CS 137: Assignment #4

Due on Friday, Oct 10, 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.
- Integers should be read using `scanf`.
- For this and all future assignments, I strongly recommend that you solve the extra practice problems in the course notes before working on the assignment.
- For this assignment, you may use any content covered until the end of chapter 6.
- **You must create more tests. Examples don't cover all the cases.**
- **You must NOT use a variable-length array.**
- **You are free to decide whether to use loops or recursion to solve any of the questions.**

Problem 1

Write the function `long long removekdigits(long long int n, int k);` that returns an integer like n after removing the first k largest digits from n, and returns the **smallest** integer possible from the resulting digits. Return 0 if no digits remain after removing k digits. Assume k<= length of n.

- **You must NOT use a variable length array (VLA). Using VLA will result in receiving a zero on this question.**
- **Assume n>=0**

Submit `kdigits.c` file containing only your implemented functions (that is, you must delete the test cases portion/the `main` function). However, **you must keep the required included libraries.**

Sample Test program (make sure you test your solution with more test cases)

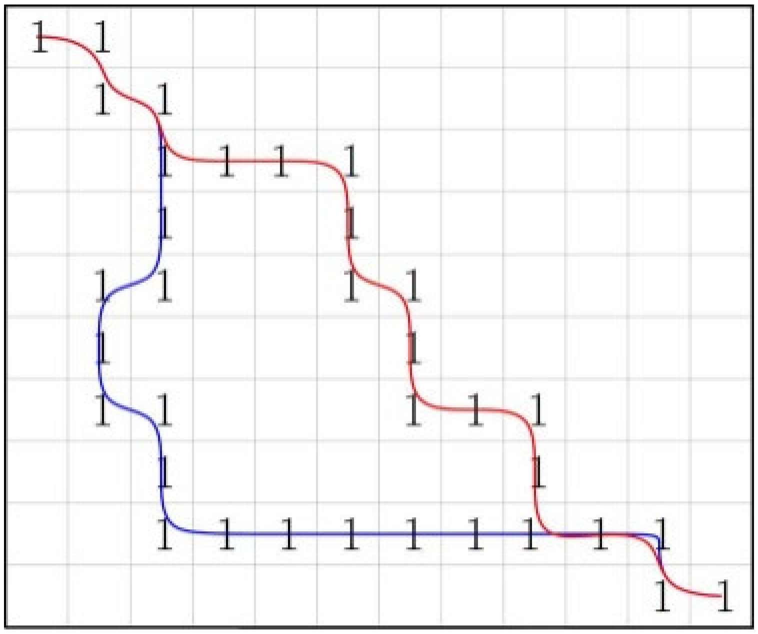
```
int main(void) {

    assert(removekdigits(345267,3)==234);
    assert(removekdigits(0,0)==0);
    assert(removekdigits(9,1)==0);
    assert(removekdigits(871,0)==178);
    assert(removekdigits(9898989,2)==88899);

    return 0;
}
```

Problem 2

Let *a* be an mxn matrix with 0 and 1 entries. Assume the top left entry `a[0][0] = 1` and the bottom right entry `a[m-1][n-1] = 1`.

A ***path*** between the top left and bottom right entries is a sequence of matrix entries such that adjacent entries in the path are adjacent in the matrix.

The figure below gives two such paths (assume the blank entries are zeros in the matrix).



A **monotone path** is a path that does not travel left or up. In the figure, the red path is monotone. The blue path is not, because it goes left.

Write a function `int monotonePath(int m, int n, int a[m][n])` that returns 1 if the matrix `a` has a monotone path from `a[0][0]` to `a[m-1][n-1]`, otherwise return 0.

Submit `path.c` file containing only your implemented function (that is, you must delete the test cases portion/the main function). However, **you must keep the required included libraries.**

Sample Test program (You need to add more test cases)

```c
int main(void)
{
    int m = 10;
    int n = 12;
    int a[][12] = {{1,1,0,0,0,0,0,0,0,0,0,0},
                   {0,1,1,0,0,0,0,0,0,0,0,0},
                   {0,0,1,1,1,1,0,0,0,0,0,0},
                   {0,0,1,0,0,1,0,0,0,0,0,0},
                   {0,1,1,0,0,1,1,0,0,0,0,0},
                   {0,1,0,0,0,0,1,0,0,0,0,0},
                   {0,1,1,0,0,0,1,1,1,0,0,0},
                   {0,0,1,0,0,0,0,0,1,0,0,0},
                   {0,0,1,1,1,1,1,1,1,1,1,0},
                   {0,0,0,0,0,0,0,0,0,0,1,1}
    };

    // a, but with one entry switched eliminating the monotone path
    int b[10][12] = {{1,1,0,0,0,0,0,0,0,0,0,0},
                     {0,1,1,0,0,0,0,0,0,0,0,0},
                     {0,0,1,1,0,1,0,0,0,0,0,0},
                     {0,0,1,0,0,1,0,0,0,0,0,0},
                     {0,1,1,0,0,1,1,0,0,0,0,0},
                     {0,1,0,0,0,0,1,0,0,0,0,0},
                     {0,1,1,0,0,0,1,1,1,0,0,0},
                     {0,0,1,0,0,0,0,0,1,0,0,0},
                     {0,0,1,1,1,1,1,1,1,1,1,0},
                     {0,0,0,0,0,0,0,0,0,0,1,1}
    };

    assert (monotonePath(m, n, a));
    assert (!monotonePath(m, n, b));
    return 0;
}
```

Problem 3

Prime factorization is the process of breaking down a composite number into a product of prime numbers raised to their respective powers. For example, the prime factorization of 60 is $2^2 \times 3^1 \times 5^1$.

In this problem, your task is to compute the prime factorization of a number n, but with the constraint that all prime factors in the factorization must be less than 1000.

Task:

Write a C program that reads a `long long int` n (assume n>=2), computes its prime factorization using only primes less than 1000, and outputs the result in the following format:

- Each prime factor should be printed in the form p^k, where p is the prime factor, and k is its power (exponent).
- If there are multiple prime factors, they should be printed with a space separating each factor.

If the number cannot be fully factored using primes less than 1000, factor it as much as possible and output the result with the remaining unfactorized part (if any).

Example:

Input:

60

Output:

2^2 3^1 5^1

Input:

997

Output:

997^1

Input:

1001

Output:

7^1 11^1 13^1

Input:

1000000000000000000

Output:

2^18 5^18

Input:

999999999999999999

Output:

3^4 7^1 11^1 13^1 19^1 37^1 17543877193^1

Input:
1022117

Output:
1022117^1

Constraints:

- Only primes less than 1000 (specifically, primes up to 997) should be used in the factorization.

Submit `primefact.c` file containing the whole program including main, your implemented functions, libraries, etc.