Lab 2

Bits and Pointers Manipulation Lab

Due: Week of September 13, 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. You will also gain practice with pointers and with file input/output.

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++.

Scenario

You've recently been hired to help 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."

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:

[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.

```
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:

```
1 #include <stdio.h>
2
3 int main() {
4     printf("TO\tArchie\n");
5     printf("RE\tI Need a Working Keyboard\n\n");
6     printf("Please order a new keyboard for me. This one is broken.\n");
7     return 0;
8 }
```

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 with this command:

```
grep -e w -e W -e '\\' problem1.c | grep -v '^\s*\*'
```

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

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

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

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

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

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.

Hints:

- Following the weighted-sum technique to convert between binary and decimal (or by examining 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.

4 Stray Values on the Stack

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. He shows you his code:

```
1  /*****************************************************************
2   * This program will output
3   **          Welcome to the
4   **     Pleistocene Petting Zoo!
5   **
6   ** Get ready for hands-on excitement on the count of three! 1.. 2.. 3..
7   ** Have fun!
8   * With brief pauses during the "Get ready" line.
9   *****************************************************************/
10
11 #include <stdio.h>
12 #include <unistd.h>
13
14 void splash_screen() {
15     const char *first_line = "\t     Welcome to the\n";
16     const char *second_line = "\tPleistocene Petting Zoo!\n";
17     printf("%s%s\n", first_line, second_line);
18 }
```

```
19
20 void count() {
21     int i;
22     sleep(1);
23     printf("Get ready for hands-on excitement on the count of three! ");
24     while (i < 3) {
25         fflush(stdout);
26         sleep(1);
27         i++;
28         printf("%d.. ", i);
29     }
30     printf("\nHave fun!\n");
31 }
32
33 int main() {
34     splash_screen();
35     count();
36     return 0;
37 }
```

Sometimes the output was what he expected:

```
      Welcome to the
 Pleistocene Petting Zoo!

Get ready for hands-on excitement on the count of three! 1.. 2.. 3..
Have fun!
```

But sometimes the output was missing the "`1..  2..  3..`":

```
      Welcome to the
 Pleistocene Petting Zoo!

