Warmup (optional)

a) The value 6 is put into the variable var. It then gets taken out, divided by 2, and put back into var. The result is 6/2 = 3.

b) In C, assignments themself return the value that is being assing, and can be used as part of another assignment. Exactly that is happening here: the floating value 5 is being assigned to the variable y, and that operation returns 5, that is being used to operate with x. Then is the 5 taken and multiplied with the value in x, wich is 2, and the result 10 is put assigned to x.
Also, the assignations are made from right to left.
x = 10, y =5

c) The values are 2, 0 and 1 (true). The *data type* of expr is int, as boolean actually doesn't exist as a data type. The values are 0 and 1 for false and true, respectively.
Version one:

```
int i = 1, j = 1, expr;
expr = i-- && j--;
printf("i = %d, j = %d, expr = %d\n", i, j, expr);
```

```
i = 0, j = 0, expr = 1
```

Version two:

```
int i = 1, j = 1, expr;
expr = i-- && --j;
printf("i = %d, j = %d, expr = %d\n", i, j, expr);
```

```
i = 0, j = 0, expr = 0
```

Exercise 4.1.

a) x = a/b;
x = 0

b) x = (double)a/b;
x = 0.75

c) x = (double)(a/b);
x = 0.0

d) x = a/(double)b;
x = 0.75

e) x = myFunc(0.8);
x = -1.0 ?

f) int i = myFunc(−1);
i = 0

g) unsigned int ui = myFunc(0);
ui = -1 ?

Exercise 4.2.

a) The for loop shifts every bit to the right, and for every iteration s increses if the last number is 1. This effectively counts the amount of bits in k that are 1.
b) If 2^m is the lowest power of 2, that is bigger than n, than m is the number iterations the for loop will take. This is due to the fact that to right a number bigger than b^u and smaller than b^(u+1), it takes u + 1 digits in base b to right that number. This is the amount of bits required to store k.
c) The signed versions of the already positive integers work equal because the bit representation doesn't change. For the negative values, however, it does change: it won't stop. I made a counter to check that it was because it was stucked in the loop and not because of something else that I haven't thought of, and it confirmes it. The loop just keeps going because the bit representation of a negative number implies that the leading digits are 1 and not 0, so for every bit shift a new 1 is generated and the loop just goes on.
d)