EC327 - Spring 2025 - Lab 2

Background

This lab assignment will build up your comfort in implementing C functions based on their defined headers. It will emphasize use of structs, arrays, and iteration.

All of the problems on this assignment will require solutions to be implemented in C.

Important

Please make sure your code compiles and runs as intended on the engineering grid. Code that does not compile will NOT be graded and will receive a 0.

Submission Instructions

You will submit this assignment as single .zip file with the following name:

<first-name>_<last-name>-lab2.zip

So for example:

ed_solovey-lab2.zip.

Your zip file should contain one file per problem in this assignment. Those files should be named like this:

<first-name>_<last-name>-lab2-<problem-number>.cpp
So for example, my problem one file would be:

ed_solovey-lab2-1.cpp

Each of the problems below will have specific instructions around what the text file for that problem should look like.

IMPORTANT: Your lab will be evaluated by an auto-grader and the presence of a **<first-name>_<last-name>-lab2-<problem-number>.cpp** file for each problem is required for the auto-grader to run. Each problem below provides you with a stub to use as a starting point for each of your **<first-name>_<last-name>-lab2-<problem-number>.cpp** submissions.

You are welcome to submit assignments before completing all of the problems, but will need to include these stubs to get any feedback from the auto-grader.

You may submit your zip files via Gradescope as many times as you would like up until the deadline for this lab assignment. Your most recent submission at the time of the deadline for this lab will be the one used to determine your grade. No late submissions will be accepted.

Note on Collaboration

You are welcome to talk to your classmates about the assignment and discuss high level ideas. However, all code that you submit must be your own. We will run code similarity tools against your submissions and will reach out with questions if anything is flagged as suspicious.

The closed book exams for this class will mostly cover material very similar to what you do in the lab and homework assignments. The best way to prepare for the exams is to do the assignments on your own and internalize the learnings from that process. Cheating on the assignments will very likely result in you not doing well on the exams.

Problem 1 - Diagonal Stars (24 points)

Submission Instructions

Your solution to this problem should contribute a single **cpp** file to your overall **lab2** zip. The **cpp** file should follow the following format:

<first-name>_<last-name>-lab2-1.cpp

So for example, my file would be:

ed_solovey-lab2-1.cpp

Actual Problem

Create a header file called **lab2_problem1.h** with the following contents:

```
#ifndef LAB2_PROBLEM1_H
#define LAB2_PROBLEM1_H
#include <string>

/**
* Parameter n represents the length of the sides of a square.  The
* goal of this function is to construct a string that puts
* the '-' character along both diagonals of that square.  For example,
* diagonalStars(5) should return a string that when printed
* with std::cout << diagonalStars(5) << std::endl; looks like:
* -   -
*  - -
*   -
*  -  -
* - -  -
*
* when called with the number 4, the string should look like:
* -  -
*  --
*  --
* -  -
*
* when printed.
```

```
*
* There should NOT be a trailing newline at the very end.
*
* If the function is called with 0 or a negative number, it should
* return the string "Not Supported".
*/
std::string diagonalStars(int n);

#endif //LAB2_PROBLEM1_H
```

Your **<first-name>_<last-name>-lab2-1.cpp** should implement the above **diagonalStars** function.

You can test your implementation from a **main** function or from tests you set up otherwise. Your submission will be tested against multiple inputs and you should convince yourself that your solution works for the general case.

As mentioned above, the autograder for this lab assignment requires that the **cpp** file for each problem be included in the submission. You are welcome to start with this stub file for this problem:

```
#include "lab2_problem1.h"

std::string diagonalStars(int n) {
  return "";
}
```

Problem 2 - DigitSums (24 points)

Submission Instructions

Your solution to this problem should contribute a single **cpp** file to your overall **lab2** zip. The **cpp** file should follow the following format:

<first-name>_<last-name>-lab2-2.cpp

So for example, my file would be:

ed_solovey-lab2-2.cpp

Actual Problem

Create a header file called **lab2_problem2.h** with the following contents:

```
#ifndef LAB2_PROBLEM2_H
#define LAB2_PROBLEM2_H

/**
* @return the sum of the following three things:
*  1) sum of all the digits in the number
*  2) the second to last digit of the number (0 if the number only has 1 digit)
*  3) the second digit of the number (0 of the numer only has 1 digit)
*
*  The method should work for positive and negative numbers.
*
*  Examples:
*  digitSums(178) returns 30 because:
*   1) 1 + 7 + 8 = 16
*   2) second to last digit is 7
*   3) second digit is 7
*
*  digitSums(-9834) returns 35 because:
*   1) 9 + 8 + 3 + 4 = 24
*   2) second to last digit is 3
*   3) second digit is 8
*
*  digitSums(8) returns 8 because:
*   1) single digit is 8 and there is no second to last or second digit
*/
int digitSums(int number);

#endif //LAB2_PROBLEM2_H
```

Your **<first-name>_<last-name>-lab2-2.cpp** should implement the above **digitSums** function.

You can test your implementation from a **main** function or from tests you set up otherwise. Your submission will be tested against multiple inputs and you should convince yourself that your solution works for the general case.

