# Department of Computing

# School of Electrical Engineering and Computer Science

**CS - 250: Data Structure and Algorithms**

**Class: BSCS 10AB**

**Lab 06 : Recursion**

**Date: 02st November, 2021**

**Time: 10:00 am – 12:50 pm   
&  
 02:00 pm – 4:50 pm**

# Mashal Ashfaque

337203

CS 10A

# 

# Lab 06 : Recursion

**Introduction**

This lab will let you practice recursion.

**Objectives**

The Objective of this lab is to train students to be able to trace, design and implement recursive algorithms.

**Tools/Software Requirement**

Visual Studio c++, Eclipse C++ IDE

**Helping Material**

Lecture slides, text book

**Description**

Recursive Design

There are five parts to designing a recursive algorithm.

1. **Identify the problem:** What are the name and arguments of the original problem to be solved?
2. **Identify the smaller problems:** What are the smaller problems that will be used to solve the original problem?
3. **Identify how the answers are composed:** Once the solutions to the smaller problems are in hand, how are they combined to get the answer to the original problem?
4. **Identify the base cases:** What are the smallest problems that must be solved directly? What are their solutions?
5. **Compose the recursive definition:** Combine the parts into a complete definition.

During lectures, the examples of problems like factorial of a number n, Fibonacci number at a position n were discussed in detail in this context. The purpose of this lab is to trace, design and implement recursive algorithms.

**Note:** As explained in the class, do include print statements in the first line of the function along with the parameter value(s). For every base case and recursive case, add a print statements as well before the base or recursive case terminates.

**Tasks:**

**Task 1 (Factorial of a number n):**

Implement a recursive function to compute the factorial of a non-negative integer n. *Trace all the function calls, convergence to the base case and how answers for various sub-calls get returned prior to getting the final answer.*

**CODE**

#include <iostream>

#include <algorithm>

using namespace std;

int factorial(int n)

{

static int i = 0; //static int to keep count of function calls

i++;

cout << "\nFunction call # " << i << endl; //tracing the function call

if (n < 0) //n has to be non negative

{

cout << "Not possible.\n";

}

else if (n == 0)

{

return 1;

}

else

{

cout << "Function call #" << i << " returned " << n << "\* factorial(" << n - 1 << ")";

return n \* factorial(n - 1);

}

}

int main()

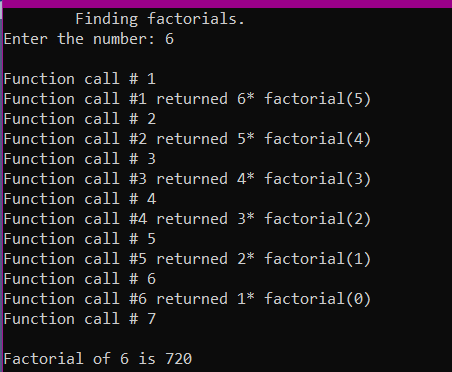
{

cout << "\tFinding factorials.\nEnter the number: ";

int n; cin >> n;

cout << "\nFactorial of " << n << " is " << factorial(n) << endl;

}



**Task 2 (Fibonacci Sequence):**

Implement a recursive function to compute the Fibonacci of a non-negative integer position n. Print the entire Fibonacci series from position **0** till position n. *Trace all the function calls, convergence to the base case and how answers for various sub-calls get returned prior to getting the final answer.*

Code:

#include<iostream>

using namespace std;

int fib(int n)

{

cout << "Call to fib(" << n << ")" << endl; //tracing the function calls

if ((n == 0) || (n == 1)) //base case

{

cout << "Base case. Answer is 1" << endl;

return 1;

}

else

{

int answer = fib(n - 1) + fib(n - 2);

cout << "End of call fib(" << n << "). Ans: " << answer << endl;

return answer;

}

}

int main()

{

int num;

cout << "Enter number ";

cin >> num;

int\* array1 = new int[num];

//array to store sequence

for (int i = 0; i < num; i++)

{

array1[i] = fib(i);

}

cout << "\n\nFabonacci series : \n\n" << endl;

