# 

# Department of Computing

**CS 250: Data Structures and Algorithms**

**Class: BSCS-9AB**

**Lab 08: Recursion**

**Date: December 01, 2020**

**Time: 10:00 am -1:00pm, 2:00pm – 5:00pm**

# Instructor: Dr. Yasir Faheem

# Lab 8: 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 2012 or gcc or g++

**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.

**Lab Tasks**

You are required to upload the lab tasks on LMS and the name of that tasks must be in this format YourFullName\_reg#\_task#.cpp

Remember to comment your code properly. Inappropriate or no comment will result in deduction of marks.

**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.*

#include <iostream>

using namespace std;

class CalculateFactorial {

public:

int factorial;

int fact(int n)

{

cout << " call to fact value " << n << endl;

if (n == 1)

{

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

return 1;

}

else

{

factorial = n \* fact(n - 1);

cout << "End of Fact" <<" "<<n;

cout <<"Answer :"<< factorial << endl;

return factorial;

}

}

};

int main()

{

CalculateFactorial\* f=new CalculateFactorial();

int result;

int value;

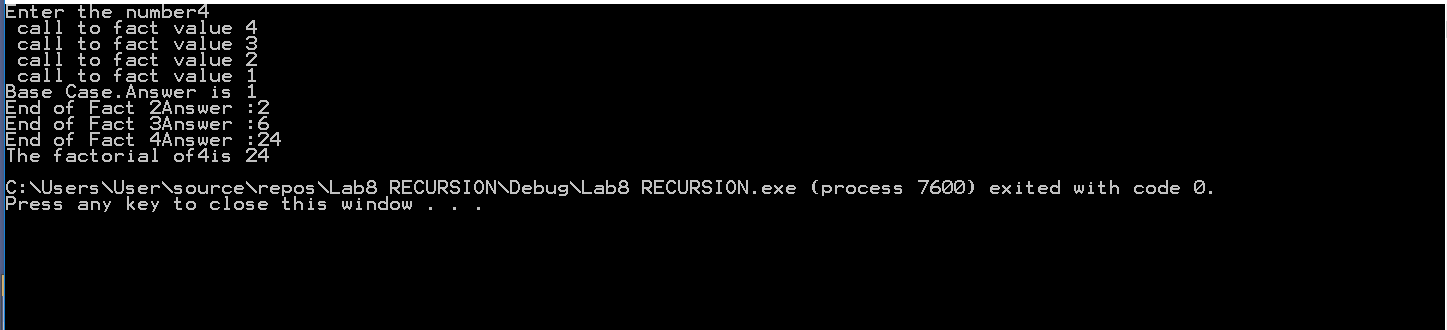
cout << "Enter the number";

cin >> value;

result = f->fact(value);

cout << "The factorial of" << value << "is " << result << 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.*

class FabonacciSeries

{public:

int sum;

int calculate\_fabonacci(int n)

{

cout << "Call To fabonacci"<<endl;

if (n == 1 || n == 0)

{

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

return 1;

}

else

{

sum = calculate\_fabonacci(n - 1) + calculate\_fabonacci(n - 2);

cout << "End of call fib" << sum<<endl;

return sum;

}

}

};

int main()

{

FabonacciSeries\* f = new FabonacciSeries();

int result;

int value;

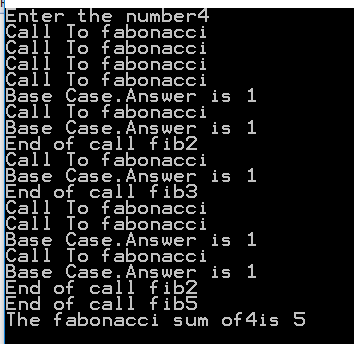
cout << "Enter the number";

cin >> value;

result = f->calculate\_fabonacci(value);

cout << "The fabonacci sum of" << value << "is " << result << endl;

}



**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.

class ARRAY

{public:

int i = 0;

int array[5] = {2,4,8,10,6};

int Search(int value)

