|  |  |  |
| --- | --- | --- |
| 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:** Spring 2021 |

**CMP305L - Data Structures and Algorithms Lab**

**Lab. Assignment 8-Recursion**

**Objectives:**

* Understand Recursion
* Develop functions using recursion
* Implement recursive traversal of linked lists

***Note:***

***Lab:*** Exercises 1,2,3,4 and 5 (10 marks)

***Bonus*:** Exercise 6 (1 mark)

**Exercise 1**

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

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

double Harmonic(double n) {

if (n == 1) return 1;

return 1.0 / n + Harmonic(n - 1);

}

int main(){

cout << Harmonic(3);

return 0;

}



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).

double Harmonic(int n) {

if (n == 1) return 1;

if (n % 2 == 0) return ((-1.0 / n) + Harmonic(n - 1));

else return ((1.0 / n) + Harmonic(n - 1));

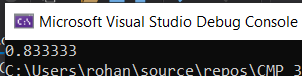
}

int main(){

cout << Harmonic(3);

return 0;

}



**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

**Hint:** Remember lines of code after the recursive call are executed in reverse order of the calls.

void IntegerPalindrome(int value) {

cout << value;

if (value == 9) return;

IntegerPalindrome(value + 1);

cout << value;

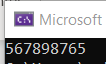
}

int main() {

IntegerPalindrome(5);

return 0;

}



**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 isPrime(int n, int x = 2) {

if (n % x == 0) return false;

if (x > sqrt(n)) return true;

return isPrime(n, x + 1);

}

int main() {

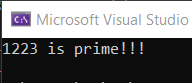
int n = 1223;

if (isPrime(n)) cout << n << " is prime!!!" << endl;

else cout << n << " is NOT prime!" << endl;

return 0;

}



**Exercise 4**

Develop and test a recursive function that converts the given decimal number into equivalent binary number. No data structure should be used.

void toBin(int dec) {

if (dec == 0) return;

toBin(dec / 2);

cout << dec % 2;

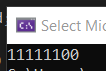
}

int main() {

toBin(252);

return 0;

}



**Exercise 5**

Write the following recursion function on a singly linked list developed in Lab3.

A function compare that takes two lists and returns true if both the lists have same values, if not return false.

#include <iostream>

using namespace std;

template <typename Object>

struct SingleNode {

Object data;

SingleNode\* next;

SingleNode(const Object& d = Object{}, SingleNode\* n = nullptr) //constructor initializing data to empty object and

: data{ d }, next{ n } { } //next pointer to null

};

template <typename Object>

SingleNode<Object>\* createSLL(Object ary[], int size) //arguments are array of Object and size of array

{

SingleNode<Object>\* first = new SingleNode<Object>(ary[0]); //create structure instance and point to first element of array

SingleNode<Object>\* temp = first; //create temp instance and point to first element of array

for (int i = 1; i < size; i++)

{

SingleNode<Object>\* node = new SingleNode<Object>(ary[i]); //create instances called nodes and point to array elements

temp->next = node; //next pointer of first element points to node

temp = node; //with each iteration, the previous node's next pointer will point to current node

}

return first; //return pointer to first element of array

}

template <typename Object>

void printSLL(SingleNode<Object>\* head)

{

while (head != nullptr)

{

cout << head->data << "\t";

head = head->next;

}

cout << endl;

}

template <typename Object>

bool isEqual(SingleNode<Object>\* l1, SingleNode<Object>\* l2) {

if (l1 == nullptr && l2 == nullptr) return true;

if (l1 == nullptr || l2 == nullptr) return false;

isEqual(l1->next, l2->next);

}

int main() {

int arr[5] = { 1,2,3,4,5 };

int arr2[5] = { 1,2,3,4,5 };

int arr3[6] = { 1,2,3,4,5,6 };

SingleNode<int>\* l1 = createSLL(arr,5);

SingleNode<int>\* l2 = createSLL(arr2, 5);

SingleNode<int>\* l3 = createSLL(arr3, 6);

cout << "l1 and l2:" << endl;

if(isEqual(l1, l2)) cout << "THEY ARE EQUAL!" << endl;

else cout << "THEY ARE NOT EQUAL!" << endl;

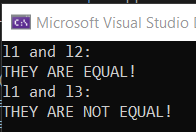
cout << "l1 and l3:" << endl;

if (isEqual(l1, l3)) cout << "THEY ARE EQUAL!" << endl;

else cout << "THEY ARE NOT EQUAL!" << endl;

return 0;

}



**Bonus:**

**Exercise 6:**

Write the following recursion function on a singly linked list developed in Lab3.

Write a single function, findMinMax that traversals the given linked list once to find both the minimum and maximum and returns a pair that contains min and max of the list.

template <typename Object>

pair<Object,Object> findMinMax (SingleNode<Object>\*lst,Object min,Object max)

Reference: <https://www.cplusplus.com/reference/utility/pair/pair/>

#include <iostream>

#include<utility>

using namespace std;

template <typename Object>

struct SingleNode {

Object data;

SingleNode\* next;

SingleNode(const Object& d = Object{}, SingleNode\* n = nullptr) //constructor initializing data to empty object and

: data{ d }, next{ n } { } //next pointer to null

};

template <typename Object>

SingleNode<Object>\* createSLL(Object ary[], int size) //arguments are array of Object and size of array

{

SingleNode<Object>\* first = new SingleNode<Object>(ary[0]); //create structure instance and point to first element of array

SingleNode<Object>\* temp = first; //create temp instance and point to first element of array

for (int i = 1; i < size; i++)

{

SingleNode<Object>\* node = new SingleNode<Object>(ary[i]); //create instances called nodes and point to array elements

temp->next = node; //next pointer of first element points to node

temp = node; //with each iteration, the previous node's next pointer will point to current node

}

return first; //return pointer to first element of array

}

template <typename Object>

pair<Object, Object> findMinMax(SingleNode<Object>\* lst, Object min, Object max) {

if (lst == nullptr) return make\_pair(min, max);

if (lst->data >= max) max = lst->data;

if (lst->data <= min) min = lst->data;

return findMinMax(lst->next, min, max);

}

int main() {

int arr[5] = { 1,2,3,4,5 };

SingleNode<int>\* l1 = createSLL(arr, 5);

pair<int, int> result = findMinMax(l1, l1->data, l1->data);

cout << "The minimum is: " << result.first << endl;

cout << "The maximum is: " << result.second << endl;

return 0;

}