//printing the sequence

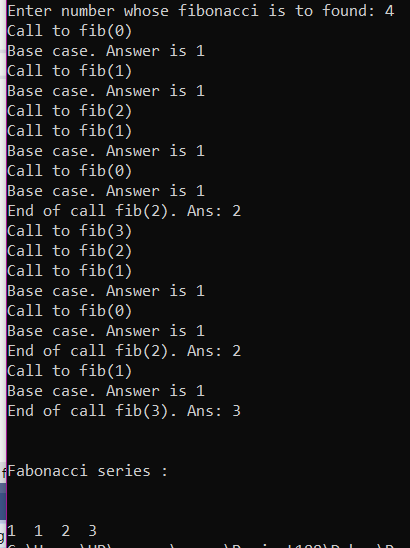
for (int i = 0; i < num; i++)

{

cout << array1[i] << " ";

}

}



**Task 3 (Recursive search):**

Implement a function that recursively searches a value in an array of size n. If the value is found, the function should return the index number in which it is stored. Otherwise, it should return -1 to show that it does not exist.

**Code:**

#include<iostream>

using namespace std;

int search(int arr[], int n,int x) //n is size of arr , x is searched element

{

static int i = 0;

if (arr[i] == x) //value found

{

return i;

}

else if(i==n) //whole array has been searched

{

cout << "Value not found" << endl;

return -1;

}

else

{

i++;

search(arr, n, x);

return i;

}

}

int main()

{

int x, n;

cout << "Enter the size of array:" << "\n";

cin >> n;

int\* arr = new int(n);

//Entering data in array

for (int i = 0; i < n; i++)

{

cout << "Enter element # " << i + 1 << endl;

cin >> arr[i];

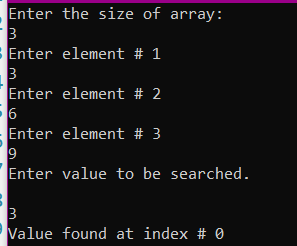
}

cout << "Enter value to be searched." << endl;

cin >> x;

cout<<"Value found at index # "<<search(arr, n, x);

}

****

**Task 4 (Palindrome):**

**A palindrome is a string** of characters (a word, phrase, or sentence) that is the **same** regardless of whether you **read** it forward or backward such as civic, kayak, 1001 etc. Implement a recursive function that checks whether the given sequence of characters is a palindrome or not. Return true if it is, false otherwise.

#include<iostream>

using namespace std;

void print(char[], int);

bool palindrome(char arr[], int l,int r) //n is size of array

{

static bool pdrm=true;

if (r == l)//only one char

{

return pdrm;

}

if (r < l) //base case, whole array has been checked

{

return pdrm;

}

else if (arr[l] == arr[r]) //check characters

{

return palindrome(arr,++ l, --r ); //proceed to next characters

}

else if (arr[l] != arr[r])

{

pdrm = false; return pdrm; //if mismatched, then pdrm is false, not a palindrome

}

}

int main()

{

int n;

while (1)

{

cout << "\nEnter the size of string:" << "\n";

cin >> n;

char\* arr = new char(n);

cout << "Enter string\n";

//storing string in char array

for (int i = 0; i < n; i++)

{

cin >> arr[i];

}

if (palindrome(arr, 0, n - 1))

{

cout << endl ;

print(arr, n);

cout << " is a palindrome";

}

else

{

cout << endl << endl;

print(arr, n);

cout << " is not a palindrome" << endl;

}

}

}

//To print string

void print(char arr[], int n)

{

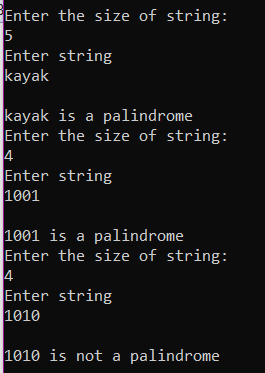
for (int i = 0; i < n; i++)

{

cout << arr[i];

}

}



**Task 5 (Print a Singly Linked List in the reverse order):**

Implement a function that prints all elements of a singly linked list in the reverse order.

**CODE:**

#include<iostream>

using namespace std;

class ListNode {

public:

int data;

ListNode\* next;

};

class LinkedList {

public:

ListNode\* start; // special variable which stores address of the head node.

