CS 211: Computer Architecture, Summer 2021
Programming Assignment 1: Introduction to C (50 points)

Instructor: David Pham

Due: July 7, 2021 at 11:55pm.

Introduction

The goal of this assignment is to help introduce you to C programming. Each part of the assignment will introduce you to a different element of programming in C. You will write a total of 5 programs. Each part of the assignment is detailed below. The programs have to run on iLab machines.

We will not give you improperly formatted files. You can assume all your input files will be in proper format as described.

No cheating or copying will be tolerated in this class. Your assignments will be automatically checked with plagiarism detection tools that are pretty powerful. Hence, you should not look at your friend's code. See CS department's academic integrity policy at:
`https://www.cs.rutgers.edu/academic-integrity/introduction`

First: Twin Prime Numbers (10 Points)

You have to write a program that will read a list of integers from a file and print if each number in the file is a twin prime number or not.

A prime number x is a twin prime number if x - 2 is a prime number or $x + 2$ is a prime number. For example, 3 is a twin prime number since 5 is a prime and 5 is a prime number since 3 is a prime. There are infinitely many twin prime numbers: 3, 5, 7, 11, 13, 17, 19, ...

Input-Output format: Your program will take the file name as input. The file contains a number of positive integers. There is one integer per line. For example, a sample input file "file1.txt" can be:

```
3
14
19
23
31
```

Your output will contain the same number of lines as the number of lines in the input file. Each line will either say *yes* if the corresponding integer is a twin prime or *no* if the corresponding integer is not a twin prime.

```
$./first file1.txt
yes
```

```
no
yes
no
yes
```

We will not give you improperly formatted files. You can assume that the files exist and all the input files are in the proper format as above.

Second: Bit functions (10 Points)

You have to write a program that will read a number followed by a series of bit operations from a file and perform the given operations sequentially on the number. The operations are as follows:

set(x, n, v)	sets the nth bit of the number x to v
comp(x, n)	sets the value of the nth bit of x to its complement (1 if 0 and 0 otherwise)
get(x, n)	returns the value of the nth bit of the number x

The least significant bit (LSB) is considered to be index 0.

Input format: Your program will take the file name as input. The first line in the file provides the value of the number to be manipulated. For the function calls this number can be considered x. This number should be considered an unsigned short. The following lines will contain the operations to manipulate the number. The format of the operations will always be the command followed by 2 numbers with tabs in between each part. For the set(x, n, v) command, the value of the second number will always be either 0 or 1. For the comp(x, n) and get(x, n) command the value of the second number will always be 0 and can be ignored.

Output format: Your output will be the resulting value of the number x after each operation, each on a new line. The get(x, n) function should only print the value of the nth bit as the value of x does not change.

Example Execution:

For example, a sample input file "file1.txt" contains:

```
5
get	0	0
comp	0	0
set	1	1
```

The result of the sample run is:

```
$./second file1.txt
1
4
6
```

Third: Bit Count functions (10 points)

In this part, you have to determine the parity of a number and the amount of 1-bit pairs present in the number. Parity refers to whether a number contains an even or odd number of 1-bits. 1-bit pairs are defined by 2 adjacent 1's without overlap with other pairs.

For example the number 7 has the binary sequence 111 and is considered to contain 1 pair while the sequence 101101111 has 3 pairs.

Input format: This program takes a single number as an argument from the command line. This number should be considered as an unsigned short.

Output format: Your program should print either "Even-Parity" if the input has an even amount of 1 bits and "Odd-Parity" otherwise, followed by a tab. Your program should then print the number of 1-bit pairs present in the number followed by a new line character.

Example Execution:

Some sample runs and results are:

```
$./third 12
Even-Parity   1

$./third 31
Odd-Parity    2
```

Fourth: Ordered Linked List (10 points))

In this part, you have to implement operations on a linked list such that it maintains a list of integers in sorted order. For example, if a list already contains 2, 5, and 8 then 1 will be inserted at the start of the list, 3 will be inserted between 2 and 5, and 10 will be inserted at the end. Duplicates should be ignored.

Input format: This program takes a file name as an argument from the command line. The file contains successive lines of input. Each line contains a string, either INSERT or DELETE, followed by a space and then an integer. For each of the lines that strts with INSERT, your program should insert that number in the linked list in sorted order if it is not already there. Your program should not insert any duplicate values. If the line starts with a DELETE, your program should delete the value if it is present in the linked list.

Output format: After every INSERT, your program should print the contents of the linked list. The values should be printed in a singled line separated by a single space. You should print EMPTY if the linked list is empty. After every DELETE your program should print SUCCESS if the item was deleted or FAILED if the item was not deleted. There should be no leading or trailing white spaces in each line of the output. You should print ERROR (and nothing else) if the files does not exist.

Example Execution: Let's assume we have a text file with the following contests:

file3.txt:

```
INSERT 2
DELETE 3
INSERT 10
INSERT 5
```

Then the results will be: $./fourth file3.txt

```
2
FAILED
2 10
2 5 10
```

Fifth: Hash Table (10 Points)

In this part, you will implement a hash table containing integers. The hash table has 10,000 buckets. An important part of a hash table is collision resolution. In this assignment, we want you to use chaining with a linked list to handle a collision. This means that if there is a collision at a particular bucket then you will maintain a linked list of all values stored at that bucket. For more information about chaining, see *http://research.cs.vt.edu/AVresearch/hashing/openhash.php*.

