# Week 4

Tuesday, February 4, 2020 5:11 PM

Week 4 - Spiral 1 - Creativity Challenge 1

Student Name: Nolan Downey
Email: ndowney@nd.edu

Problem 3 Statement - Given an input char* array, write a recursive function that checks whether it is a palindrome.

a) Identify Objectives
   1) Write a program that takes in a character array from the command line
   2) Write a function that passes that character array and checks if it is a palindrome
   3) Use recursion to check the array, not indexing or iterating
   4) If the character array is a palindrome, the function returns the array and tells the user that the array is the same way forwards and backwards (return true)
   5) If it is not a palindrome, then let the user know that it is not a palindrome (return false)

b) Identify Risks and Alternatives
   1) Risk: I need to know if the characters are going to be in Unicode or ASCII
   2) Alternative: I can assume ASCII since we are running on the Notre Dame student machines
   3) Risk: A way to find the length of the array is needed to sufficiently run the function (give the last value of the array)
   4) Alternative: A function was provided in the given code which returns the length of the array from the command line.
   5) Risk: Checking an even number length array and an odd number length array may require different approaches
   6) Alternative: A simple base case statement should take care of this issue with recursive calls
   7) Risk: A simple for loop will check all elements of the array, and an if statement should evaluate if the array is a palindrome
   8) Alternative: The problem specifically states to use recursion, so the same logic should be used by calling the function again
   9) Risk: The function needs to somehow return something to let the user know whether the array is a palindrome
   10) Alternative: Use a boolean function to return true or false

c) Product Development and Testing:
   1) First is determining what needs to be passed to the function. Argv[1], or the array from the command line, needs to be passed to the function. Then, to use the logic of iterating without using a for loop, two values need to be passed to the function, the first and the last value of the array (which will be slowly incremented up or down respectively). So the function should have three values, a char array and two unsigned ints.
   2) The base case needs to handle both even and odd numbered arrays. In an even numbered array, the first and last values will be one away from each other, and then the next iteration of first will be one larger than the last value, so the base case needs to end the recursion when first > last. However, for odd numbered arrays, the first and last values would be equal to one another when checking the middle value of the array. If the function gets to this point, there is no need to check the final, middle value as the function can simply return true if it gets to this point.
   3) I also utilized the given function that returns the length of the array. (Given in the code).
   4) The following test case should depict how I approached this problem.

i. First call
    1) palindrome(argv[1], 0, arrLen -1)
ii. Base Case
    1) if (first >= last)
        a) return true;
iii. else
    1) call palindrome(arr, first + 1, last - 1);
iv. If array is one letter, should immediately return true. If the first and last letters are not equal, the function returns false.
v. The second call checks the second element of the array in comparison to the second to last element of the array, and either continues to call again, returns true, or returns false

5) Test Case:
    i. command line array - madam
    ii. First call
        1) arr[0] == arr[len-1]
            a) m == m
        2) 0 !>= 4
    iii. calls function again
        1) arr[1] == arr[len-1-1]
            a) a == a
        2) 1 !>=1 3
    iv. calls function again
        1) arr[2] == arr[len-1-1-1]
            a) d == d
        2) 2 == 2
        3) returns true because of base case
6) Second Test Case:
    i. command line array - 91239
    ii. First call
        1) arr[0] == arr[len-1]
            a) 9 == 9
        2) 0 !>= 4
    iii. calls function again
        1) arr[1] == arr[len-1-1]
            a) 1 !=1 3
        2) returns false;
    iv. Exits recursion

d) Planning the next phase
    1) After implementing this code, I found no issues after running the provided test case and running some of my own testing
    2) After comparison with Prof. Morrisson's solution, I may know a way to optimize
    3) For now, I believe I have found the best way to solve this issue