ListNode\* last; // special variable which stores address of the last node.

ListNode\* preloc; //to be used by Search(value) method to store address of logical predecessor of value in a list.

ListNode\* loc; //to be used by Search(value) method to store address of the node containing the searched value in a list. If it is not found it contains NULL.

int length = 0;

LinkedList() //constructor to create empty linked list

{

start = NULL;

preloc = NULL;

loc = NULL;

}

//other functions

bool isEmpty(); // checks whether the list is empty or not. Returns true if empty and false otherwise.

void InsertAtFront(int value); // takes input from a user and inserts it at the front of a list

void InsertAtEnd(int value); // takes input from a user and inserts it at the tail end of a list

void search(int value); //searches value entered by user, it will also be used in other functions like insertion etc

void InsertSorted(int value);//To maintain a sorted list, you shall implement this function.

//Note that if you are maintaining a sorted list then do not call InsertAtFront(value) and InsertAt(Front)

//functions in the main function

void PrintList();

};

bool LinkedList::isEmpty()

{

return start == NULL;

}

void LinkedList::InsertAtFront(int value)

{

ListNode\* newnode = new ListNode();

newnode->data = value;

if (isEmpty())

{

start = newnode;

last = newnode;

}

else

{

newnode->next = start;

start = newnode;

}

length++;

}

void LinkedList::InsertAtEnd(int value)

{

ListNode\* newnode = new ListNode();

newnode->data = value;

if (isEmpty())

{

start = newnode;

last = newnode;

}

else

{

last->next = newnode;

last = newnode;

}

}

void LinkedList::search(int value)

{

loc = start;

preloc = NULL;

if (isEmpty())

{

return;

}

while (loc != NULL && loc->data < value)

{

preloc = loc;

loc = loc->next;

}

if (loc != NULL && loc->data < value)

{

loc = NULL; //as value is not found so set loc equal to null.

}

}

void LinkedList::PrintList()

{

if (!isEmpty())

{

ListNode\* temp = start;

while (temp != NULL)

{

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

temp = temp->next;

}

}

else

{

cout << "List is Empty.\n";

}

}

void LinkedList::InsertSorted(int value)

{

search(value);

if (loc == NULL)

{

if (preloc == NULL)

InsertAtFront(value);

else

{

ListNode\* newnode = new ListNode();

newnode->data = value;

newnode->next = preloc->next;

preloc->next = newnode;

if (preloc == last) //insertion at tail end

last = newnode;

}

length++;

}

else

cout << value << " already exists in list. Duplication not allowed." << endl;

}

void print(ListNode\* start)

{

if(start!=NULL)

{

print(start->next);

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

}

}

int main()

{

LinkedList mylist;

//Inserting some data in the list

for (int i = 0; i < 5; i++)

{

mylist.InsertSorted(i);

}

cout << "Original List:\n";

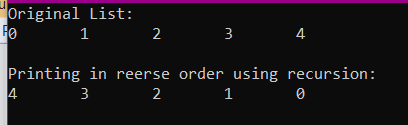
mylist.PrintList();

cout << "\n\nPrinting in reverse order using recursion:\n";

print(mylist.start);

cout << endl << endl;

}



**Task 6 (Reverse a Singly Linked List):**

Implement a function that rearranges a singly linked list by reversing its order. For instance, if the original list is 1→2→3→4→5, the updated list should be 5→4→3→2→1.

**Code:**

#include<iostream>

using namespace std;

class ListNode {

public:

int data;

ListNode\* next;

};

class LinkedList {

public:

ListNode\* start; // special variable which stores address of the head node.

ListNode\* last; // special variable which stores address of the last node.

ListNode\* preloc; //to be used by Search(value) method to store address of logical predecessor of value in a list.