Get ready for hands-on excitement on the count of three!
Have fun!
```

What mistake did Archie make? What change to *one* line will fix Archie's bug? Place your answers in *answers4.txt*.

5 Challenge and Response

You plug in your shiny, new keyboard, tune your satellite radio to the Greatest Hits of the 1920s, and settle in to solving a more interesting problem.

To protect against corporate espionage, you are responsible for writing code for a challenge-and-response system. Anybody can challenge anyone else in the Pleistocene Petting Zoo's

non-public areas by providing the name of a book and a word contained within the book, and the person being challenged must respond with another word from that book, based on certain rules:

- All of the book's words are sorted alphabetically without regard to capitalization (for example, "hello" occurs after "Hear" and before "HELP")
- The challenge word occurs *occurrences* times in the book
- If *occurrences* is an even number then the response word is the word *occurrences* places **before** the challenge word in the alphabetized list; if the challenge word is less than *occurrences* places from the start of the list then the response word is the first word in the list
- If *occurrences* is an odd number then the response word is the word *occurrences* places **after** the challenge word in the alphabetized list; if the challenge word is less than *occurrences* places from the end of the list then the response word is the last word in the list

Here is a simple example. Suppose the words in the specified book are:

word	*occurrences*
apple	7
banana	4
carrot	15
date	3
eggplant	2
fig	6
granola	9
horseradish	9
ice	6
jelly	3
kale	1
lemon	2
mango	8
naan	7
orange	5
pineapple	1
quinoa	11
raisin	4
spaghetti	10
tomato	12

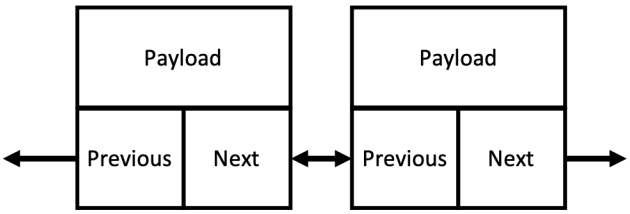
If the challenge word is "horseradish" then because horseradish occurs 9 times in the book, the response word is "quinoa," which is 9 places in the list after "horseradish." If the

challenge word is "eggplant" then the response is "carrot," which is 2 places earlier in the list than "eggplant." If the challenge word is "banana" then the response word is "apple," which is the first word in the list. If the challenge word is "quinoa" then the response word is "tomato," the last word in the list.

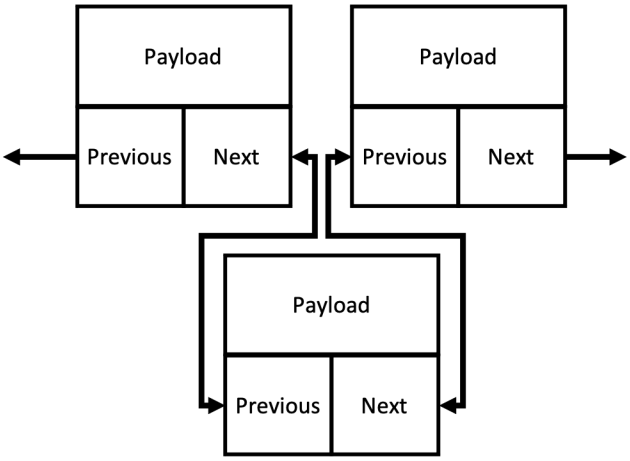
You break the problem down into four sub-problems:

5.1 Doubly-Linked List

A *doubly-linked list* is a linked list with the property that each node maintains a link not only to the `next` node but also a link to the `previous` node. In C, these links are pointers.



Inserting new node, C between adjacent nodes A and B (where $B = A.next$ and $A = B.previous$) requires connecting $C.previous$ to A and $C.next$ to B, and re-assigning $A.next$ to C and $B.prevous$ to C.



The starter code includes a **`struct`** for a doubly-linked list's node. Besides the `next` and `previous` pointers, the node has a payload: a string for a word and an integer for the number of times that word occurs in the book. In *problem5.c*, write the code to create a node and the code to build a doubly-linked list.

The starter code also includes a **`print_list()`** function to help with debugging. This function traverses a linked list, printing each node's payload and its `previous` and `next` pointers.

5.2 Alphabetize Words

In Problem 2, you wrote code to convert uppercase letters to lowercase letters. In *problem5.c*, write code that calls that function to convert all letters in a word to lowercase letters (do not copy the **`to_lowercase()`** function into *problem5.c*; we will link to the function in *problem2.c*).

The starter code includes a function to compare two words (you do not need to write this function) but it assumes that both words are completely lowercase.

5.3 Insertion Sort

The *insertion sort* algorithm reads an input and then traverses a sorted list to find the proper location in the sorted list for the input. The input is then inserted into the list at that location.

For this problem, the user will be prompted to enter the name of a book, which will be the filename of a file that contains all of the book's words. All punctuation has already been removed from the files, and each line in the file contains exactly one word. In *problem5.c*, write the code to read the file one line at a time.[3] For each word, convert it to lowercase, and then traverse the list to find the appropriate place for the word. (Note that there will not be a list to traverse when your code reads the first word!) If the word is not in the list then create a node for that word and insert it into the list at the correct location. If there is already a node containing that word, then increment that node's variable that tracks the number of occurrences.

If your program requires more than a few seconds to build the list, there is a bug in your code.

5.4 Respond to a Challenge

After the word list is complete (after you have inserted all words in the file), the user will be prompted to enter the challenge word. In *problem5.c*, write the code to traverse the word list to find that word. (Do not copy the **`is_even()`** function into *problem5.c*; we will link to the function in *problem3.c*.) If the word is not present in the list, output "(word) is not present!" where "(word)" is the challenge word. If the word is in the list then use the number of occurrences recorded in that word's node to find the response word as described in the challenge-and-response rules, and output that word.

If your program does not provide the response word nearly instantaneously, there is a bug in your code.

[3] See §7.5 of Kernighan & Ritchie's *The C Programming Language*, 2nd ed. for **`fopen()`** and **`fclose()`**, and §7.7 for **`fgets()`**.

Building the Executable