As mentioned above, the autograder for this lab assignment requires that the **cpp** file for each problem be included in the submission. You are welcome to start with this stub file for this problem:

```
#include "lab2_problem2.h"

int digitSums(int number) {
  return 0;
}
```

Problem 3 - All Substrings Without Char (25 points)

Submission Instructions

Your solution to this problem should contribute a single **txt** file to your overall **lab2** zip. The **txt** file should follow the following format:

<first-name>_<last-name>-lab2-3.cpp

So for example, my file would be:

ed_solovey-lab2-3.cpp

Actual Problem

```
#ifndef LAB2_PROBLEM3_H
#define LAB2_PROBLEM3_H
#include <string>

/**
* Helper struct to pass along an array of characters representing a
* string as well as their size.
*/
struct CharArrayWithSize {
   int size;
   const char* charArray;
};
```

```
/**
* This function should return a string representation of substrings of
* all lengths of the  string represented by input.charArray.
*
* The substrings should be separated by the newline character \n.
*
* The substrings should be ordered first by length (shorters strings first),
* and then by their appearance in the input string (leftmost first).
*
* Substrings that contain the excludeChar character at any location,
* should be excluded from the resulting string.  Note that the exclusion should
* happen after the substring has been computed not before (this will be obvious
* from the examples below).
*
* Identical substrings should be repeated.  For example, in the string "hello",
* both single character substrings, 'l', should be included.
*
* The last substring should also have a trailing newline character.
*
* For example, if called with ({5, "hello"}, 'a'), the function should return:
* "h\ne\nl\nl\no\nhe\nel\nll\nlo\nhel\nell\nllo\nhell\nello\nhello\n", which when
* printed should look like:
*
* h
* e
* l
* l
* o
* he
* el
* ll
* lo
* hel
* ell
* llo
* hell
* ello
* hello
*
* And if called with ( {8, "computer"}, 'p' ) the function should return:
*
* c
* o
* m
* u
* t
* e
```

```
* r
* co
* om
* ut
* te
* er
* com
* ute
* ter
* uter
*
* When called with an input that should return no substrings at all, the function
* should return an empty string, "".
*
* Note that the return type of the function is std::string, a C++ style string,
* while the input is an
* array of characters (equivalent to char*), which is a C-style string.  We will do
* a lot more with
* C-style strings later, but for this problem please be aware of the following
things:
*
*  1) you can access C-style strings like you would any other array
*  2) You can concatenate a C-style string to a C++ string with cString +=
* cStyleString;
*  3) For above to work as expected, the cStyleString should have a null character,
* '\0', as its last character.  So, when building the C-style string, allocate one
* extra space
*  in the array and set the last element to '\0'.
*/
std::string allSubstringsWithoutChar(CharArrayWithSize input, char excludeChar);

#endif //LAB2_PROBLEM3_H
```

Your **<first-name>_<last-name>-lab2-3.cpp** should implement the above **allSubstringsWithoutChar** function.

You can test your implementation from a **main** function or from tests you set up otherwise. Your submission will be tested against multiple inputs and you should convince yourself that your solution works for the general case.

As mentioned above, the autograder for this lab assignment requires that the **cpp** file for each problem be included in the submission. You are welcome to start with this stub file for this problem:

```
#include "lab2_problem3.h"

std::string allSubstringsWithoutChar(CharArrayWithSize input,
char excludeChar){
  return "";
}
```

Problem 4 - Greatest Common Factor (27 points)

Submission Instructions

Your solution to this problem should contribute a single **cpp** file to your overall **lab2** zip. The **cpp** file should follow the following format:

<first-name>_<last-name>-lab2-4.cpp

So for example, my file would be:

ed_solovey-lab2-4.cpp

Actual Problem

Create a header file called **lab2_problem4.h** with the following contents:

```
#ifndef LAB2_PROBLEM4_H
#define LAB2_PROBLEM4_H

/**
* Helper struct to represent a compound response from the
* identifyGCF function, containing the greatest common
* factor as well as the total number of factors.
*/
struct GreatestCommonFactorResult {
   int greatestCommonFactor;
   int numberOfCommonFactors;
```

```
};

/**
* identifyGCF returns the greatest common factor as
* well as the total number of common factors for
* five input integers.
*
* A common factor is an integer that evenly divides
* all five numbers without leaving a remainder.
*
* The greatest common factor is the largest common
* factor that the five input integers share.
*
* Negative numbers should be treated the same as their
* positive counterparts.
*
* Any number divides zero, so if zero is present
* in the input it does not eliminate any potential
* common factors.  However, if all input numbers are zero
* the response should be {0, 0}.
*
* Some examples:
* identifyGCF({ 24, 96, 64, 240, 800 }) returns {8,4}
* identifyGCF({ 14, 42, -56, 98, -28 }) returns {14,4}
* identifyGCF({ 14, 0, -26, 13, 39 }) returns {1,1}
*/
GreatestCommonFactorResult identifyGCF(int input[5]);

#endif //LAB2_PROBLEM4_H
```

Your **<first-name>_<last-name>-lab2-4.cpp** should implement the above **identifyGCF** function. You can test your implementation from a **main** function or from tests you set up otherwise. Your submission will be tested against multiple inputs and you should convince yourself that your solution works for the general case.

As mentioned above, the autograder for this lab assignment requires that the **cpp** file for each problem be included in the submission. You are welcome to start with this stub file for this problem:

```cpp
#include "lab2_problem4.h"

GreatestCommonFactorResult identifyGCF(int input[5]) {
    return {1,1};
}
```