ListNode\* loc; //to be used by Search(value) method to store address of the node containing the searched value in a list. If it is not found it contains NULL.

int length = 0;

LinkedList() //constructor to create empty linked list

{

start = NULL;

preloc = NULL;

loc = NULL;

}

//other functions

bool isEmpty(); // checks whether the list is empty or not. Returns true if empty and false otherwise.

void InsertAtFront(int value); // takes input from a user and inserts it at the front of a list

void InsertAtEnd(int value); // takes input from a user and inserts it at the tail end of a list

void search(int value); //searches value entered by user, it will also be used in other functions like insertion etc

void InsertSorted(int value);//To maintain a sorted list, you shall implement this function.

//Note that if you are maintaining a sorted list then do not call InsertAtFront(value) and InsertAt(Front)

//functions in the main function

ListNode\* reverselist(ListNode\* node);

void PrintList();

};

bool LinkedList::isEmpty()

{

return start == NULL;

}

void LinkedList::InsertAtFront(int value)

{

ListNode\* newnode = new ListNode();

newnode->data = value;

if (isEmpty())

{

start = newnode;

last = newnode;

}

else

{

newnode->next = start;

start = newnode;

}

length++;

}

void LinkedList::InsertAtEnd(int value)

{

ListNode\* newnode = new ListNode();

newnode->data = value;

if (isEmpty())

{

start = newnode;

last = newnode;

}

else

{

last->next = newnode;

last = newnode;

}

}

void LinkedList::search(int value)

{

loc = start;

preloc = NULL;

if (isEmpty())

{

return;

}

while (loc != NULL && loc->data < value)

{

preloc = loc;

loc = loc->next;

}

if (loc != NULL && loc->data < value)

{

loc = NULL; //as value is not found so set loc equal to null.

}

}

void LinkedList::PrintList()

{

if (!isEmpty())

{

ListNode\* temp = start;

while (temp != NULL)

{

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

temp = temp->next;

}

}

else

{

cout << "List is Empty.\n";

}

}

void LinkedList::InsertSorted(int value)

{

search(value);

if (loc == NULL)

{

if (preloc == NULL)

InsertAtFront(value);

else

{

ListNode\* newnode = new ListNode();

newnode->data = value;

newnode->next = preloc->next;

preloc->next = newnode;

if (preloc == last) //insertion at tail end

last = newnode;

}

length++;

}

else

cout << value << " already exists in list. Duplication not allowed." << endl;

}

/\* Function to reverse the linked list \*/

ListNode\* LinkedList::reverselist(ListNode\* node)

{

if (node == NULL)

return NULL;

if (node->next == NULL)

{

start = node;

return node;

}

ListNode\* node1 = reverselist(node->next);

node1->next = node;

node->next = NULL;

return node;

}

int main()

{

LinkedList mylist;

//Inserting some data in the list

for (int i = 0; i < 5; i++)

{

mylist.InsertSorted(i);

}

cout << "Original List:\n";

mylist.PrintList();

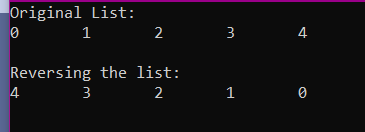
cout << "\n\nReversing the list:\n";

mylist.reverselist(mylist.start);

mylist.PrintList();

cout << endl << endl;

}



**Task 7 (Rearrange a Singly Linked List based on even odd positioned nodes):**

Implement a function that rearranges a singly linked list by separately connecting the **odd positioned** nodes in the same order, and the even positioned nodes in the reverse order. Finally, connect the reversed sub-list of the even positioned nodes before the odd positioned nodes. Finally, update the **start** and **last** pointer variables.

* Examples 1→2→3→4→5→6, the updated list should be 1→3→5→6→4→2.
* Examples 2→4→5→6→8→9→11, the updated list should be 9→6→4→2→5→8→11.

#include<iostream>

using namespace std;

class ListNode

{

public:

int data;

ListNode\* next;

};

class LinkedList

{

public:

ListNode\* start; // special variable which stores address of the head node.

ListNode\* last; // special variable which stores address of the last node.

int length = 0;

ListNode\* pLoc\_; //to be used by Search(value) method to store address of logical predecessor of value in a list.

ListNode\* Loc\_; //to be used by Search(value) method to store address of the node containing the searched value in a list. If it is not found it contains NULL.