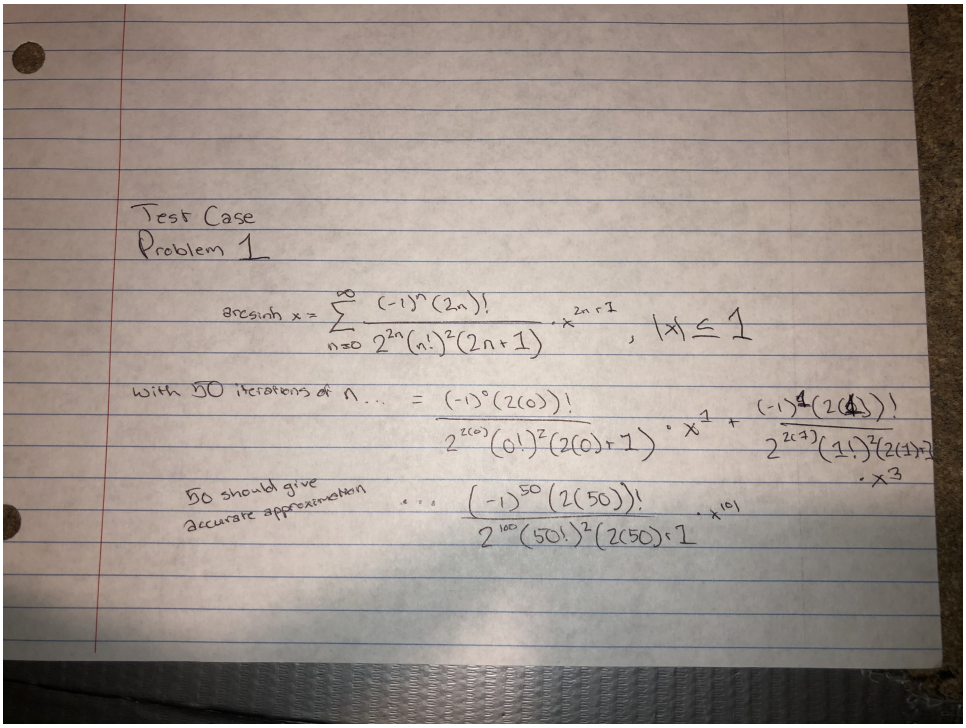
Kelly Buchanan -

Problem 1 - A hyperbolic function is similar to a conventional trigonometric function, except they are used to trace hyperbola instead of a circle. These functions frequently occur in the solutions of many linear differential equations, and have real-world applications in physical and electromagnetic

theory. The *inverse hyperbolic* function is used to determine the original hyperbolic angle. Since the inverse of a trigonometric function includes *arc* and the hyperbolic function includes *h*, we say that the inverse hyperbolic sin function is *arcsineh*.
Write a function that solves the arcsineh function for a value x read in from the command line.

1) Identify Objectives:
    1) Write a program that takes in an inputted x value (between 1 and -1) from the command line and passes it a function that calculates the arcsinh value.
    2) Write a function which approximates the arcsinh value for a given value x
    3) Use recursion to continue increasing the value after every iteration
    4) Use the equation for arcsinh(x) to recursively add for n amount of iterations
    5) Return the result of the equation with all iterations added together

2) Risks and Alternatives:
    1) Risk: Wrong number of arguments in the command line or the number x is out of the range 1 to -1
    2) Alternative: Use getArgv1Num function and return error if these requirements aren't met
    3) Risk: The right number of iterations needs to be evaluated so the recursion formula does not overload or cause an inaccurate approximation
    4) Alternative: set a value to the number of iterations when calling the function in main
    5) Risk: Using the right approximation with the right equation
    6) Alternative: The equation was given to us by the professor
    7) Risk: Along with this risk, a function is needed to calculate the value of a factorial
    8) Alternative: This function is given to us by the professor
    9) Risk: The function needs to utilize the right data type so that it returns an accurate value
    10) Alternative: A double should approximate the value the best
3) Product Testing and Development
    1) The function needs to use recursion to constantly iterate until n = 0, while returning a value through every iteration that then adds together after the recursion ends
    2) The function arcsinh need to pass two values, the double x and an integer n which is used for the number of iterations
        i. utilizes the cmath method pow and the given factorial method to return a value for the equation
        ii. the function needs to call itself to continue to returning double values while decreasing n by 1 to decrease the number of iterations
    3) Call the arcsinh function in main with the value x from the command line and the static value n (50) for the number of iterations
4) Test Case:




5)
6) Planning the Next Steps
    1) Even though I did not implement this code myself, the approach to the code should work. The only main step I have for the next phase would be to change the number of iterations for the function if it was not giving an accurate approximation or was overloading the computing device

