CS 137: Assignment #3

Due on Friday, Oct 3, 2025, at 11:59 PM

Submit all programs using the Marmoset Submission and Testing Server located at
https://marmoset.student.cs.uwaterloo.ca/

Victoria Sakhnini

Fall 2025

Notes:

- Use the examples to guide your formatting for your output. Remember to terminate your output with a newline character unless the question asks something different.
- **You must solve all problems using recursion. You must NOT use loops.**
- For this and all future assignments, I strongly recommend you solve the extra practice problems in the course textbook before working on the assignment.
- You must NOT use the MATH Library

Problem 1

The Universal Product Code (UPC) is a globally recognized barcode used for tracking trade items in retail stores. The UPC-A barcode has a scannable strip of black bars and white spaces above a sequence of 12 numerical digits. Only numerical digits are permitted; no letters or other characters can appear on a UPC-A barcode. A built-in mechanism detects errors, such as swapping adjacent digits or mistyping a single digit.

A 12-digit UPC-A barcode consists of 11 digits plus a check digit at the end. To verify the check digit, follow these steps:

1. Sum the digits in the odd positions (starting from the left) and multiply the sum by 3. The most left digit is in position 1.
2. Sum the digits in the even positions, excluding the 12th digit.
3. Add the results from steps 1 and 2.
4. Find the remainder when the result from step 3 is divided by 10. If the remainder is 0, the check digit is 0. Otherwise, subtract the remainder from 10 to get the check digit.
5. Verify that the calculated check digit matches the 12th digit of the UPC-A barcode. If they match, the barcode is valid; otherwise, it is not.

For example, consider the UPC-A number 725272730706:

1. The odd-position digits are 7, 5, 7, 7, 0, 0. Their sum, multiplied by 3, is (7 + 5 + 7 + 7 + 0 + 0) * 3 = 78.
2. The even-position digits are 2, 2, 2, 3, 7 (excluding the 12th digit). Their sum is 2 + 2 + 2 + 3 + 7 = 16.
3. Adding the results from steps 1 and 2 gives 78 + 16 = 94.
4. The remainder of 94 divided by 10 is 4, so the check digit is 10 - 4 = 6.
5. The calculated check digit (6) matches the 12th digit of the UPC-A barcode, so it is valid.

For this question, the marmoset basic tests ("public" tests) will use the asserts from the provided `main` function. You should modify `main` by adding additional tests (we suggest adding your tests "below" ours).

Note:

- In C, a twelve-digit number will exceed `INT_MIN`. Therefore, use `long long int` instead.
- While UPC numbers often have leading zeros, for this question, you can assume the first digit (on the left) will be between 1 and 9.
- Implement the `bool validate_upc(long long upc)` function.
- You are provided with an initial `upc.c` file to solve this problem

You are to submit the file `upc.c` containing only your implemented function and any additional functions you defined (that is, you must delete the test cases portion and the `main` function). However, **you must keep the required included libraries.**

Here is a sample main function and tests:

```c
int main(void) {
  assert(validate_upc(725272730706));
  assert(!validate_upc(725272730704));
  assert(!validate_upc(640754291001));
  assert(validate_upc(640754291004));
  assert(validate_upc(693508003156));
  assert(validate_upc(123456789012));
  assert(!validate_upc(123456789013));
  return 0;
}

```

Problem 2

For this question, you will determine if a given string is a magic sequence. A magic sequence is defined as follows:

- magic string = S
- S → 1G5
- G → 2HG|3|4G
- H → 5|6H

Notes:

- Let x → y mean that y can replace x.
- The | operator means that either option is valid.
- The numbers 1, 2, 3, 4, 5, and 6 are integers to be read one by one from standard input.
- You can therefore check if a sequence is a magic string by applying the aforementioned rules to convert the sequence to S.

Example:
Feed in 1, 2, 3, 2, 2, 5 integer-by-integer. The transformation is as follows:
12535 → 12H35 → 12HG5 → 1G5 → S
Therefore, 1, 2, 5, 3, 5 is a magic sequence.

Complete the program `magicseq.c` which can determine if a given sequence of integers is a magic sequence. Submit the whole solution after deleting the `main` function but keep the included library.

You are to define the function `int magicSequence()` that reads integers between 1 – 6 inclusive and returns 1 if the input is a magic sequence, otherwise it returns 0.

```
int main(void){
    printf("%d\n", magicSequence());
}
```

Examples:

```
1 2 6 5 3 5 ?
1

1 2 6 3 5 !
0
```

Problem 3

Create a C program `treeprint.c` that includes the function `void tree(int n);` which takes an integer `n>0` and prints a tree ASCII art picture.

- The height of the tree is `2n+1`.
- The tree is made of an isosceles triangle with a height of `n+1` and a base of `2n+1`.
- The tree trunk is a rectangle with a length of `n` and a width of half of `n` if `n` is an even number or half of `n+1` if `n` is an odd number.
- The tree should be vertically symmetric. If the width of the trunk is an even number, we would add 1 to the width. Please note that when the width is 1, you only need to print one single vertical line as the trunk.
- `*` characters should print the top of the triangles.
- The sides of the triangle should be printed with `+` characters.
- `|` characters should print the trunk of the tree.
- Each line starts with a **.** character and ends with a **.** character except the base of the triangle, which begins with `*` character and ends with `*` character.

You are to submit this file containing <u>only</u> your implemented function and any additional functions you defined (that is, you <u>must delete</u> the test cases portion <u>and</u> the `main` function). However, <u>**you must keep the required included libraries.**</u>

Examples:

Calling `tree(1)` prints:

```
.*.
*+*
.|.
```

Calling `tree(2)` prints:

```
. * .
.+ +.
*+++*
. | .
. | .
```

Calling `tree(4)` prints:

```
.   *   .
.  + +  .
. +   + .
.+     +.
*+++++++*
.  | |  .
.  | |  .
.  | |  .
.  | |  .
```

Calling `tree(5)` prints:

```
.    *    .
.   + +   .
.  +   +  .
. +     + .
.+       +.
*+++++++++*
.   | |   .
.   | |   .
.   | |   .
.   | |   .
.   | |   .
```