Spring 2022 courses

DONE cc Test 2

SETTINGS

Introduction to Programming in C/C++ (CSC 100) test 2:

- 10 questions from quiz 4-6 (possibly modified)
- 10 new questions
- Partial credit enabled
- You can change answers before submitting
- You can resume an incomplete submission
- It is mentioned if > 1 answer is correct

Good luck!

TODO Which of these are allowed conversion specifiers?

More than one answer is correct.

TRUE:

- %-20d
- %.4e
- %g
- %.3f

```
int i=100; float x=100.f;
printf("|%-20d|\n",i);
printf("|%.4e|\n",x);
printf("|%g|\n",i);
printf("|%.3f|\n",x);
```

TODO If the format string is "%d#%d" and the input is •5•#•96, `scanf` succeeds

Note: in •5•#•96, the circle stands for white-space (an empty character)

FALSE

TODO You should not put a new-line character `\n` at the end of a `scanf` format string

TRUE

TODO What's wrong with this program?

The output of the program below should be:

```
: The tax on 200.00 is 35.00, so the grandtotal is 235.00.
```

but instead the output is

```
: The tax on -1093221452 is 0, so the grandtotal is 1080623104.
```

```
int tax;
float taxrate, subtotal, grandtotal;

taxrate = 0.175;
subtotal = 200;
tax = subtotal * taxrate;
grandtotal = subtotal + tax;

printf("The tax on %d is %d, so the grandtotal is %d\n",
       subtotal, tax, grandtotal);
```

Which of these measures will fix the program?

More than one answer is correct.

TRUE:

- Declare `tax` as `float` variable
- Change the conversion specifications from `%d` to `%.2f`

FALSE:

- Initialize `subtotal` as `2 * 100`
- Declare `taxrate`, `subtotal` and `grandtotal` as `int`

SOLUTION

```
float tax, taxrate, subtotal, grandtotal;

taxrate = 0.175;
subtotal = 200;
tax = subtotal * taxrate;
grandtotal = subtotal + tax;

printf("The tax on %.2f is %.2f, so the grandtotal is %.2f\n",
       subtotal, tax, grandtotal);
```

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.

`TODO` How many mistakes can you find in these statements?

```
float int1=1, int2=2;
printf("%d\n"; int 1 + int 2);
```

TRUE:

- 3

FALSE:

- 4
- 2
- 0

```
int int1=1, int2=2;
printf("%d\n", int1 + int2);
```

TODO Which of these conversion specifications will print an `int` left aligned on 4 spaces?

TRUE:

- `"%-4d"`

FALSE:

- `"%4d"`
- `"%l4d"`
- `"%-4f"`

```
int i=3;
printf("|%-4d|\n",i);
printf("|%4d|\n",i);
printf("|%-4f|",i);
```

TODO Identify the tokens in the printf statement

```
printf ( "Height: %d\n"   ,   height )  ;
  A    B        C         B     A    B  B
```

A	function identifier
B	punctuation
C	string literal

TODO Identify the properly formatted format strings

Tip: don't worry about the string content, only about the string following the syntax rules (= compiler won't complain)

More than one answer is correct.

TRUE:

- `"integer: %d\\integer: %f\n"`
- `"hello%5e ffff ggg\n"`
- `"%#d"`

FALSE:

- `"%ffffff\n`

```
int i=10; float x=10.0;
printf("integer: %d\\integer: %f \n",i,x);
printf("hello%5e ffff ggg \n", x);
printf("%#d",i);
```

TODO 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 (aka terminal).

Order the steps below from 1-4 where 1 is the first and 4 is the last step.

1. Write source code in an Org-mode code block (#+begin_src…#+end_src) and put `:tangle file.c` in the header of the code block
2. Tangle the Org-mode code block with `C-c C-v t` to obtain the C source code file `file.c`
3. Compile `file.c` on the command line with the command: `gcc -o outfile file.c` to get an executable file `outfile`
4. Run the executable file named `outfile` on the shell by entering it at the prompt as: `outfile.exe` or `outfile` (Windows, eshell), or `./outfile` (MacOS, eshell)

TODO Upon compilation, the preprocessor looks for `#define` and `#include` directives

TRUE

This code will not compile because the `return 0;` is missing at the end of `main()`

```
int main(void) {
  // nothing to see or do here
}
```

FALSE

What does `void` in `main(void)` mean?

