Week 04, Homework 04	Weight: 2%	Due: Monday second week of mid-semester break, 09:00am (via `sync`)

Pre-homework Preparation:

- **Lab: Week 04, Lab 04**
- **Lectures: Weeks 01, 02, 03, 04**

Homework Activities:

Obtain the homework files using **`sync`**.

For each question's source file, be sure to fill in the file comment header accurately!

Question 1: Input checking

In the file **`hw04q01.c`** write a function called **`confirm()`** that produces the following output on the terminal:

```
Confirm (y/n):
```

The function should then read one character and if the value is **`'y'`** or **`'Y'`** then **`confirm()`** should return a true value (non-zero). If the value of the character is **`'n'`** or **`'N'`** then **`confirm()`** should return false. Otherwise **`confirm()`** should print:

```
Input not recognised, try again.
```

The function should repeat the prompt to confirm and attempt to get a good response. It should do this as many times as necessary.

Note that there should be a space at the end of the prompt, but no newline, so a complete session might look like this:

```
Confirm (y/n): x
Input not recognised, try again.
Confirm (y/n): C
Input not recognised, try again.
Confirm (y/n): N
```

To test your code write your own **`main()`** function in hw04q01.c and use it to call **`confirm()`**. That way you can run your own tests by using **`make`**, followed by **`./hw04q01`**.

Question 2: Character classifier

In **hw04q02.c** write a function called **classify_character()** that conducts a terminal session as follows:

```
Enter character: 3
'3' is a digit.
```

The function should be able to output the following messages depending on the character *c* entered.

```
'c' is a digit.
'c' is a lowercase letter.
'c' is an uppercase letter.
'c' is whitespace.
'c' is a symbol.
```

A symbol is any punctuation or bracket. Consult the ASCII table at http://simple.wikipedia.ork/wiki/ASCII to determine what characters are symbols.

We will declare space, tab and newline to be whitespace characters (actually there are more, but that is all we will test for).

The classify_character() function only needs to classify one character (you do not need to write a loop).

To test your code write your own **main()** function in **hw04q02.c** which calls **classify_character()**.

Make sure to define at least one function to help with *each character classification*.

To use **printf()** and **scanf()** you will need to include the standard header **stdio.h** in **hw04q02.c**. This was done for you in Question 1 but in the remaining questions it will be your responsibility.

Question 3: Guess the number

In **hw04q03.c** write a function called **guess_the_number()** that can conduct terminal sessions like this:

```
I'm thinking of a number between 0 and 100, what is it?
Your guess: 50
Lower
Your guess: 25
Higher
Your guess: 37
Higher
Your guess: 43
Lower
Your guess: 40
Lower
Your guess: 38
You got it!
It took you 6 guesses.
```

Your function should 'think of' a different number in the range 0 to 100 each time it is called (hint: **srand**, **time** and **rand** – don't forget to include appropriate headers – maybe try **man rand** and **man time** at the command prompt).

Decide what your code should do if the user enters something other than a number when guessing and implement it to ensure a good user experience.

Question 4: Calculator

In hw04q04.c write a function called **`calculator()`** that presents an interactive desktop calculator. Here is a sample session:

```
0.00000
: =
= 1.5
1.50000
: *
* v
Error, not a number.
* 2
3.00000
: o
Error, not a command.
Commands are:
+ add
- subtract
* multiply
/ divide
= set value
h print this help
q quit
: q
Goodbye!
```

Again you can write your own main function in **`hw04q01.c`** to help you test your **`calculator()`** function, or you can run **`make test`** to run a series of strict tests.
Note the following about the way that calculator works:

- First it prints the current running total
- Then it prints the : prompt (followed by a space) to ask for a command
- The **+**, **−**, ***** and **/** commands all need a number (the argument) to be entered
- When entering the argument the command (followed by a space) should be used as the prompt.
- If the user enters a bad argument the program outputs **`"Error, not a number."`** and asks again until the user enters a good argument (hint: **`scanf()`** returns the number of specifiers that it managed to match).
- If the user enters a bad command the program outputs **`"Error, not a command."`** and the help text.

You should structure your algorithm using several functions, each properly commented.

Question 5: Adventure game

The original adventure game was called 'Colossal Cave Adventure', or usually just **`adventure`**. It was text-based and distributed for free with early Unix systems.

The source code for a version of **`adventure`** is included in **`~/p1.2015s1/homework04/working_copy/q05/adventure`**. Change directory to that location and type make to build it (ignore the various compiler warnings that this generates) and then **`./adventure`** to play.

When you feel you are familiar with the basic style of the game, return to the q05 directory and, using your knowledge of C, implement your own adventure game. It should:

- Be based on a theme of your own choosing, which is fun and not offensive
- Allow the user to enter commands consisting of a single character each
- Allow the user to move between a number of locations, receiving short descriptions of each
- Allow the user to collect at least one object and use it in another location to solve a simple puzzle.
- Have a way for the user to win.
- Have a way for the user to get help – a 'h' or '?' command would be suitable.
- Have a way for the user to quit.

You should structure your game algorithm using functions.

Homework Submission:

Run the `sync` command to submit your completed homework.

Marking Criteria:

Have you completed each of the following?

Marking Criteria:	Week 03, Homework 03 Weight 2%	Maximum Possible Mark:	Mark achieved?
Q1:	Code correctly handles 'y', 'Y', 'n' and 'N'	8	
	Code correctly handles bad input.	2	
Q2:	Code reliably classifies characters	5	
	A function helps with each classification	5	
Q3:	Code reliably plays a game of guess the number	4	
	Random number generated appropriately	2	
	Code handles bad input well	2	
	Code well-structured into functions	2	
Q4:	Calculator functionality implemented	5	
	Output is as specified in the question	2	
	Code well-structured into functions	2	
Q5:	Can visit several rooms	3	
	Can collect an object to solve a puzzle elsewhere	3	
	Algorithm structured into functions	2	
	Parameters and return values used to exchange data between functions	2	
Total:		**50**	