Information Processing 2 — Week 1 exercises

Complete as many exercises as you can in-class on your own laptop. You do not need to submit the answers online. Instead, ask an instructor to check your answer when prompted to do so.

Formatted output using `printf`

1 Using multiple printfs to generate one line of output

In class we wrote this function to greet the world:

```
#include <stdio.h>

int main()
{
    printf("hello, world\n");
    return 0;
}
```

We could also print the word `"hello,"` (including a comma), then the word `" world"` (preceded by a space), and finally the newline `"\n"` using three separate calls to `printf()`. Modify your program to print its output in these three separate parts.

▷▷ **[2 points]** Ask an instructor to check and record your work.

2 Using field widths in conversion specifiers

Within a `printf` format string, the conversion specifier '`%i`' says to print the next argument as a decimal integer. You can include a minimum field width for the output in the specifier. The field width is a decimal number immediately following the '`%`' symbol. If the output requires fewer characters than the given field width then additional spaces are printed before the output to fill up the entire field.

The following program prints the number 42 using as many characters as are needed for the number (in this case two) followed by a newline character.

```
int main()
{
    printf("%d\n", 42);
    return 0;
}
```

Modify the program so that it prints the number 42 in a field ten characters wide, with the number right justified within the field.

▷▷ **[2 points]** Ask an instructor to check and record your work.

3 Lining up columns of numbers

The temperature chart program we wrote in class looked like this:

```
#include <stdio.h>

int main()
{
    int fahr, celsius;
    int lower, upper, step;

    lower =   0;    // lower limit of temperature
    upper = 300;    // upper limit of temperature
    step  =  20;    // step size between temperatures

    fahr = lower;
    while (fahr <= upper) {
        celsius = (fahr - 32) * 5 / 9;
        printf("%d\t%d\n", fahr, celsius);
        fahr = fahr + step;
    }

    return 0;
}
```

The output from this program is in two columns, with the first few appearing like this:

```
0       -17
20      -6
40      4
60      15
```

One problem with this is that the numbers are left justified. Numbers are usually printed right justified in a fixed field width, so that their final 'units' digits all 'line up' vertically.

Modify the conversion specifier in the `printf()` format string to use field widths of 3 for both columns of numbers, with a single space separating the two columns. When you are done, the first few rows should look like this:

```
  0 -17
 20  -6
 40   4
 60  15
```

▷▷ **[2 points]** Ask an instructor to check and record your work.

4 Using floating-point values

The Celsius values in the program are not very accurate because the arithmetic is all being performed using integers.

Modify the program so that the variables `fahr` and `celsius` are of type `float` rather than `int`. Note that you will have to change the conversion specifiers in the `printf()` format string because now the second and third arguments are floating point numbers, not integers. The conversion `"%`x`.`y`f"` will print a floating point number in a field of width x using y digits of precision after the decimal point. Ensure each column of numbers is printed in a field six characters wide with one digit of precision after the decimal point. When you are done, the first few columns should look like this:

```
  0.0  -17.8
 20.0   -6.7
 40.0    4.4
 60.0   15.6
```

▷▷ **[2 points]** Ask an instructor to check and record your work.

5 Celsius to Fahrenheit conversion

To perform Celsius to Fahrenheit conversion the formula is:

$$\text{fahrenheit} = 32 + \text{celsius} * 9/5$$

Modify your program to print a table of Celsius to Fahrenheit conversions covering the range 0 C to 100 C in steps of 10 C. Verify that your program produces a table that begins and ends like this:

```
  0.0   32.0
 10.0   50.0
     ...
 90.0  194.0
100.0  212.0
```

▷▷ **[2 points]** Ask an instructor to check and record your work.

Challenge

In the above programs you used `"%d"` to print decimal integers and `"%f"` to print floating point numbers. The following program prints the wrong result:

```
#include <stdio.h>

int main()
{
    printf("%f\n", 42);
    printf("%d\n", 42.0);

    return 0;
}
```

WRONG

Why does this program produce the wrong result?

DATA TYPE