A hash table can be implemented in many ways in C. You must find a simple way to implement a hash table structure where you have easy access to the buckets through the hash function. As a reminder, a hash table is a structure that has a number of buckets for elements to "hash" into. You will determine where the element falls in the table using the hash function.

You must not do a lienar search of the 10,000 element array. We will not award any credit for O(n) time implementation of searches or insertions in the common case.

For this problem, you have to use the following hash function: key modulo the number of buckets.

Input format: This program takes a file name as argument from the command line. The file contains successive lines of input. Each line contains a character, either 'i' or 's', followed by a tab and then an integer. For each line that starts with 'i', your program should insert that number in the hash table if it is not present. If the line starts with a 's', your program should search the hash table for that value.

Output format: Your program prints two counts: (1) the number of insertions where collision occurred, and (2) the number of successful searches. In the first line your program should print the number of collisions, i.e. during insertion if the bucket already had some data (may not be the same value) then you need to count that as one collision. In the next line, your program should print the number of searches where the value was present in the hash table. You can assume that the program inputs will have proper structure.

Example Execution: Let's assume we have a text file with the following contests:

file2.txt:

```
i 10
i 12
```

s 10
i 10010
s 5
s 10010

Then the results will be:
$./fifth file2.txt
1
2

Structure of your submission folder

All files must be included in the `pa1` folder. The `pa1` directory in your tar file must contain 5 subdirectories, one for each each of the parts. The name of the directories should be named first through fifth (in lower case). Each directory should contain a c source file, a header file (if you use it) and a Makefile. For example, the subdirectory first will contain, first.c, first.h (if you create one) and Makefile (the names are case sensitive).

```
pa1
|- first
   |-- first.c
   |-- first.h (if used)
   |-- Makefile
|- second
   |-- second.c
   |-- second.h (if used)
   |-- Makefile
|- third
   |-- third.c
   |-- third.h (if used)
   |-- Makefile
|- fourth
   |-- fourth.c
   |-- fourth.h (if used)
   |-- Makefile
|- fifth
   |-- fifth.c
   |-- fifth.h (if used)
   |-- Makefile
```

Submission

You have to e-submit the assignment using Sakai. Your submission should be a tar file named `pa1.tar`. To create this file, put everything that you are submitting into a directory (folder) named `pa1`. Then, `cd` into the directory containing `pa1` (that is, `pa1`'s parent directory) and run the following command:

```
tar cvf pa1.tar pa1
```

To check that you have correctly created the tar file, you should copy it (`pa1.tar`) into an empty directory and run the following command:

```
tar xvf pa1.tar
```

This should create a directory named `pa1` in the (previously) empty directory.

The `pa1` directory in your tar file must contain 3 subdirectories, one each for each of the parts. The name of the directories should be named first through third (in lower case). Each directory should contain a c source file, a header file (if necessary) and a make file. For example, the subdirectory first will contain, first.c, first.h and Makefile (the names are case sensitive).

AutoGrader

We provide the AutoGrader to test your assignment. AutoGrader is provided as autograder.tar. Executing the following command will create the autograder folder.

```
$tar xvf autograder.tar
```

There are two modes available for testing your assignment with the AutoGrader.

First mode

Testing when you are writing code with a `pa1` folder

(1) Lets say you have a `pa1` folder with the directory structure as described in the assignment.

(2) Copy the folder to the directory of the autograder

(3) Run the autograder with the following command

$python auto_grader.py

It will run your programs and print your scores.

Second mode

This mode is to test your final submission (i.e, pa1.tar)

(1) Copy pa1.tar to the auto_grader directory

(2) Run the auto_grader with pa1.tar as the argument.

The command line is

$python auto_grader.py pa1.tar

Scoring

The autograder will print out information about the compilation and the testing process. At the end, if your assignment is completely correct, the score will something similar to what is given below.

```
You scored
5.0  in  second
5.0  in  fifth
5.0  in  fourth
5.0  in  third
5.0  in  first
Your TOTAL SCORE =  25.0 /25
Your assignment will be graded for another 25 points with test cases not given to you
```

Grading Guidelines

This is a large class so that necessarily the most significant part of your grade will be based on programmatic checking of your program. That is, we will build the binary using the Makefile and source code that you submitted, and then test the binary for correct functionality against a set of inputs. Thus:

- **You should not see or use your friend's code either partially or fully. We will run state of the art plagiarism detectors. We will report everything caught by the tool to Office of Student Conduct**.
- You should make sure that we can build your program by just running `make`.
- You should test your code as thoroughly as you can. For example, programs should *not* crash with memory errors.
- Your program should produce the output following the example format shown in previous sections. Any variation in the output format can result **in up to 100% penalty**. Be especially careful to not add extra whitespace or newlines. That means you will probably not get any credit if you forgot to comment out some debugging message.

Be careful to follow all instructions. If something doesn't seem right, ask on the discussion forum.