Code Quality Standards

Your code must meet the code quality standards. If you've taken CMPUT 174 before these should be familiar to you.

- Use readable indentation.
  - Blocks should be indented (everything between `{` and `}`)
  - One line should not have more than one statement on it. However, a long statement should be split into multiple lines.
- Use only idiomatic for loops.
- Use descriptive variable names. It should be obvious to the person reading (and marking your code) what each variable does.
- Every switch case should have a break.
- Never use goto.
- Never use control flow without curly braces (`if`, `else`, `do`, `while`, `for`, etc.)
- Use `<stdbool.h>`, `bool`, `true`, and `false` to represent boolean values.
  - Never compare with `true`, e.g. never `== true`.
- Do not leave commented-out code in your code.
- Provide comments for anything that's not totally and completely obvious.
- Always check to see if I/O functions were actually sucessful.
- On an unexpected error, print out a useful error message and exit the program.
  - For invalid input from the user you should handle it by asking the user to try again or by exiting the program with `exit(1)`, `exit(2)`, etc. or returning 1 or 2 etc. from main.
  - For unexpected errors, such as fgets failing to read anything, consider `abort()`.
- Main should only return 0 if the program was successful.
- Do not use magic literals (magic numbers or magic strings).
  - If a value has a particular meaning, give a meaningful name with `#define` or by declaring a constant with `const`.
  - Values other than 0 and 1 with the same meaning should not appear more than once.
  - 0 or 1 with a meaning other than the immediately obvious should also be given a name.
  - String literals should not appear more than once.
  - This includes magic numbers that appear in strings!
- Program should compile without warnings with `gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3`.
- For more info, see "Code & Compilation Requirements" on eClass.

Note: The next assignment will have more Code Quality requirements.

Testing your Program

Correct input-output examples are provided. For example, `q1a-test1-input.txt` is the input to your `./question1` program. If your program is correct, its output will match `q1a-test1-expected-output.txt`.

You can tell if your output matches exactly by saving the output of your program to a file with bash's `>` redirection operator. For example, `./question1 >my-output-1.txt` will save the output of your question1 program into the file named `my-output-1.txt` instead of showing it on the screen. Be warned! It will overwrite the file, deleting anything that used to be in `my-output-1.txt`.

Similarly, you can give input to your program from a file instead of typing it by using bash's `<` redirection operator. For example, `./question1 <q1a-test1-input.txt` will run your program with the contents of `q1a-test1-input.txt` instead of being typed out.

These two can be combined. For example,

```
./question1 <q1a-test1-input.txt >my-output-1.txt
```

will use the contents of `q1a-test1-input.txt` as input and save the output of your program in `my-outputr-1.txt`.

When you want to check if your output is correct, you can then use the `diff` command from bash to compare two files. For example,

```
diff -b my-output-1.txt q1a-test1-expected-output.txt
```

will compare the two files `my-output-1.txt` and `q1a-test1-expected-output.txt` and show you any differences. `-b` tells diff to ignore extra spaces and tabs.

`diff` will only show you something if there's a difference between the two files. If `diff` doesn't show you anything, that means the two files were the same!

So, putting it all together, to check if your program handles one example input correctly, you can run:

```
./question1 <q1a-test1-input.txt >my-output-1.txt
diff -b my-output-1.txt q1a-test1-expected-output.txt
```

If `diff` doesn't show you anything, that means the two files were the same, so your output is correct.

This is what the included scripts (`test-q1a.sh`, etc.) do.

However, the examples are just that: examples. If your code doesn't produce the correct output for other inputs it will still be marked wrong.

Questions

Question 1

Overview

Write a C program that simulates the Monty Hall game. Monty Hall is a scenario where you imagine being on a game show and you're given a chance to win a car by selecting one of the three doors. Behind one door is a car while behind the other two doors are goats. The car and the goats were placed randomly behind the doors before the show. Monty Hall works according to following rules:

1. The player choose one of the three doors.
2. Monty chooses another door and opens it revealing a goat. (Monty never chooses the door with the car behind it!)
3. Monty asks the player if they want to switch the door choice or keep the original choice.
4. The user may switch by pressing `Y` or `y`. Alternatively, the user may choose to not switch by pressing `N` or `n`.
5. The door selected by the player is opened and the player wins or loses. The message is displayed accordingly.
6. Monty asks the player if they want to play again.

