Homework 6

David MHS Webster

**1)**

a) I) the if part sets the parameters by which the recursion should end. Without it, there would be n o condition under which the program would return and a stack overflow would eventually occur because return is never called, infinite recursion

II) the P function raises m to the n power. If m is 2 and n is 2, the result will be 4. if m is 2 and n is 5, the result is 32.

b) The effect would result in infinite recursion and thus an inevitable stack overflow error. Recursive Statements written within an if statement, assuming the return condition is eventually met, would (assuming you have enough memory) not result in a stack overflow. Statements after the if statement, outside of any if statement (or similar tricks- inline if using ?:) may not result in recursion if and only if a return is eventually made before that recursive call can be made.

**2)** strange counts how many times the number 2 “goes into” n. Note that if one were to enter a number 0 or less, the program would cause a stack overflow. This is because, of course, n/2 would not move toward 0. if N is 2, then the result = 1 + strange(n/2); line would be called exactly once, since 2 only goes into 2 once. If N was 3, it would be much the same. 4? It would be called twice and thus the return value would be 2. To fix the error, the function could either be altered to only accept unsigned integers, or n==1 could be changed to n<=1.

3) I have written the function as well as written a program to test it.

Here is the code (it should be attached seperately as well for ease-of-reading), compiled and tested on g++ (Debian 8.3.0-2) 8.3.0

#include<iostream>

#include<cstdlib>

#include<cmath>

#include<cstring>

#include<cstdio>

bool isPalinDrome(char\* inword, int length){ //use strlen to calculate string length for length

char test1, test2;

if(length <= 1) //strings with length 1 are already palindromes

return true;

//The first and last characters

test1 = inword[0]; test2 = inword[length -1];

//printf("\n%c versus %c, hmmm....", test1, test2);

if(test1 == test2)

return true && isPalinDrome(inword + 1, length - 2);

else

return false;

return true;

//WORD: ABRACADABRA

//Iteration 1: A VS A, call against [1] and [length-2]

//Iteration 2: B VS R, return false

//return is false

}

int main(){

char\* strings[] = {"Hello", "ABBA", "GODDOG", "ABRACADABRA", "YAGGAY", "GODADOG"};

int strings\_length = 6;

for(int i = 0; i < strings\_length; i++)

printf("\nString %i is palindrome: %i", i, isPalinDrome(strings[i], strlen(strings[i]) ) ) ;

}

4) Pascal’s Identity is a theory, postulating that for any natural number, the equation given in the instructions is true. Pascal’s Identity is not a value and cannot be calculated. It is, however, possible to calculate (n, k) and I have written a program to do so (Also attached for ease of reading), compiled and tested on g++ (Debian 8.3.0-2) 8.3.0

#include<iostream>

#include<cstdlib>

#include<cmath>

#include<cstring>

#include<cstdio>

typedef unsigned int uint; //makes this so much easier to type

typedef long int lint; //ditto

long int factorial(lint n){ //Converting from uint to int has an issue, this solves it

if(n <= 1)

return 1;

else

return n \* factorial(n-1);

} //this is recursive, so it counts

double Pascal\_Identity(uint n, uint r){

double result;

result = (double)(factorial((lint)n)) / (double)(factorial((lint)n - (lint)r) \* factorial(r));

return result;

}

int main(){

//for(lint i = 0; i < 10; i++)

// printf("\nfactorial(%i) is %i", i, factorial(i));

//printf("\n");

//Proof of the pudding

for(lint n = 0; n < 10; n++)

for(lint r = 0; r < 10; r++)

printf("\nn = %ld and r = %ld, and Pascal\_Identity(n, r) is %f", n, r, Pascal\_Identity(n, r));

return 0;

}