|  |  |  |
| --- | --- | --- |
| American University of Sharjah  College of Engineering  Department of Computer Science & Engineering  P. O. Box 26666, Sharjah, UAE |  | **Instructors:** Dr. Michel Pasquier  **Lab Instructor:** Praveena Kolli  **Office:** EB2-12  **Phone**: 971-6-5152352  **e-mail**: pkolli@aus.edu  **Semester:** Summer 2017 |

**CMP305L - Data Structures and Algorithms Lab**

**Lab. Assignment 7-Recursion**

**Objectives:**

* Understand Recursion
* Develop functions using recursion

**Exercise 1**

1. Develop and test a recursive function that calculates the alternate series:

F(n) =  1 - {1 \over 2} + {1 \over 3} - {1 \over 4} + {1 \over 5} - \cdots =\sum_{n=1}^\infty (-1)^{n+1} {1 \over n}=\ln(2).

#include <iostream>

using namespace std;

double series(int n){

if (n == 1){ return 1; }

else

{

if (n % 2 == 0)

{

//n = n\*(-1);

return((-1.0) / (double)n) + (series(n -1 ));

}

else

{

return((1.0) / (double)n) + (series(n - 1));

}

}

}

void main()

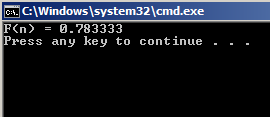
{

double m;

m = series(5.0);

cout << "F(n) = " << m << endl;

}



**Exercise 2**

Develop and test the following *recursive function* that takes an *integer* (0 to 9) and prints the output as shown in sample the input/output.

void IntegerPalindrome(int value);

*Sample Input/Output:*

Enter an integer: 0

0123456789876543210

Enter an integer: 5

567898765

#include <iostream>

using namespace std;

void IntegerPalindrome(int value){

if (value == 9){

cout << 9;

}

else{

cout << value;

IntegerPalindrome(value + 1);

cout << value;

}

}

void main()

{

int m;

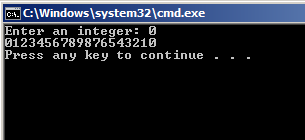
cout << "Enter an integer: ";

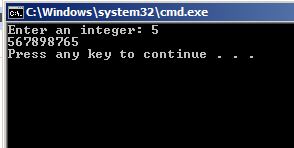
cin >> m;

IntegerPalindrome(m);

cout << "\n";

}





**Exercise 3**

Develop and test a recursive *function* to check if positive integer ***n*** is a prime. An integer ***n*** is a prime if is divisible only by ***1*** and ***itself*** and *not* by any integer in the range from 2 to *sqrt(n)* (both inclusive).

***Note:*** 0 and 1 are not prime numbers. The *sqrt* function provided by <cmath> library returns a double and you must consider only the integral part of it.

#include <iostream>

#include<cmath>

using namespace std;

bool prime(double p, int x){

if (p == 0) return false;

if (p == 1) return false;

int ch1 = 1;

double ch2 = 1;

ch1 = p / x;

ch2 = p / x;

if (p == 3)return true;

if (ch1 == ch2){

if (p != x){

return false;

}

}

else{

prime(p, x + 1);

if (x > sqrt(p))return false;

return true;

}

}

void main()

{

double p;

cout << "Please enter a positive integer: ";

cin >> p;

if (prime(p, 2))

{

cout << "Integer is a prime\n";

}

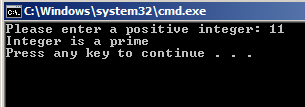
else

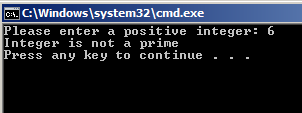
{

cout << "Integer is not a prime\n";

}

}





**Exercise 4**

Develop and test a *recursive* *function* that takes an integer ***n*** as its argument and prints all the *Fibonacci* numbers right from *F(0)* through *F(n ).*  The function must also print the total number of recursive function calls made after printing the last *Fibonacci* number. Recall that the *Fibonacci* sequence is deﬁned by the relation

*F (n) = F (n−1) + F (n−2),* where *F (0) = 0 and F (1) = 1.*

*Sample input/output:*

Enter the range of the Fibonacci series: 15

Fibonacci Series: 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987

The number of recursive calls made =?

Press any key to continue . **. .**

#include <iostream>

#include<cmath>

using namespace std;

int fib(int f){

if (f == 0 || f == 1){

return f;

}

else{

return(fib(f - 1) + fib(f - 2));

}

}

void main()

{

int f;

cout << "Enter the range of the Fibonacci series: ";

cin >> f;

cout << "Fibonacci Series: ";

int x = 0;

while (x <= f+1){

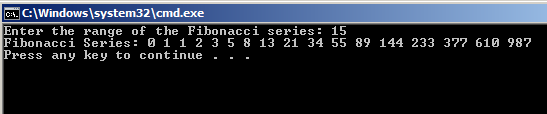
cout << fib(x) << " ";

x++;

}

cout << "\n";

}



**Exercise 5**

1. Develop and test a *recursive* *function* that takes an array of *integers*, its *size* and a target integer *value* as its *arguments* and returns *true* if target *value* is found in the array else returns *false.* The function must use **linear search** algorithm to search for the target *value* in the array.

Note: Print the array inside the function to understand the range for every iteration.

**Exercise 6**

Implement the *destructor*functions of the given *UnsortedType* list (linked list version) using recursion. Test your implementation with a suitable main program.

UnsortedType::~UnsortedType()

// Post: List is empty; all items have been deallocated.

{

NodeType\* tempPtr;

while (listData != NULL)

{

tempPtr = listData;

listData = listData->next;

delete tempPtr;

}

}