TRUE:

- The function does not take input

FALSE:

- The function will not return any output
- `main(void)` is an empty function (no statements)
- The program will always return a `0`.

Be the compiler!

The following program statements should print out `300` but it does not even compile! What's wrong?

```
int fvoid1=100, fvoid2=200, void;
printf("%dn", void = fvoid1 + fvoid2);
```

This question has more than one correct answer.

TRUE:

- The keyword `void` is protected, cannot be a variable
- There is a `\` missing in the formatter `%dn`

FALSE:

- The variables in `printf` are not recognized
- The formatter should be `%f`

Feedback: the correct statements would be

```
int fvoid1=100, fvoid2=200, void0;
printf("%d\n", void0 = fvoid1 + fvoid2);
```

Which of these are not legal identifiers?

TRUE:

- `100_bottles`
- `bottles-100`

FALSE:

- `_100_bottles`
- `one___hundred___bottles`

```
int 100_bottles;
int bottles-100;
int _100_bottles;
int one___hundred___bottles;
```

Be the compiler!

The function call `hello()` below should print out `hello world` but it does not though the program compiles and runs. The warning is `"hello() defined but not used"`

```
#include <stdio.h>

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

TRUE:

- The function `hello()` needs to be called inside `main`.

FALSE:

- There are too many `return 0;` commands - only one is needed.
- This is an Emacs problem. It works on the command line.
- The function `puts()` does not print. It should be `printf()`

Reading input

Suppose that we call `scanf` as follows:

```
scanf("%d%f%d", &i, &x, &j);
```

If the user enters

```
10.3 5 6
```

What will be the values of `i`, `x`, and `j` after the call?

Assume that `i` and `j` are uninitialized `int` variables, and `x` is an uninitialized `float` variable.

TRUE:

- `i=10, x=.300000, j=5`

FALSE:

- `i=10, x=.300000, j=5`
- `i=10, x=5.000000 j=6`
- `i=10.3, x=0, j=6`

Feedback: program to check

```
int i, j;
float x;
scanf("%d%f%d", &i, &x, &j);
printf("i = %d, x = %f, j = %d\n", i, x, j);
```

```
i = 10563276, x = 0.000000, j = 67
```

The input is parsed as 10 .3 5 because that is the input that was formatted in scanf(). This program would print out what we entered:

```
int x,j;
float i;
scanf("%f%d%d", &i, &x, &j);
printf("i = %.1f, x = %d, j = %d\n", i, x, j);
```

```
i = 10.3, x = 5, j = 6
```

Reading input

Suppose that we call `scanf` as follows:

```
scanf("%f%d%f", &x, &i, &y);
```

If the user enters

```
12.3 45.6 789
```

What will be the values of `x`, `i`, and `y` after the call?

Assume that `x` and `y` are uninitialized `float` variables, and `i` is an uninitialized `int` variable.

TRUE:

- `x=12.300000, i=45, y=.600000`

FALSE:

- `x=12.300000, i=45, y=789.000000`
- `x=12.3, i=45.789, y=0.000000`
- `x=12.30, i=45, y=789.00`

Feedback: program to check

```
int i;
float x,y;
scanf("%f%d%f", &x, &i, &y);
printf("x = %f, i = %d, y = %f\n", x, i, y);
```

```
x = 12.300000, i = 45, y = 0.600000
```

The input is parsed as 12.3 45 .6 because that is the input that was formatted in scanf(). This program would print out what we want to see.

```
int y;
float x,i;
scanf("%f%f%d", &x, &i, &y);
printf("i = %f, x = %f, j = %d\n", x, i, y);
```

```
i = 12.300000, x = 45.599998, j = 789
```

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);
```

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

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

Author: Marcus Birkenkrahe
Created: 2022-03-17 Thu 17:04
Validate