Lab 2

Bit Manipulation Lab

Due: Week of January 31, before the start of your lab section

This is an individual-effort project. You may discuss concepts and syntax with other students, but you may discuss solutions only with the professor and the TAs. Sharing code with or copying code from another student or the internet is prohibited.

The purpose of this assignment is to give you more confidence in C programming and to begin your exposure to the underlying bit-level representation of data.

The instructions are written assuming you will edit and run the code on your account on the *csce.unl.edu* Linux server. If you wish, you may edit and run the code in a different environment; be sure that your compiler suppresses no warnings, and that if you are using an IDE that it is configured for C and not C++.

Learning Objectives

After successful completion of this assignment, students will be able to:

- Use the ASCII table to determine the corresponding integer values of C **`char`** values.
- Apply arithmetic operators and comparators to C char values.
- Construct and use a bitmask.
- Use bitwise operators and bit shift operators to create and modify values.

Continuing Forward

Your experience with viewing values as bit patterns will be applicable in future labs, as will bit masks and bit operations. Some of the functions you write in this lab will be used in the next lab.

During Lab Time

During your lab period, the TAs will demonstrate how to read the ASCII table and will provide a refresher on bitwise AND, bitwise OR, and left- and right-shifts. During the remaining time, the TAs will be available to answer questions.

No Spaghetti Code Allowed

In the interest of keeping your code readable, you may *not* use any **`goto`** statements, nor may you use any **`continue`** statements, nor may you use any **`break`** statements to exit from a loop, nor may you have any functions **`return`** from within a loop.

Scenario

You've recently been hired to help get the Pleistocene Petting Zoo get started. Your new employer, Archie, is surprisingly honest: he admits to you that some expenses were spared. Archie cheerfully points out that any challenge is also an opportunity to succeed. You suspect your job will offer plenty of "opportunities to succeed."

Figure 1: Some expenses were spared.

1 Using the ASCII table

You soon discover your first ~~challenge~~ opportunity. Archie purchased your workstation from government surplus. Your keyboard is left over from early 2001 and is missing the letter *W*![1] You decide to write an email requesting a new keyboard:

```
TO⟶Archie↩
RE⟶I Need a Working Keyboard↩
↩
Please order a new keyboard for me.  This one is broken.↩
```

(Note: here, the ⟶ symbol represents the `TAB` character which is needed by the email program, and the ↩ symbol represents a `NEWLINE` character.)

You quickly realize that you can't type this directly into your mail program because of the missing *W* key. So you decide to write a short program that will output the text that you want to send. The code you would like to write is:

```c
#include <stdio.h>

int main() {
    printf("TO\tArchie\n");
    printf("RE\tI Need a Working Keyboard\n\n");
    printf("Please order a new keyboard for me. This one is broken.\n");
```

[1] In January 2001, when President Bill Clinton's staff left the White House so that President George W. Bush's staff could move in, they removed the *W* key from several keyboards as a prank.

```
    return 0;
}
```

Of course, the `W` and the `w` are still a problem, but you realize you can insert those characters by using their ASCII values.[2] For example,

`printf("Hello World!\n");`

can be replaced with

`printf("%s%c%s\n", "Hello ", ..., "orld!");`

replacing ... with the ASCII value for `W`. Recall that the first argument for **`printf()`** is a *format string*: `%s` specifies that a string should be placed at that position in the output, and `%c` specifies that a character should be placed at that position in the output.

As you open your editor, the \ key falls off the keyboard, preventing you from typing `\t` and `\n`.

Edit `problem1.c` so that it produces the specified output without using the W key or the backslash key. Build the executable with the command: `make keyboardlab1` – be sure to fix both errors and warnings.

You can double-check that you aren't using the W key or the backslash key by running the constraint-checking shell script:

```
./constraint-check.sh
```

(If you get a "`Permission denied`" error message, then run the command

```
chmod +x constraint-check.sh
```

and then run the shell script.)

You can check that your program has the correct output with this command:

```
./keyboardlab1 | diff - problem1oracle
```

2 Treating Characters as Numbers

Archie replies to your email, assuring you that a new keyboard has been ordered. Meanwhile, he needs you to write some code that will convert uppercase letters to lowercase letters and to indicate whether or not a character is a decimal digit. You realize this is easy work since those actual functions are part of the standard C library with their prototypes in `ctype.h`. As you get ready to impress your boss with how fast you can "write" this code by calling those standard functions, the *3* key (which is also used for *#*) falls off of your keyboard, preventing you from typing **`#include`** `<ctype.h>`. Several other number keys fall off soon thereafter (only *0*, *7*, and *9* remain), along with the *s* key. The *f* key is looking fragile, so you decide that you had better not type too many **`if`** statements (and without the *s* key, you can't use a **`switch`** statement at all).

