EC327 - Spring 2025 - Homework 3

Background

This homework assignment will continue to build our C/C++ foundations and will primarily focus on memory management. All of the problems on this assignment will require solutions to be implemented in C/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>-hw3.zip

So for example:

ed_solovey-hw3.zip.

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

<first-name>_<last-name>-hw3-<problem-number>.cpp
So for example, my problem one file would be:

ed_solovey-hw3-1.cpp

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

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 on the 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 - String Concatenation (15 points)

Submission Instructions

Your solution to this problem should contribute a single **cpp** file to your overall **hw3** zip. The **cpp** file should follow the following format:

<first-name>_<last-name>-hw3-1.cpp

So for example, my file would be:

ed_solovey-hw3-1.cpp

Actual Problem

Create a header file called **hw3_problem1.h** with the following contents:

```
#ifndef HW3_PROBLEM1_H
#define HW3_PROBLEM1_H

/**
* Concatenates two C-style strings into a single C-style string.
*
* @param first the input string that will appear on the left of the
* concatenated result
* @param second the input string that will appear on the right of the
* concatenated result
* @return pointer to a C-style sring that holds the results of the
* concatenation.
*/
char *concatenate(const char *first, const char *second);

#endif //HW3_PROBLEM1_H
```

Your **<first-name>_<last-name>-hw3-1.cpp** should implement the above **concatenate** function.

REQUIREMENT: For this particular problem you should NOT use any built-in string manipulation helper functions like **strcpy** and **strcat**. Your solution should only involve character manipulation of memory allocated for C-style strings.

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.

An example **main** function to test your implementation:

```
#include <stdio.h>
#include <stdlib.h>

#include "hw3_problem1.h"

int main(void)
{
   char* leftInput = "Hi there! ";
   char* rightInput = "Classmates! ";

   char* result = concatenate(leftInput, rightInput);

   printf("result: %s\n", result);
   free(result);

   return 0;
}
```

This should output:

```
result: Hi there! Classmates!
```

Problem 2 - Reverse Odd (15 points)

Submission Instructions

Your solution to this problem should contribute a single **cpp** file to your overall **hw3** zip. The **cpp** file should follow the following format:

<first-name>_<last-name>-hw3-2.cpp

So for example, my file would be:

ed_solovey-hw3-2.cpp

Actual Problem

Create a header file called **hw3_problem2.h** with the following contents:

```
#ifndef HW3_PROBLEM2_H
#define HW3_PROBLEM2_H
#include <stdbool.h>

struct ReverseResult {
   bool success;
   char** reversedStrings;
};

/**
* Reverses all of the input strings as long as they have an odd number of
* characters.  If they are all odd, the resulting struct should have
* success=true and reversedStrings pointing at the resulting array of C-style
* strings.
*
* If any of the strings have an even number of characters, the resulting
* struct should have success=false and reversedStrings=NULL.
*
*
* @param inputStrings an array of input strings of length inputStringsLength
* @param inputStringsLength number of input strings
* @return success=true plus reversed strings as long as all strings have an
* odd number of characters, and
* success=false plus NULL if any of the strings have an even number of
* characters.
*/
ReverseResult reverseOdd(const char** inputStrings, int inputStringsLength);

#endif //HW3_PROBLEM2_H
```

Your **<first-name>_<last-name>-hw3-2.cpp** should implement the above **reverseOdd** function.

REQUIREMENTS:

- For this particular problem you should NOT use any built-in string manipulation helper functions like **strcpy** and **strcat**. Your solution should only involve character manipulation of memory allocated for C-style strings.
- If you have already allocated memory for the result when you determine that one of the inputs has an even number of characters and you are going to return **NULL** for the **reversedStrings** part of the response, you must use **free** to free up that memory - otherwise your code will have a memory leak.