LinkedList()

{

start = NULL;

last = NULL;

length = 0;

pLoc\_ = NULL;

Loc\_ = NULL;

}

bool isEmpty()

{

return start == NULL;

}

void PrintList(ListNode\* node)

{

if (node == NULL)

return;

cout << " " << node->data;

PrintList(node->next);

}

void InsertAtEnd(int value)

{

ListNode\* newnode = new ListNode();

newnode->data = value;

if (!isEmpty())

{

last->next = newnode;

last = newnode;

}

else

{

start = newnode;

last = newnode;

}

}

void Search(int value)

{

Loc\_ = start;

pLoc\_ = NULL;

if (isEmpty())

return;

if (Loc\_ != NULL && Loc\_->data < value)

return;

else

{

pLoc\_ = Loc\_;

Loc\_ = Loc\_->next;

Search(Loc\_->data);

}

if (Loc\_ != NULL && Loc\_->data != value)

Loc\_ = NULL;

}

ListNode\* oddtemp;

ListNode\* eventemp;

ListNode\* ES;

int c = 1, d = 0; // as there is no loop here so these variables will help in recursive function calls

void SeparateOddEvenNodes(ListNode\* Ofirst, ListNode\* Efirst)

{ // c variable helps to initaialize only oddtemp and even temp with Ostart and Estart

//only 1 time and not again and agin during function calls

if (c == 1)

{

oddtemp = Ofirst;

eventemp = Efirst;

c++;

}

if (eventemp->next == NULL || oddtemp->next == NULL)

{

return;

}

else

{

oddtemp->next = eventemp->next;

oddtemp = oddtemp->next;

if (eventemp->next->next != NULL)

{

eventemp->next = oddtemp->next;

eventemp = eventemp->next;

}

d++;

SeparateOddEvenNodes(oddtemp->next, eventemp->next);

}

// d variable helps to to link evenlast with oddfirst only 1 time and not again and again

//as there are recursive calls that calls itself again

d--;

if (d == 0)

{

ES = Efirst;

reverse(ES);

ListNode\* temp = ES;

while (temp->next != NULL)

{

temp = temp->next;

}

temp->next = Ofirst; //attaching even nodes with odd nodes at last

oddtemp->next = NULL;

start = ES;

}

}

ListNode\* reverse(ListNode\* head) //reversing the evennodes

{

if (head->next == NULL)

{

return ES = head;

}

reverse(head->next); // recursive function calls

ListNode\* newnode = head->next;

newnode->next = head; // reversing

head->next = NULL;

}

};

int main()

{

LinkedList\* ll = new LinkedList(); // ll is object of class LinkedList

LinkedList\* ll2 = new LinkedList();

int value;

//inserting data into list

for (int i=1;i<7;i++)

{

ll->InsertAtEnd(i);

}

cout << endl << "Original list" << endl;

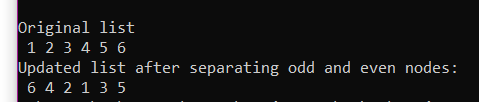
ll->PrintList(ll->start);

ll->SeparateOddEvenNodes(ll->start, ll->start->next);

cout << "\nUpdated list after separating odd and even nodes: \n";

ll->PrintList(ll->start);

}



**Task 8: Fair division**



**Alice** and **Bob** inherited a collection of paintings. However, they will receive it only if the collection can be **divided into two parts of exactly equal price**. (Otherwise, it will be donated to a local art museum.)

**Is the collection divisible into two exactly equal halves? We have to find the answer.**

The prices of the paintings are provided as an array of integers. For example:

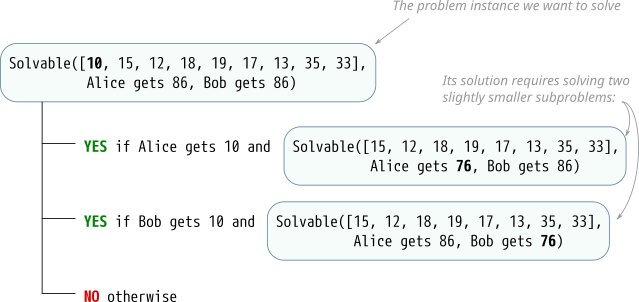
int prices [] = {10, 15, 12, 18, 19, 17, 13, 35, 33};

Here, the total sum is 172, so each person should receive the sum of 86. In this specific example, a solution exists, it is the following partition: (10 + 15 + 12 + 19 + 17 + 13) = (18 + 35 + 33) = 86.