Additional Requirements

- Put your C code for this question in `question1.c`
- You should compile the program as `./question1`
- You must make a ./question1.sh to compile and run your question1.c program
- You must demonstrate the proper use of a switch statement in this question.
- `stdlib.h` has a random function that you need to use to generate a number randomly, `rand()`.
    - To get help on using `rand()` try `man 3 rand` at your bash prompt!
    - Do not use `srand()`, or the car will be behind a different door and your output will be wrong!
- Wait for the user to press enter after the yes-or-no questions. If the user enters more than one letter, ignore the other letters. Only consider the first letter.
    - For example, `Yes` should count as yes, `no` should count as no, `Ninetails` should also count as no. This is just to keep things simple for you.
- If you need a string buffer, use a buffer big enough to hold a string 1024 chars long.
- Represent the doors as integers 0, 1, 2.
    - The int returned by `rand()` is a random integer that can be very large. Divide it by three and take the remainder to get a random integer 0, 1, or 2.
- At the beginning of a game, take 1 random number from `rand()` to decide which door the car is behind.

    - If the player's first guess is the door with the car, Monty can choose to reveal either goat. For this assignment, have monty choose to reveal the goat behind the door with the lowest number.
    - Example: Car is behind #1. Player's initial guess is door #1. Monty reveals the goat behind door #2.
    - If the player's first guess is a door with a goat, Monty must reveal the other goat.
- The car can only move between games.
- You should only need to modify one number and no strings in your code to change the number of doors.

Example Runs

Following is an example run of a program in which the car was behind door 2.

```
Choose a door from 1 to 3: 1
You chose door #1.
Monty opens door #3, revealing a goat!
Would you like to switch to door #2? Y
You chose door #2.
Monty opens door #2, revealing a car! You win!
You've won 1 games out of 1 games.
Would you like to play again? n
```

Following is an example of putting in invalid inputs:

```
Choose a door from 1 to 3: I don't know
Choose a door from 1 to 3: There's only 3?
Choose a door from 1 to 3: OKay um...
Choose a door from 1 to 3:
Choose a door from 1 to 3: 4
Choose a door from 1 to 3: Door 2
Choose a door from 1 to 3: 1
You chose door #1.
Monty opens door #3, revealing a goat!
Would you like to switch to door #2? Why?
Error: please enter Y or N: What's wrong with Door #1?
Error: please enter Y or N:
Error: please enter Y or N: Maybe.
Error: please enter Y or N:
Error: please enter Y or N: Yes
Monty opens door #2, revealing a car! You win!
Would you like to play again? Not really
```

Marking

- ☐ **1 Point** Program output is correct for a valid input. (Examples: `test-q1a.sh`)
- ☐ **1 Point** Program output is correct for an invalid input. (Examples: `test-q1b.sh`)
- ☐ **1 Point** Program uses switch statements (and uses them correctly).
- ☐ **1 Point** Quality of question1.c meets the quality standards, listed above.

- **1 Point** `question1.sh` meets the requirements above.

Hints

- If you are left with an extra newline after doing scanf, try adding %*c to the scanf. This will eat 1 character (the newline character) but you do not need to provide an argument to scanf and it won't count towards the return value of scanf. See `man 3 scanf`.
- It's probably easier to use `fgets` to read all user responses. If you use `fgets`, you can convert the string it reads into its first argument to an int using the function `atoi`. See `man 3 atoi`.

Question 2

Write a C program `question2.c` that takes a number as input 1 to 9 and prints a number pattern as follows. If the user enters a number that's not 1 to 9 or not a number the program returns 1 and prints "Invalid input\n" to the terminal.

Additional Requirements

- Put your C code for this question in `question2.c`
- You should compile the program as `./question2`
- You must make ./question2.sh to compile and run your question2.c program
- Program input and output should match the examples.

Example runs

```
Enter a number between 1 and 9: 0
Invalid input
Enter a number between 1 and 9: 1
11
Enter a number between 1 and 9: 2
_1_212
212_1_
Enter a number between 1 and 9: 3
__1__32123
_212__212_
32123__1__
Enter a number between 1 and 9: 4
___1___4321234
__212___32123_
_32123___212__
4321234___1___
Enter a number between 1 and 9: 5
____1____543212345
___212____4321234_
__32123____32123__
_4321234____212___
543212345____1____
Enter a number between 1 and 9: 6
```

```
_____1_____65432123456
____212_____543212345_
___32123_____4321234__
__4321234_____32123___
_543212345_____212____
65432123456_____1_____
Enter a number between 1 and 9: 7
______1______7654321234567
_____212______65432123456_
____32123______543212345__
___4321234______4321234___
__543212345______32123____
_65432123456______212_____
7654321234567______1______
Enter a number between 1 and 9: 8
_______1_______876543212345678
______212_______7654321234567_
_____32123_______65432123456__
____4321234_______543212345___
___543212345_______4321234____
__65432123456_______32123_____
_7654321234567_______212______
876543212345678_______1_______
Enter a number between 1 and 9: 9
________1________98765432123456789
_______212________876543212345678_
______32123________7654321234567__
_____4321234________65432123456___
____543212345________543212345____
___65432123456________4321234_____
__7654321234567________32123______
_876543212345678________212_______
98765432123456789________1________
Enter a number between 1 and 9: 10
Invalid input
```

To run the tests implement question2.sh to compile and run your question2.c.

