Solutions: 4 Variables Practice

Table of Contents

1. 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;
```

2. 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;
```

3. 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;
```

4. 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 the code in 1 shown below
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
```

5. 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
```

6. 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
```

7. 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
```

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: 2023-02-23 Thu 07:23