Edit `problem2.c` so that

- **`iz_digit()`** returns 1 if the character is a decimal digit ('0', '1', '2', ...) and 0 otherwise

[2]Use the ASCII table in the textbook or type `man ascii` in a terminal window.

- **`decapitalize()`** will return the lowercase version of an uppercase letter ('A', 'B', 'C', . . .) but will return the original character if it is not an uppercase letter

You may not **`#include`** any headers, you may not use any number keys other than the 0, 9, and 7 (which is also used for **`&`**) keys, you may not use **`switch`** statements, and you may use at most one **`if`** statement in each function.

Build the executable with the command: `make keyboardlab2` – be sure to fix both errors and warnings.

You can double-check that you aren't using disallowed keys by running the constraint-checking shell script:
`./constraint-check.sh`
Because you are allowed at most one **`if`** statement in each function, you may see two lines with **`if`** statements reported when the script checks for 'f'. You may also see some comments reported when the script checks for '*'.

3 Using Bitmasks and Shifting Bits

Your keyboard was mistakenly delivered to the Plywood Scenery Cutting Studio instead of the Pleistocene Petting Zoo. While that gets sorted out, you "borrow" some tar from the La Brea Tar Pits diorama and use the tar to re-attach your keyboard's missing keys. As you fasten a Scrabble tile in place for the *`W`*, more keys fall off, denying you the use of *`+`*, *`-`*, *`/`*, *`%`*, *`5`*, and *`b`*. You cannot get any more tar from the diorama, so you sit down to your next programming tasks without those keys.

Edit `problem3.c` so that

- **`is_even()`** returns 1 if the number is even (that is, divisible by 2) and 0 if the number is odd
- **`produce_multiple_of_ten()`** will always output a multiple of 10 following a specific formula: if a number is even then divide it by 2; otherwise, subtract 1 from the number and multiply the difference by 5 (for example, an input of 7 yields 30 because $(7 - 1) \times 5 = 30$); repeat until the last decimal digit is 0.

These numbers are guaranteed to be non-negative. You may not use addition (+), subtraction (-), division (/), nor modulo (%). You also may not use the number 5 nor the letter b. (Exceptions: you *may* use the forward-slash (/) for comments, and the percent-sign (%) that is already present in the **`sprintf()`** calls' format strings is allowed)

Hints:

- Following the weighted-sum technique to convert between binary and decimal (or by examining the textbook's Table 2.1), you will note that all even numbers have a 0 as their least significant bit, and all odd numbers have a 1 as their least significant bit
- Less obvious is that you can subtract 1 from an odd number by changing its least significant bit to a 0

- As we will cover in Chapter 3, you can halve a number by shifting its bits to the right by one
- You can create an integer by producing its bit pattern through a series of bit operations

Build the executable with the command: `make keyboardlab3` – be sure to fix both errors and warnings.

You can double-check that you aren't using disallowed keys by running the constraint-checking shell script:
`./constraint-check.sh`
You may see some comments reported when the script checks for `'*'`.

Turn-in and Grading

When you have completed this assignment, upload *problem1.c*, *problem2.c*, and *problem3.c* to Canvas.

This assignment is worth 20 points.

_____ **+4** *problem1.c* produces the specified output.

_____ **+4** **`iz_digit()`** in *problem2.c* determines whether or not a character is a digit.

_____ **+4** **`decapitalize()`** in *problem2.c* converts uppercase letters to lowercase and leaves other characters unchanged.

_____ **+4** **`is_even()`** in *problem3.c* determines whether a number is even or odd.

`produce_multiple_of_ten()` in *problem3.c* has the following:

_____ **+1** Code to assign the value 5 to the variable `five`

_____ **+1** Code to divide an even number by 2

_____ **+1** Code to subtract 1 from an odd number

_____ **+1** Correct functionality

Penalties

_____ **-4** The solution to *problem1.c* uses `w`, `W`, `\n`, or `\t`.

_____ **-4** **`iz_digit())`** uses a digit other than 0 and 9, uses a **`switch`** statment, or uses more than one **`if`** statement.

_____ **-4** **`decapitalize()`** **`#include`**s one or more files, uses a **`switch`** statment, or uses more than one **`if`** statement.

_____ **-4** **`is_even()`** uses arithmetic.

_____ **-4** **`produce_multiple_of_ten()`** uses addition, subtraction, division, or modulo; or **`produce_multiple_of_ten()`** uses the literal the value 5, 0x5, 05, or 0b101.

_____ **-1** for each **`goto`** statement, **`continue`** statement, **`break`** statement used to exit from a loop, or **`return`** statement that occurs within a loop.

Epilogue

Great news! Archie brings you your new keyboard. He also brings you a problem of his own. Because you were held up with the broken keyboard, Archie decided to try some programming on his own, and his code is behaving strangely.

To be continued...