### How to solve the problem recursively?

Consider the example above. Is it possible to divide [10, 15, 12, 18, 19, 17, 13, 35, 33] into sums of 86 and 86?

Each item should go either to Alice or to Bob. Let’s take the first item, **10**. Should we give it to Bob or Alice? In either case, there can be a solution. So, let’s try both options:



If we can give **10** to Alice, and the rest can be divided so that she gets 76 and Bob gets 86, then a solution exists (and Alice gets 10).

Also, if we can give **10** to Bob and the rest can be divided so that he gets 76 and Alice gets 86, then the solution also exists (and Bob gets 10).

Otherwise, there is no solution.

### Programming task

In the same program, write a function:

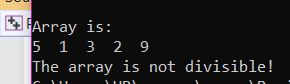
bool divisible(int \*prices, int size);

which returns true if the collection is divisible, and false otherwise. The prices are provided in the array prices, and size is the number of items in the array.

Your function should not allocate new memory dynamically. Pass the same array data into recursive function calls. If you need more variables such as left and right boundary variables, or the amounts that should be given to Alice and Bob, make a helper function with any necessary extra variables.

(It is possible to make the program to actually print out the solution, who gets which item. For that, in each of the YES branches, once you know that a solution to the subproblem exists, print the current item and the name of the person who gets it.)

***This is not a simple task, but if you can do it, this is great!***



#include <iostream>

using namespace std;

void print(int arr[], int n);

//finding the sum of elements of array through recursion

int RangeofSum(int\* arr, int left, int right)

{

if (right - left == 0) return arr[right]; //until left index has reached last element

return arr[left] + RangeofSum(arr, ++left, right);

}

bool divisible(int a, int b, int\* array, int size)

{

if (size != 0 && a - array[size - 1] >= 0)

{

a -= array[size - 1];

if (divisible(a, b, array, --size))

{

if (a == 0 && b == 0)

return true;

else if (size <= 0)

return false;

}

}

else if (size != 0 && b - array[size - 1] >= 0)

{

b -= array[size - 1];

if (divisible(a, b, array, --size))

{

if (a == 0 && b == 0)

return true;

else if (size == 0)

return false;

}

}

else

{

if (a == 0 && b == 0)

return true;

return false;

}

}

int main()

{

int size = 5;

int\* array1 = new int[size];

array1[0] = 5;

array1[1] = 1;

array1[2] = 3;

array1[3] = 2;

array1[4] = 9;

cout << "\nArray is: \n";

print(array1, 5);

int sum = RangeofSum(array1, 0, size - 1);

//cout << "sum = " << sumRange(array1, 0, size-1);

int a = sum / 2;

int b = sum / 2;

if (sum % 2 == 0)

{

if (divisible(a, b, array1, 5))

cout << "\nThe array is divisble!";

else

cout << "\nThe array is not divisible!";

}

else

cout << "\nThe array is not divisible!";

}

void print(int arr[], int n)

{

for (int i = 0; i < n; i++)

{

cout << arr[i]<<" ";

}

}

**Deliverables:**

Compile a single word document by filling in the solution part and submit this Word file on LMS. The name of word document should follow this format. i.e. **YourFullName(reg)\_Lab#.** This lab grading policy is as follows: The lab is graded between 0 to 10 marks. The submitted solution can get a maximum of 5 marks. At the end of each lab or in the next lab, there will be a viva related to the tasks. The viva has a weightage of 5 marks. Insert the solution/answer in this document. You must show the implementation of the tasks in the designing tool, along with your complete Word document to get your work graded. You must also submit this Word document on the LMS. In case of any problems discuss it by emailing it to [aftab.farooq@seecs.edu.pk](mailto:aftab.farooq@seecs.edu.pk).

**Note:** Students are required to upload the lab on LMS before deadline.

Use proper indentation and comments. Lack of comments and indentation will result in deduction of marks.