Week 8 - Introductory Programming Lab 4

Exercise 6: `if` conditional

Background

The logic of an `if` statement is fairly simple. If the statement is true, the *code block* (indicated by a statement or the contents of a `{...}` block) is run. A closely related command is the `if ... else`, which acts precisely as you would expect.

Here are two cautions to avoid common problems.

1. In C, `x = 1` and `x == 1` do completely different things:
    - The single = is an *assignment* operator: it changes to the value of x to be 1.
    - The double == is the *comparison* operator: it returns true if x is equal to 1.
2. If you forget to use the `{ ... }` code block, then it will be very easy to make a mistake later on. The computer will happily execute statements which you did not think would be running. This is illustrated at the bottom of the "conditional" example.

Technical details

Conditional:

```
#include <stdio.h>

int main() {
     int x = 3;

     if (x == 4) {
         // this line will not be printed
         printf("x is equal to 4.\n");
     }
     // one of these two lines will be printed
     if ((x > 10) && (1 != 0)) {
         printf("Expression is true.\n");
         printf("x is greater than 10.\n");
     } else {
         printf("Expression is false.\n");
         printf("x is not greater than 10.\n");
     }

     // something weird happens here!
     if (x > 10)
         printf("Expression is true.\n");
         printf("x is greater than 10.\n");

     getchar();
}
```

The logical operators (not `!`) (and `&&`) (or `||`) are particularly useful with `if`. Occasionally the exclusive or `^` is useful. Parentheses `()` are highly encouraged when using logical operators.

Your task...

Create a "guessing game" program. Your program should:

- Make the computer pick a random number to be the answer. It should be an `int` between 1 and 32 (inclusive).
- Use the `getRand()` function, but *do not modify it*. Instead, you should take the value it returns (a `float` between 0 and 0.999...) and do some math to transform that into an `int` between 1 and 32).
- Write a *function* for the user's guess. This function must:
  - have one integer argument `(int correct_answer)`,
  - read an `int` from the keyboard,
  - check the user's `int` against the correct answer,
  - output the appropriate message ("correct" / "too high" / "too low"),
  - returns a 1 if the user's guess was correct, and `0` if the user's guess was wrong.
- Give the user 5 chances to guess the right answer.
  (hint: you have already written a function that checks a guess. What happens if you copy&paste that function call 5 times?)
- End the game if the user is correct, or if he's used up all 5 chances. Print either a "you win" or "you lose", as appropriate.
- You may *not* use any loops for this program.

(optional: instead of guessing an `int`, ask players to guess a `float`. Then, instead of trying to guess exactly the right number, the players win if they guess a number within `0.1` of the correct value.)

Show your work to a demonstrator/GTA.

Exercise 7: `While` loops

Background

The simplest loops in C are `while` loops. These loops are very similar to `if` statements. The difference is that there is not `else` statement included in a `while` loop, and the `{ ... }` code block will be repeated as long as the statement is true.

There is no built-in `true` statement in C, so we use `1` for true and `0` for false.

To stop an infinite loop, use `ctrl-c`

A closely-related form is the `do ... while` loop. Examine the code presented -- is there any difference in the output? Try setting `int i=10;`

Technical details

Infinite loops are fun:

```
#include <stdio.h>

int main()
{
     while (1) {
         printf("Infinite loop is infinite.\n");
     }
    getchar();
}
```

Non-infinite loops are more useful:

```
#include <stdio.h>

int main()
{
     int i = 0;
     while (i < 10) {
         printf("The loop has repeated ");
         printf("%i times.\n", i);
         i++;
     }
     getchar();
}
```

Another form of loop:

```
#include <stdio.h>

int main()
{
     int i = 0;
     do {
         printf("The loop has repeated ");
         printf("%i times.\n", i);
         i++;
     } while (i < 10);
     getchar();
}
```

Your task...

Save your guessing game from the previous exercise ("if conditionals") with a new file name; we will be modifying it, but you don't want to lose all your hard work.

Modify your guessing game:

- Instead of giving the user X chances, keep on offering guesses until the user is correct.
- Keep track of the number of guesses, and tell the player how many guesses it took.
- After the first game is finished, start a new game with a different answer to guess. Keep track of the number of guesses the player needed to win.
- After the second (and later) games are finished, tell the player their score (number of guesses) and the best score (the fewest number of guesses). If the player's score is better, store it as the best score.
- Keep on repeating until the player wins with 4 or fewer guesses. (or until the player hits `ctrl-c`)

(optional: randomly change the rules -- instead of always going from 1 to 32, let the computer randomly decide to do 100 to 131, -97 to -66, etc. If you change the size of the range, you can't compare high scores for different games. However, you could program the computer to randomly select "all numbers divisible by 4 between 32 and 156" and the like.)

Show your work to a demonstrator/GTA.