Run test-Q2A.sh from the assignment directory:

```
bash test-Q2A.sh
```

Run test-Q2A.sh from the assignment directory

```
bash test-Q2B.sh
```

The test cases should produce no output whatsoever.

Marking

- ☐ **1 Point** Program output is correct for a valid input. (Examples: `test-q2a.sh`)
- ☐ **1 Point** Program output is correct for an invalid input. (Examples: `test-q2b.sh`)

- ☐ **1 Point** Program uses idiomatic for loops to print the number triangles
- ☐ **1 Point** Quality of question2.c meets the quality standards, listed above.
- ☐ **1 Point** `question2.sh` meets the requirements above.

Hints

To check the return value of your program you can do `echo $?` in bash immediately after your program.

```
$ ./question2
Enter a number between 1 and 9:
lasjkdf
Invalid input
$ echo $?
1
```

Your program returns whatever it returns from main. If you're not in main, you can have your program exit with a specific return value by using the `exit` function from `stdlib.h`. For example, `exit(1)` will cause the program to exit and return 1. `return 1` in `main` will do the same thing.

Submission

Test your program!

Always test your code on the VM or a Lab computer before submitting!

Make a 1 line (excluding the comments and header) shell script for each question that will compile and run the C program for that question. Name the scripts `question1.sh` and `question2.sh` respectively. Make sure the program successfully compiles the program and then runs it. If the program doesn't compile it should not run the executable. The shell program should use 1 operator to achieve this and it should all fit on the same line. You can assume the shell script is run in the directory that contains both the source code and the executable. Run the `test-Q1A.sh` script for question1. Run the `test-Q1B.sh` script for question1. Run the `test-Q2A.sh` script for question2. Run the `test-Q2B.sh` script for question2. The scripts should produce no output.

Tar it up!

Make a tar ball of your assignment. It should not be compressed. The tar name is `__YOUR__CCID__-assignment1.tar`

the tar ball should contain:

- `__YOUR__CCID__-assignment2/` # the directory
- `__YOUR__CCID__-assignment2/README.md` # this README filled out with your name, CCID, ID #, collaborators and sources.
- `__YOUR__CCID__-assignment2/question1.c` # C program
- `__YOUR__CCID__-assignment2/question2.c` # C program
- `__YOUR__CCID__-assignment2/question1` # executable
- `__YOUR__CCID__-assignment2/question2` # executable
- `__YOUR__CCID__-assignment2/question1.sh` # shell script
- `__YOUR__CCID__-assignment2/question2.sh` # shell script

Dr. Hindle's assignment2 tar would be called `hindle1-assignment2.tar` and it will contain:

- `hindle1-assignment2/` # the directory
- `hindle1-assignment2/README.md` # this README filled out with Dr. Hindle's name, CCID, ID #, collaborators and sources.
- `hindle1-assignment2/question1.c` # C program
- `hindle1-assignment2/question2.c` # C program
- `hindle1-assignment2/question1` # executable
- `hindle1-assignment2/question2` # executable
- `hindle1-assignment2/question1.sh` # shell script
- `hindle1-assignment2/question2.sh` # shell script

Submit it!

Upload to eClass! Be sure to submit it to the correct section.

Marking

This is a 10-point assignment. It will be scaled to 4 marks. (4% of your final grade in the course: A 10/10 is 100% is 4 marks.) Partial marks may be given at the TA's discretion.

- You will lose all marks if not a tar (a `.tar` file that can be unpacked using `tar -xf`)
- You will lose all marks if files not named correctly and inside a correctly named directory (folder)
- You will lose all marks if `README.md` does not contain the correct information!
    - Markdown format (use `README.md` in the example as a template)
    - Name, CCID, ID #
    - Your sources
    - Who you consulted with
    - The license statement below

License

This software is NOT free software. Any derivatives and relevant shared files are under the following license:

Copyright 2020 Abram Hindle, Hazel Campbell

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, and submit for grading and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

- The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
- You may not publish, distribute, sublicense, and/or sell copies of the Software.
- You may not share derivatives with anyone except UAlberta staff.
- You may not pay anyone to implement this assignment and relevant code.
- Paid tutors who work on this code owe the Department of Computing Science at the University of Alberta $10000 CAD per derivative.
- By publishing this code publicly said publisher owes the Department of Computing Science at the University of Alberta $10000 CAD.
- You must not engage in plagiarism

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.