Build the executable with the command: `make keyboardlab5` – be sure to fix both errors and warnings.

The Books

As a small "book" to work with as you write your code, we have provided Abraham Lincoln's Gettysburg Address (filename "GettysburgAddress"). This assignment's appendix has a table of the words in the Gettysburg Address and the corresponding response word. We hope that this table will be helpful when you test your code.

For grading, we will use one of these five books:[4]

- Lewis Carroll's *Alice's Adventures in Wonderland* (filename "AliceInWonderland") https://www.gutenberg.org/ebooks/11
- Mary Shelly's *Frankenstein; Or, The Modern Promethius* (filename "Frankenstein") https://www.gutenberg.org/ebooks/84
- Arthur Conan Doyle's *The Lost World* (filename "TheLostWorld") https://www.gutenberg.org/ebooks/139
- Jonathan Swift's *Gulliver's Travels into Several Remote Nations of the World* (filename "GulliversTravels") https://www.gutenberg.org/ebooks/829
- Oscar Wilde's *The Importance of Being Earnest: A Trivial Comedy for Serious People* (filename "TheImportanceOfBeingEarnest") https://www.gutenberg.org/ebooks/844

Turn-in and Grading

When you have completed this assignment, upload *problem1.c*, *problem2.c*, *problem3.c*, *answers4.txt*, and *problem5.c* to Canvas.

This assignment is worth 45 points.

_____ **+4** *problem1.c* produces the specified output without the use of `w`, `W`, `\n`, or `\t`.

_____ **+4** **`is_digit()`** in *problem2.c* determines whether or not a character is a digit, without **`#include`**ing any files, without using any numbers other than 0 and 9, without using a **`switch`** statment, and using at most one **`if`** statement.

[4]For each book, this statement applies: "This eBook is for the use of anyone anywhere in the United States and most other parts of the world at no cost and with almost no restrictions whatsoever. You may copy it, give it away or re-use it under the terms of the Project Gutenberg License included with this eBook or online at www.gutenberg.org. If you are not located in the United States, you'll have to check the laws of the country where you are located before using this ebook." A copy of the Project Gutenberg license is included with the starter code (filename "ProjectGutenbergLicense").

_____ **+4** **`to_lowercase()`** in *problem2.c* converts uppercase letters to lowercase and leaves other characters unchanged, without **`#include`**ing any files, without using a **`switch`** statment, and using at most one **`if`** statement.

_____ **+4** **`is_even()`** in *problem3.c* determines whether a number is even or odd, without using arithmetic.

_____ **+4** **`produce_multiple_of_ten()`** in *problem3.c* has the following, without using arithmetic other than multiplication, and without using the values `5`, `0x5`, `05`, or `0b101`:

- Code to assign the value 5 to the variable `five`
- Code to divide an even number by 2
- Code to subtract 1 from an odd number
- Correct functionality

_____ **+3** Student's answers in *answers4.txt* demonstrate an understanding of the bug in the code and how to correct it.

_____ **+5** Implemented a doubly-linked list in *problem5.c*: **`struct`** `node`, **`create_node()`**, **`insert_after()`**, **`insert_before()`**.

_____ **+2** Implemented **`word_to_lowercase()`** in *problem5.c*.

_____ **+5** Implemented insertion sort in *problem5.c*: **`insert_word()`**.

_____ **+5** **`build_list()`** in *problem5.c* opens a file for reading, reads one line at a time, passes the word that is read to **`insert_word()`**, and closes the file after the last line has been read.

_____ **+5** **`respond()`** in *problem5.c* produces the correct response word in accordance with the specified rules.

Appendix: Gettysburg Address Challenge-and-Response Table

challenge word	*occurrences*	*response word*
a	7	and
above	1	add
add	1	advanced
advanced	1	ago
ago	1	all
all	1	altogether
altogether	1	and
and	6	above
any	1	are
are	3	be
as	1	battlefield
battlefield	1	be
be	2	as
before	1	birth
birth	1	brave
brave	1	brought
brought	1	but
but	2	brave
by	1	can
can	2	but
cannot	3	come
cause	1	civil
civil	1	come
come	1	conceived
conceived	2	civil
consecrate	1	consecrated
consecrated	1	continent
continent	1	created
created	1	dead
dead	3	detract
dedicate	2	created
dedicated	4	continent
detract	1	devotion
devotion	2	dedicated
did	1	died
died	1	do
do	1	earth
earth	1	endure
endure	1	engaged
engaged	1	equal
equal	1	far
far	2	engaged
fathers	1	field
field	1	final
final	1	fitting
fitting	1	for
for	5	freedom
forget	1	forth
forth	1	fought
fought	1	four
four	1	freedom
freedom	1	from
from	2	four
full	1	gave
gave	2	from
god	1	government
government	1	great
great	3	have
ground	1	hallow
hallow	1	have
have	5	increased
here	8	full
highly	1	honored
honored	1	in
in	4	have
increased	1	is
is	3	last
it	5	live
larger	1	last
last	1	liberty
liberty	1	little
little	1	live
live	1	lives
lives	1	living
living	2	live
long	2	lives
measure	1	men
men	2	long
met	1	might
might	1	nation
nation	5	not
never	1	new
new	2	nation
nobly	1	nor
nor	1	not
not	2	nobly
note	1	now
now	1	of
of	5	perish
on	1	or
or	2	of
our	2	on
people	3	poor
perish	1	place
place	1	poor
poor	1	portion
portion	1	power
power	1	proper
proper	1	proposition
proposition	1	rather
rather	2	proper
remaining	1	remember
remember	1	resolve
resolve	1	resting
resting	1	say

challenge word	*occurrences*	*response word*
say	1	score
score	1	sense
sense	1	seven
seven	1	shall
shall	3	struggled
should	1	so
so	3	task
struggled	1	take
take	1	task
task	1	testing
testing	1	that
that	12	resting
the	10	sense
their	1	these
these	2	the
they	3	thus
this	6	testing
those	1	thus
thus	1	to
to	8	that
under	1	unfinished
unfinished	1	upon
upon	1	us
us	3	we
vain	1	war
war	2	us
we	10	this
what	2	war
whether	1	which
which	2	what
who	3	world
will	1	work
work	1	world
world	1	years
years	1	years