{

cout << "call to search for" << "(" <<value << ")"<<endl;

int n =sizeof(array)/sizeof(int);

if (i == n)

{

cout << "value not found in array" << endl;

return 0;

}

if (array[i] == value)

{

cout << "Base Case"<<endl<<"Value is found at pos"<<i <<endl ;

cout << i;

return i;

}

else

{

i++;

Search(value);

}

}

};

int main()

{

ARRAY\* a = new ARRAY();

int value;

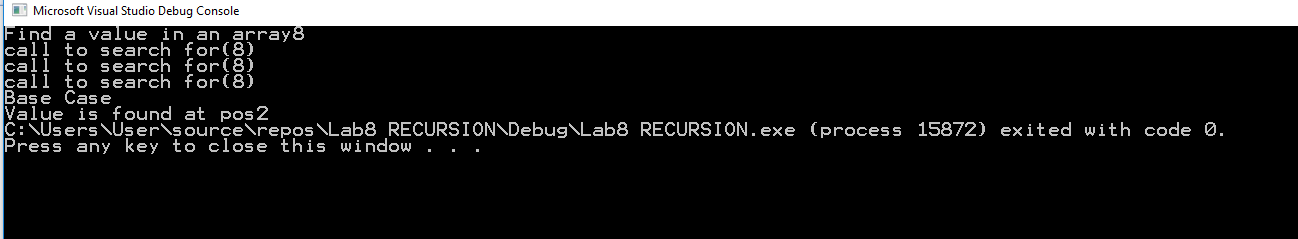
int result;

cout << "Find a value in an array";

cin >> value;

a->Search(value);

}



**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.

class Palindrome

{public:

bool isPalindromeRec(char str[], int s, int e)

{

//Single character, return true

if (s == e)

{

return true;

}

//If first and last characters do not match

//Return false

if (str[s] != str[e])

{

return false;

}

//If they are same, recursion for remaining

if (s < e + 1)

{

return isPalindromeRec(str, s + 1, e - 1);

}

//If it reaches end, return true

return true;

}

bool isPalindrome(char str[])

{

int n = strlen(str);

//empty string is palindrome

if (n == 0)

{

return true;

}

//Else, call recursive function

return isPalindromeRec(str, 0, n - 1);

}

};

int main()

{

Palindrome\* p = new Palindrome();

char str[] = "racecar";

if (p->isPalindrome(str))

{

cout << "Given input string is palindrome" << endl;;

}

else

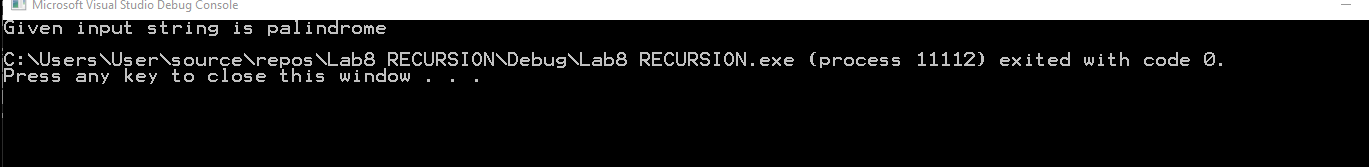
{

cout << "Input string is not palindrome" << endl;

}

return 0;

}



**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.

void printReverse(ListNode\* head)

{

cout << "call to function"<< endl;

// Base case

if (head == NULL)

{

cout << "The List Ends here"<<endl;

return;

}

else {

// print the list after head node

printReverse(head->next);

cout << "function returns for " << head->data<<endl;

}

// After everything else is printed, print head

//cout << head->data << " ";

}

int main()

{

LinkList\* a = new LinkList();

