions and use ALGORITHMS. Programming languages are called "formal' because they require strict rules to be used at all. Machines need that because they have no flexibility and are dumb (this is no different for so-called "learning" machines, though the limits of inaccuracies the machines can deal with, are expanded).

Author: Marcus Birkenkrahe
Created: 2022-03-11 Fri 08:43
Validate