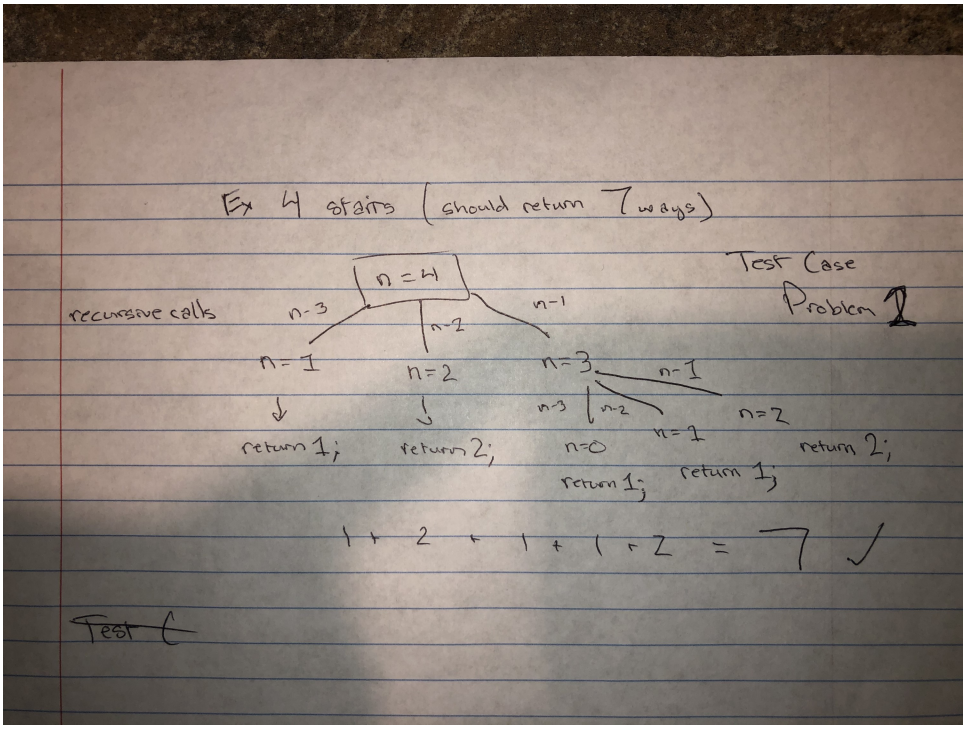
Connor Ruff -

Problem 2 Statement: A child is running up a staircase with n steps and can hop either 1 step, 2 steps, or 3 steps at a time. Implement a function to count how many ways the child can run up the stairs.

1) Identify Objectives:
    1) We need a given value of n for the number of steps to calculate the number of ways.
    2) Write a function for the number of ways that the child can make it up the staircase using recursion
    3) The base case needs to exit the recursion so that the number of ways the child can go up the steps are all added together
    4) Output the number of ways to up the given number of steps the user
2) Risks and Alternatives:
    1) Risk: N, or the number of iterations, needs to be more than the amount subtracted in the equation, otherwise it will go into negative numbers
    2) Alternative: Add a condition that n is a valid number (>= 3).
    3) Risk: Improper values are entered into the command line
    4) Alternative: Assume n will be a positive integer (given in problem) and check that x is within the given range
    5) Risk: Wrong number of command line arguments
    6) Alternative: Check with given function whether correct number of arguments is given
    7) Risk: The function is not considering every single pathway
    8) Alternative: Use the given testing script to ensure results of all kinds
3) Product Development and Testing
    1) Implement a count variable the function which will be incremented by every recursive call

2) Check the initial number of stairs because if they are less than 2, the function can return 1. If the number is 2, the function can return 2.
3) The recursive call will need to be called three times to determine the accurate number of ways
    i. Function(n-3), function(n-2), and function(n-1)
    ii. All of these calls will add together to return an accurate number of ways
    iii. Each call of the function will continue until n <= 2.
4) Use the getArgv1Num(argc, argv) to identify n from the command line and then call the function which determines the possible number of ways
5) The recursive calls should be added together to return a total number of ways to get up the staircase

4) Test Case:
5)




6)
7) Planning the next phase:
    1) Even though I did not implement the code, I believe this approach is an accurate and efficient way to solve this problem. I will check with the given solution to determine if there is a more efficient way to go about this process.