Problem 3 - Linked Lists (15 points)

Submission Instructions

Your solution to this problem should contribute a single **cpp** file to your overall **hw3** zip. The **cpp** file should follow the following format:

<first-name>_<last-name>-hw3-3.cpp

So for example, my file would be:

ed_solovey-hw3-3.cpp

Actual Problem

Create a header file called **hw3_problem3.h** with the following contents:

```
#ifndef HW3_PROBLEM3_H
#define HW3_PROBLEM3_H
#include <stddef.h>
```

```
#include <stdio.h>

/**
* A node in a linked-list storing integer values.
*/
struct Node {
   int value;
   struct Node* next;
};

/**
* Helper function for debugging current state of a linked-list
* pointed to by head.
*/
static void printList(struct Node* head) {
   printf("Printing List:\n");
   while (head != NULL) {
       printf("%d -> ", head->value);
       head = head->next;
   }
   printf("\n");
}

/**
* Inserts newValue into the appropriate location in the linked list
* pointed to by head.
*
* The assumption is that the input linked list is sorted in ascending
* order and the behavior of this function is undefined in the case
* that it is not.
*
* This function should behave as expected even if head is pointing to
* NULL - it should return a pointer to a linked list of one node
* with value of newValue.
*
* @param pointer to the head of a linked-list that is sorted in
* ascending order
* @param newValue the new integer value to be inserted into the
* sorted linked list
*
* @return pointer to the head of the resulting linked-list.  This very
* well may point to the same memory location that head is pointing to.
*/
Node* insertInOrder(Node* head, int newValue);

#endif //HW3_PROBLEM3_H
```

Your **<first-name>_<last-name>-hw3-3.cpp** should implement the above **insertInOrder** function.

Make sure that your implementation covers all edge cases. As an example of expected behavior:

```
#include <stdio.h>
#include "hw3_problem3.h"

int main(void)
{
   struct Node fourth = {17, NULL};
   struct Node third = {12, &fourth};
   struct Node second = {9, &third};
   struct Node first = {3, &second};

   printList(&first);

   insertInOrder(&first, 13);

   printList(&first);

   return 0;
}
```

Should return:

```
Printing List:
3 -> 9 -> 12 -> 17 ->
Printing List:
3 -> 9 -> 12 -> 13 -> 17 ->
```

Problem 4 - insertInOrder Signature Explanation (10 points)

Submission Instructions

Your solution to this problem should contribute a single **txt** file to your overall **hw3** zip. The **txt** file should follow the following format:

<first-name>_<last-name>-hw3-4.txt

So for example, my file would be:

ed_solovey-hw3-4.txt

Actual Problem

The signature of the **insertInOrder** function in the above problem, Problem 3 was:

```
Node* insertInOrder(Node* head, int newValue);
```

What would be the problem with simplifying the signature to be:

```
void insertInOrder(Node* head, int newValue);
```

and having **head** always point at the head of the resulting linked list?

Once you have identified the problem, can you think of another way that we could have modified the signature to address the concern?

What are the trade-offs between addressing it the way we have in the problem, and the alternate way that you have identified?

Problem 5 - Change Amounts (45 points)

Submission Instructions

Your solution to this problem should contribute a single **cpp** file to your overall **hw3** zip. The **cpp** file should follow the following format:

<first-name>_<last-name>-hw3-5.cpp

So for example, my file would be:

ed_solovey-hw3-5.cpp

Actual Problem

Create a header file called **hw3_problem5.h** with the following contents:

```
#ifndef HW3_PROBLEM5_H
#define HW3_PROBLEM5_H
#include <stdio.h>

/**
* Enum representing all of the coins in circulation.
*/
typedef enum {
   PENNY = 1,
   NICKEL = 5,
   DIME = 10,
   QUARTER = 25
} Coin;

/**
* Convenience for passing around an array of coins along with their
size.
*/
struct CoinArray {
   Coin* coins;
   int size;
```

```c
};

/**
* Convenience for passing around multiple coin arrays along
* with the number of total arrays.
*/
struct CoinArrays {
   struct CoinArray* arrays;
   int size;
};

/**
* Comparator for Coins so that we can use qsort to sort them
* in ascending order.
*/
static int compareCoins(const void* a, const void* b) {
   return *(Coin*)a - *(Coin*)b;
}

/**
* Comparator for CoinArrays so that we can use qsort to sort
* the in ascending order by number of Coins in the array.
*/
static int compareCoinArrays(const void* a, const void* b) {
   return (*(struct CoinArray*)a).size - (*(struct CoinArray*)b).size;
}

/**
* Utility for printing a collection of coin arrays.
*/
static void printCoinArrays(struct CoinArrays coinArrays) {
   printf("Have %d arrays:\n", coinArrays.size);
   printf("\n");

   for (int i = 0; i < coinArrays.size; i++) {
       printf("Array of size %d:\n", coinArrays.arrays[i].size);
       for (int j = 0; j < coinArrays.arrays[i].size; j++) {
           printf("%d, ", coinArrays.arrays[i].coins[j]);
       }
       printf("\n");
   }
```

```
}

/**
* Identify all possible ways to use any number of Coins to
* combine to total the totalCents amount.
*
* totalCents is assumed to be a non-negative number.
*
* Requirements:
* 1) resulting combinations should be ordered in ascending order
* with respect to number of coins
* 2) the coins in each combination should be ordered in ascending order
*
* Examples:
*
* possibleChangeAmounts(7) should return the following CoinArrays:
* 1,1,5
* 1,1,1,1,1,1,1
*
* possibleChangeAmounts(12) should return the following CoinArrays:
* 1, 1, 10
* 1, 1, 5, 5
* 1, 1, 1, 1, 1, 1, 1, 5
* 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
*
* possibleChangeAounts(25) should return the following CoinArrays:
*
* 25
* 5, 10, 10
* 5, 5, 5, 10
* 5, 5, 5, 5, 5
* 1, 1, 1, 1, 1, 10, 10
* 1, 1, 1, 1, 1, 5, 5, 10
* 1, 1, 1, 1, 1, 5, 5, 5, 5
* 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 10
* 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 5, 5
* 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 10
* 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 5
* 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5
* 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1
```

```
 *
 */
CoinArrays possibleChangeAmounts(int totalCents);

#endif //HW3_PROBLEM5_H
```

Your **<first-name>_<last-name>-hw3-5.cpp** should implement the above **possibleChangeAmounts** function.

Requirements:

- Your solution for this problem must use recursion.
- Your solution should return each combination just once regardless of order (order does not matter). For example, if asked for combinations to total 6 cents, do not return {1,5} and {5,1}. You should only return one of those two combinations.
- resulting combinations should be ordered in ascending order with respect to the number of coins. This will help us auto-grade your submissions
- The coins in each resulting combination should be ordered in ascending order. This will also help us auto-grade your submissions.
-

Hints:

- **hw3_problem5.h** provides you with structs and helpers for this implementation.
  - Familiarize yourselves with the **Coins** enum (just 4 coins to worry about) and the **CoinArray** and **CoinsArray** structs.
  - **compareCoins** will allow you to use qsort to sort a **CoinArray**. This in turn will help you make sure that you are not over-including combinations that differ only in order.
  - **compareCoinArrays** will allow you to use qsort to sort a **CoinsArray**. This will help you satisfy the requirement that the resulting combinations be ordered in ascending number of coins in the combination.
  - **printCoinArrays** will help you debug your work as you are making progress.
- I suggest that
  - you have a helper function that adds a single coin to an in-progress potential solution.
  - A helper function that checks whether a solution you are exploring is already contained in the collection of solutions you have found, so that you can exclude it
  - A helper function that adds a found solution to the collection of solutions you have already found

    - A helper function that performs the core recursion, and have your **possibleChangeAmounts** call into this helper after setting up all initial conditions.
- Think about the base case(s) for your recursion and the recursive step you should take at each iteration.
- Try your in-progress solution on some very simple examples, e.g. possibleChangeAmounts(2) and possibleChangeAmounts(5) first. Make sure those behave as expected and they try out some higher numbers.