|  |  |  |
| --- | --- | --- |
| 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 9 – Application of Recursion**

***Objectives:***

To understand

* Program recursive functions with accumulators
* Implement recursive traversal of linked lists
* Develop programs that make use of recursion

***Note:***

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

***Bonus*:** Exercise 4 (1 mark)

**Exercise 1**

Implement a recursive function *PI(n*) that calculates *π*with *n*terms, as per the series given below. Write two versions: (a) *without* accumulator and (b) *with accumulator.* Note that you need first to derive a proper recurrence formula, then implement it.

**π = 3** +

1. Code:

double PI(int n = 3)

{

if(n == 0) return 3.0;

int den = 2 \* n;

double fraction = 4.0/((den) \* (den+1) \* (den+2));

fraction \*= (n % 2 == 0 ? -1 : 1);

return fraction + PI(--n);

}

int main()

{

int n;

cout << "Enter how many terms of PI to calculate: ";

//term with 3 is the 0th term

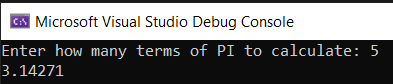
cin >> n;

cout << PI(n);

return 0;

}

Screenshot:



1. Code:

double PI(int n = 3, double acc = 3.0)

{

if(n == 0) return acc;

int den = 2 \* n;

double fraction = 4.0/((den) \* (den+1) \* (den+2));

acc += (n % 2 == 0 ? -1 : 1) \* fraction;

return PIAcc(--n, acc);

}

int main()

{

int n;

cout << "Enter how many terms of PI to calculate: ";

//term with 3 is the 0th term

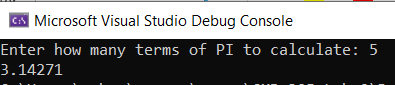
cin >> n;

cout << PI(n);

return 0;

}

Screenshot:



**Exercise 2**

Write a recursive function that mixes up the elements of two given linked lists i.e., taking an element alternately from one then from the other, etc. The lists may be of any length; the extra elements from the longer list should be appended at the end. Use singly linked lists from Lab 3.

*Examples:* head = alternateMix( headList1, headList2 );

If headList1 points to the list [3,4,7] and headList2 points to the list [2,5,6,8,] then head will point to a new list containing [3,2,4,5,7,6,8].

Likewise, mixing the two lists [2,4,6,8] and [3,5,7] will produce the new list [2,3,4,5,6,7,8], while mixing [7,1,4] and [2,8,9,5,3,6] will yield [7,2,1,8,4,9,5,3,6].

**Code:**

#include <iostream>

#include <string>

using namespace std;

//defining single node

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

};

//function to create Singly Linked List with an array of values

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>

SingleNode<Object>\* alternateMix(SingleNode<Object>\* headList1, SingleNode<Object>\* headList2) {

if (headList1 == nullptr && headList2 == nullptr) return nullptr;

if (headList1 != nullptr && headList2 == nullptr) return headList1;

if (headList1 == nullptr && headList2 != nullptr) return headList2;

//None of the above, hence normal case where we add alternate node:

headList1->next= alternateMix(headList2, headList1->next);

return headList1;

}

int main() {

int arr1[] = { 1,2,3,4,5,6 }, size = 6;

int arr2[] = { 7,8,9,10,11,12 };

SingleNode<int>\* head1 = createSLL<int>(arr1, size);

SingleNode<int>\* head2 = createSLL<int>(arr2, size);

cout << "Printing List1: \n";

printSLL(head1);

cout << endl;

cout << "Printing List2: \n";

printSLL(head2);

cout << endl;

SingleNode<int>\* newhead = alternateMix(head1, head2);

cout << "Printing alternate list:\n";

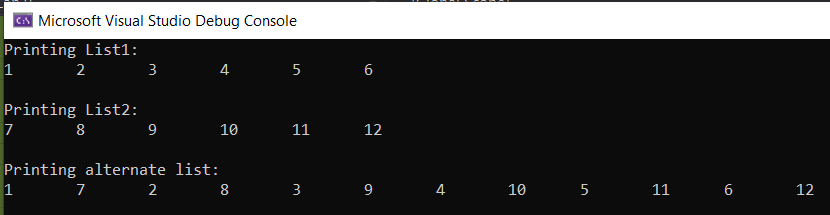
printSLL(newhead);

cout << endl;

return 0;

}

**Screenshot:**



***Exercise 3:***

Using the given program demo/shell, write a recursive function that draws an *H-Tree* of given depth *N*. The five *H-Trees* for *N= 1* to *N= 5* are shown hereafter, for your reference.

Note that the task only requires to devise the recursive logic. All graphic code is included and you only need to use the *drawH()* and *drawline()* functions, as may be necessary.

|  |  |  |
| --- | --- | --- |
| http://www.itu.dk/people/kasper/BITP-2011/00_Revision%20exercises/recursive_procedures/applet/htree1.jpg | https://encrypted-tbn2.gstatic.com/images?q=tbn:ANd9GcSD5ZVuteLxDQ1F8hTNBOiiKH9BVndqTIycjpFBOg9DvRDS2ONm3g | http://www.cs.princeton.edu/courses/archive/fall02/cos126/assignments/htree3.jpg |
| http://www.cs.princeton.edu/courses/archive/spr01/cs126/assignments/htree4.jpg | http://www.cs.princeton.edu/courses/archive/fall03/cs126/assignments/htree5.png |  |

**Note:** This program uses the *simple-SVG* library from <https://github.com/adishavit/simple-svg>. Interestingly, the entire library is contained in a single header file: *simple\_svg\_1.0.0.hpp.*

Windows users: The library is included in the Visual Studio project given to you.

