EXERCISE 10

Switch Statements

In other languages, like Ruby, you have a switch-statement that can take any expression. Some languages, like Python, don't have a switch-statement because an if-statement with Boolean expressions is about the same thing. For these languages, switch-statements are more like alternatives to if-statements and work the same internally.

In C, the switch-statement is actually quite different and is really a *jump table*. Instead of random Boolean expressions, you can only put expressions that result in integers. These integers are used to calculate jumps from the top of the switch to the part that matches that value. Here's some code to help you understand this concept of jump tables:

ex10.c

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
    if (argc != 2) {
        printf("ERROR: You need one argument.\n");
        // this is how you abort a program
        return 1;
    }

    int i = 0;
    for (i = 0; argv[1][i] != '\0'; i++) {
        char letter = argv[1][i];

        switch (letter) {
            case 'a':
            case 'A':
                printf("%d: 'A'\n", i);
                break;

            case 'e':
            case 'E':
                printf("%d: 'E'\n", i);
                break;

            case 'i':
            case 'I':
                printf("%d: 'I'\n", i);
                break;

            case 'o':
            case 'O':
                printf("%d: 'O'\n", i);
                break;

            case 'u':
            case 'U':
                printf("%d: 'U'\n", i);
                break;

            case 'y':
            case 'Y':
                if (i > 2) {
                    // it's only sometimes Y
                    printf("%d: 'Y'\n", i);
                }
                break;

            default:
                printf("%d: %c is not a vowel\n", i, letter);
        }
    }

    return 0;
}
```

In this program, we take a single command line argument and print out all vowels in an incredibly tedious way to demonstrate a `switch-statement`. Here's how the `switch-statement` works:

- The compiler marks the place in the program where the `switch-statement` starts. Let's call this location Y.
- It then evaluates the expression in `switch(letter)` to come up with a number. In this case, the number will be the raw ASCII code of the letter in `argv[1]`.
- The compiler also translates each of the `case` blocks like `case 'A':` into a location in the program that's that far away. So the code under `case 'A'` is at Y + A in the program.
- It then does the math to figure out where Y + letter is located in the `switch-statement`, and if it's too far, then it adjusts it to Y + default.
- Once it knows the location, the program *jumps* to that spot in the code, and then continues running. This is why you have `break` on some of the `case` blocks but not on others.
- If `'a'` is entered, then it jumps to `case 'a'`. There's no break, so it "falls through" to the one right under it, `case 'A'`, which has code and a `break`.
- Finally, it runs this code, hits the break, and then exits out of the `switch-statement` entirely.

This is a deep dive into how the `switch-statement` works, but in practice you just have to remember a few simple rules:

- Always include a `default:` branch so that you catch any missing inputs.
- Don't allow *fall through* unless you really want it. It's also a good idea to add a `//fallthrough` comment so people know it's on purpose.
- Always write the `case` and the `break` before you write the code that goes in it.
- Try to use `if-statements` instead if you can.

What You Should See

Here's an example of me playing with this, and also demonstrating various ways to pass in the argument:

Exercise 10 Session

```
$ make ex10
cc -Wall -gex10.c   -o ex10
$ ./ex10
ERROR: You need one argument.
$
$ ./ex10 Zed
0: Z is not a vowel
1: 'E'
2: d is not a vowel
$
$ ./ex10 Zed Shaw
ERROR: You need one argument.
$
$ ./ex10 "Zed Shaw"
0: Z is not a vowel
1: 'E'
2: d is not a vowel
3:   is not a vowel
4: S is not a vowel
5: h is not a vowel
6: 'A'
7: w is not a vowel
$
```

Remember that there's an `if-statement` at the top that exits with a `return 1;` when you don't provide enough arguments. A return that's not 0 indicates to the OS that the program had an error. You can test for any value that's greater than 0 in scripts and other programs to figure out what happened.

How to Break It

It's *incredibly* easy to break a `switch-statement`. Here are just a few ways you can mess one of these up:

- Forget a `break`, and it'll run two or more blocks of code you don't want it to run.
- Forget a `default`, and it'll silently ignore values you forgot.
- Accidentally put a variable into the `switch` that evaluates to something unexpected, like an `int`, which becomes weird values.
- Use uninitialized values in the `switch`.

You can also break this program in a few other ways. See if you can bust it yourself.

Extra Credit

- Write another program that uses math on the letter to convert it to lowercase, and then remove all of the extraneous uppercase letters in the switch.
- Use the `','` (comma) to initialize `letter` in the `for-loop`.
- Make it handle all of the arguments you pass it with yet another `for-loop`.
- Convert this `switch-statement` to an `if-statement`. Which do you like better?
- In the case for `'Y'` I have the break outside of the `if-statement`. What's the impact of this, and what happens if you move it inside of the `if-statement`. Prove to yourself that you're right.