CS 137: Assignment #4

Due on Friday, Oct 28, 2022, 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 2022

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 7.
- **You must create more tests. Examples don't cover all the cases.**
- **You may use MATH Library for only problem 3.**

Problem 1

Write the function `bool canComplete(int i, int arr[], int len)` that returns true if it is possible to go from the first index to the last index of the array by jumping over the elements. At each index i, the maximum jump length is array[i].

Assume that arr is of non-negative integers.

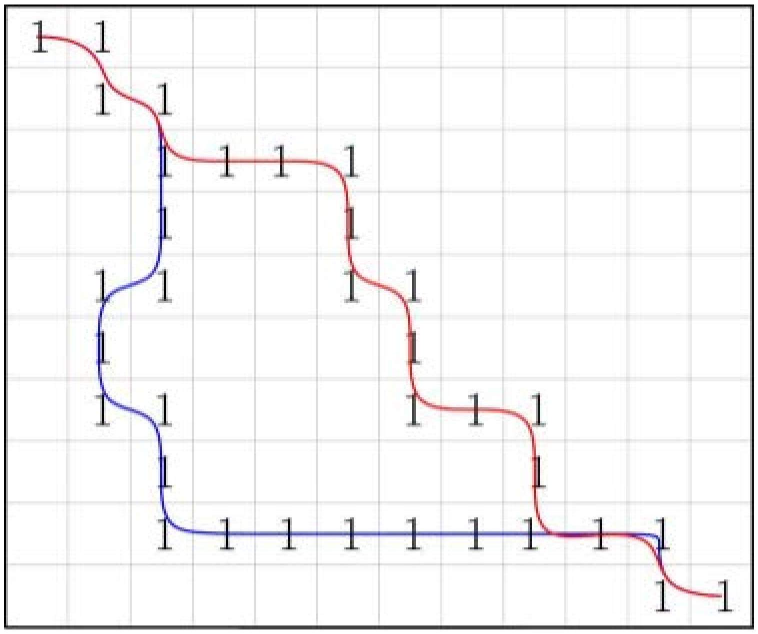
Submit `jumpgame.c` file containing only your implemented function (that is, you must delete the test cases portion/the `main` function). However, **you must keep the required included libraries.**

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

```c
#include <assert.h>
#include <stdbool.h>
int main(void) {
    int arr1[5] = {2, 3, 1, 2, 2};
    int arr2[4] = {2, 1, 0, 2};
    int arr3[10] = {1,1,1,2,0,1,2,3,1,1};

    assert(canComplete(0, arr1, 5) == true);
    assert(canComplete(0, arr2, 4) == false);
    assert(canComplete(0, arr3, 10) == true);
    arr3[8]=0;
    arr3[9]=0;
    assert(canComplete(0, arr3, 10) == true);
    arr3[7]=1;
    assert(canComplete(0, arr3, 10) == false);

    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. Adjacent vertically or horizontally (NOT diagonally).
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)

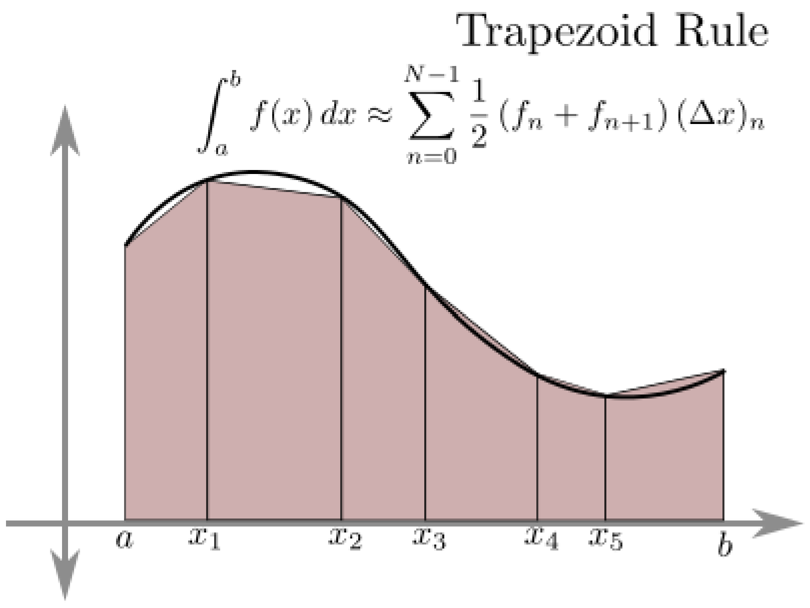
```
int main(void)
{
    int m = 10;
    int n = 12;
    int a[][12] = {{1,1,0,0,0,0,0,0,0,0,0,0},
                   {0,1,1,0,0,0,0,0,0,0,0,0},
                   {0,0,1,1,1,1,0,0,0,0,0,0},
                   {0,0,1,0,0,1,0,0,0,0,0,0},
                   {0,1,1,0,0,1,1,0,0,0,0,0},
                 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

Create a C program named `trapezoidal.c` that consists of a function

```
// b>a, n>0
double trapezoidal(double (*f)(double), double a, double b, double
epsilon, int n);
```

that computes an approximation to find the definite integral of a function.


Trapezoid Rule
$\int_a^b f(x)\,dx \approx \sum_{n=0}^{N-1} \frac{1}{2}(f_n + f_{n+1})(\Delta x)_n$
a
x_1
x_2
x_3
x_4
x_5
b

The function f(x) is divided into many sub-intervals, and each interval is approximated by a Trapezium. Then the area of trapeziums is calculated to find the integral, which is the area under the curve. The more the number of trapeziums is used, the better the approximation.

Trapezodial begins by taking the dividing the area into n sub-intervals, calculate the area of trapeziums, then continue doing the same by doubling the number of sub-intervals until the difference between the last two results (area of n trapeziums vs area of 2n trapeziums) calculated is less or equal epsilon.

Note: You are to submit this file containing only your implemented function (that is, you must delete the test cases portion and the `main` function). However, **you must keep the required included libraries.**

Sample code for testing is below.

```c
#include <stdio.h>
#include <math.h>
#include <assert.h>
#define PI acos(-1)

double f1(double x)
{
        return x * x;
}

double f2(double x)
{
        return cos(x);
}

double f3(double x)
{
        return sqrt(x);
}

double trapezoidal(double (*f) (double), double a, double b, double epsilon, int n)
{

}

int main(void)
{
 assert(fabs(trapezoidal(f1, 5, 10, 0.00001, 2) - 291.667) <= 0.001);
 assert(fabs(trapezoidal(f2, PI/2, 3, 0.00001, 5) - -0.858879) <= 0.000001);
assert(fabs(trapezoidal(f3, 1, 4, 0.00001, 2) - 4.666) <= 0.001);
return 0;
}

```