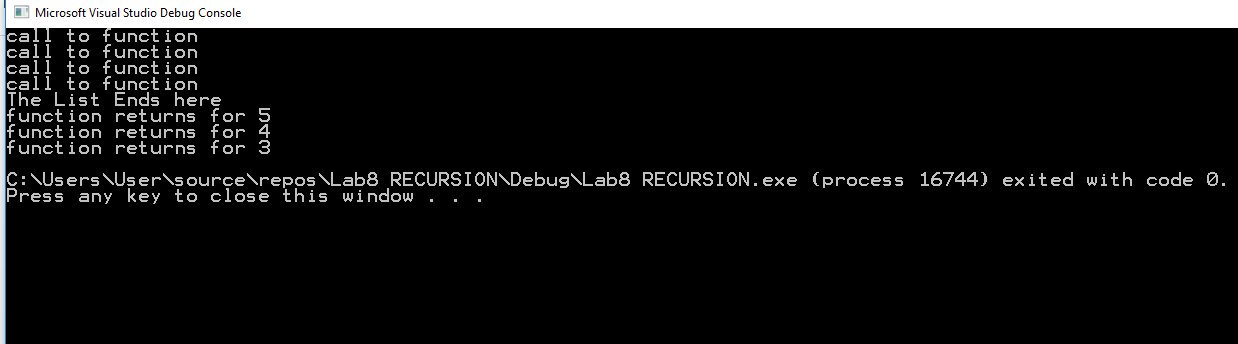
a->insertatFront(5);

a->insertatFront(4);

a->insertatFront(3);

a->printReverse(a->start);

}



**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.

ListNode\* Reverse(ListNode\* head)

{

cout << "call to function for " <<head->data<< endl;

// Base case

if (head==last)

{

start = head;

cout << "The List Ends here" <<head->data<< endl;

cout << endl;

return head;

}

// print the list after head node

Reverse(head->next);

cout << "function returns for"<<head->data<<endl;

head->next->next = head;

head->next = NULL;

last = head;

return head;

//cout << "function returns for " << head->data << endl;

}

void print()

{

temp = start;

while (temp != NULL)

{

cout << temp->data;

temp = temp->next;

}

}

};

int main()

{

LinkList\* a = new LinkList();

a->insertatFront(5);

a->insertatFront(4);

a->insertatFront(3);

cout <<"Print before reversing"<<endl;

a->print();

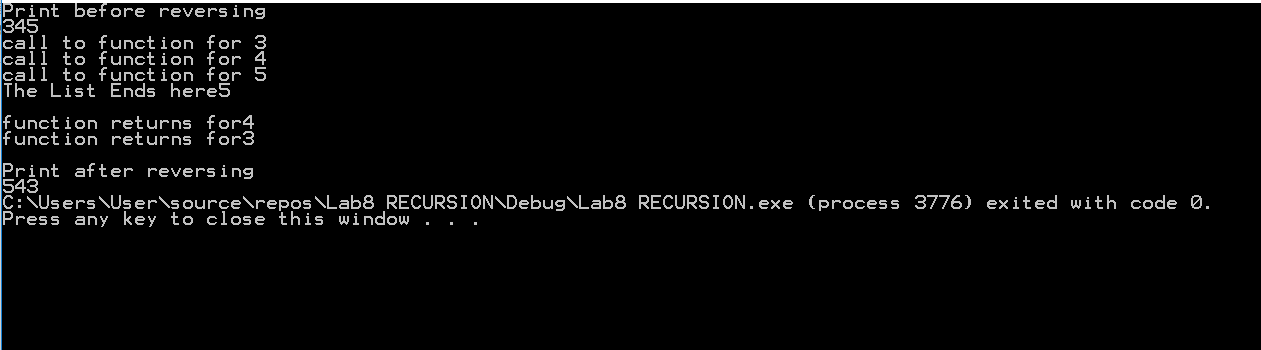
cout << endl;

a->Reverse(a->start);

cout <<endl<< "Print after reversing" << endl;

a->print();

}



**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.

**Tasks for IMDB Project (not part of this lab)**

As already communicated, the deadline to submit the project has been extended till December 06, 2020. Your task is to implement at least one recursive function in your project be it search, print or any other operation.

**Deliverable**

You are required to upload the lab tasks on LMS before the deadline.