Mac users: You can use Xcode if you are familiar with it or, simply, you can compile the code in the Terminal with *g++ -std=c++11 -o demo demo.cpp* and run the program as *./demo.*

The easiest way to view the created SVG image is to drag and drop it in your web browser.

**CODE:**

// A simple-SVG program example -- Michel Pasquier, 28 March 2021

//

// This demo program uses the Simple-SVG library to draw lines and create

// images for CMP 305 labs. All graphic code is provided so one can focus

// solely on the logic of the application and recursively drawing shapes.

//

// Change the code where indicated below to implement your own function,

// calling drawline(a, b) as necessary. See sample usage, also to create

// points... Change the title as you wish e.g., to display your name/s.

//

// Compile on macOS in the Terminal with: g++ -std=c++11 -o test demo.cpp

// Run the program in the Terminal with: ./demo

// Drag and drop the created SVG image in your web browser to see it.

//

// The library consists of a single header file, included below, and can

// be found at https://github.com/adishavit/simple-svg (was from Google).

// The following code is adapted from the given main program example.

#include "simple\_svg\_1.0.0.hpp"

using namespace svg;

#include <string>

using namespace std;

void drawline( Point a, Point b, Document& doc, Color color ) {

doc << (Polyline{ Stroke{1, color}} << a << b );

}

// draw a single H, centered at x,y of the given side length

void drawH( double x, double y, double len, Document& doc, Color color ) {

// compute the coordinates of the 4 tips of the H

double x0 = x-len/2, x1 = x+len/2, y0 = y-len/2, y1 = y+len/2;

// draw the 3 line segments of the H: left, right, horizontal

drawline( Point{x0,y0}, Point{x0,y1}, doc, color );

drawline( Point{x1,y0}, Point{x1,y1}, doc, color );

drawline( Point{x0,y}, Point{x1,y}, doc, color );

}

// plot a H-tree of order n centered at x,y of the given length

void drawHtree( int n, double x, double y, double len, Document& doc, Color color ) {

if (n == 1) return;

drawH(x - len / 2, y - len / 2, len / 2, doc, color);

drawH(x - len / 2, y + len / 2, len / 2, doc, color);

drawH(x + len / 2, y - len / 2, len / 2, doc, color);

drawH(x + len / 2, y + len / 2, len / 2, doc, color);

drawHtree(n-1, x - len / 2, y - len / 2, len / 2, doc, color);

drawHtree(n-1,x - len / 2, y + len / 2, len / 2, doc, color);

drawHtree(n-1,x + len / 2, y - len / 2, len / 2, doc, color);

drawHtree(n-1,x + len / 2, y + len / 2, len / 2, doc, color);

}

int main() {

const int SIZE {340}; // dimensions of the square image

string FILE {"ztest\_image.svg"}; // name of the saved image file

string TITLE {"H-tree Fractal"}; // title text displayed at top left

Dimensions dim{ SIZE, SIZE };

Document doc{ FILE, Layout{ dim, Layout::TopLeft }};

doc << Rectangle{ Point{0, 0}, dim.width, dim.height, Color::Black };

doc << Text{ Point{10, 16}, TITLE, Color::Green, Font{ 10, "Verdana" }};

drawH(SIZE/2, SIZE/2, SIZE/2, doc, Color::Green);

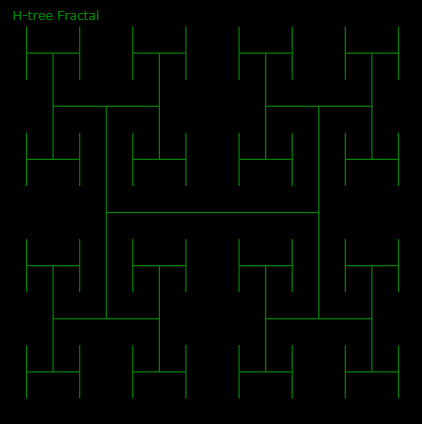
drawHtree( 3, SIZE/2, SIZE/2, SIZE/2, doc, Color::Green );

// end of your code

doc.save();

}

**Screenshot:**



**Bonus:**

***Exercise 4:***

Write the below functions,

1. Write and test an accumulator recursive function that takes two vector iterators, it1 and it2 as parameters and returns the sum of the numbers of the vector between it1 and it2.

double calculate(vector<int>::iterator it1,

vector<int>::iterator it2, double sum=0)

1. Generalize the code in part (a) by making it a function templates, i.e.:

template< typename Iter, typename Value >

Value Sum( Iter it1, Iter it2, Value sum );

1. **Code:**

//Q4)a)

double calculate(vector<double>::iterator it1, vector<double>::iterator it2, double sum = 0) {

if (it1 >= --it2) return (sum);

sum += (\*it1)+(\*it2);

return calculate(++it1, it2, sum);

}

int main() {

vector<double> arr = { 1,2,3,4,5,6,7,8,9,10 };

cout << calculate(arr.begin(), arr.end()) << endl;

return 0;

}

**Screenshot:**



1. **Code:**

//Q4)b)

template< typename Iter, typename Value >

Value Sum(Iter it1, Iter it2, Value sum) {

if (it1 >= --it2) return (sum);

sum += (\*it1) + (\*it2);

return calculate(++it1, it2, sum);

}

int main() {

vector<double> arr = { 1,2,3,4,5,6,7,8,9,10 };

cout << Sum(arr.begin(), arr.end(),0.0) << endl;

return 0;

}

**Screenshot:**

