# 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**

# Instructor: Dr. Yasir Faheem

# Lab Engineer: Aftab Farooq

# 

# 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>

using namespace std;

int factorial(int num);

int main()

{

int num = 0;

cout << " Enter a non-negative number: "; cin >> num;// input from user

cout << " Factorial is: " << factorial(num) << endl;// output value

return 0;

}

int factorial(int num)

{

if (num == 0)

{

cout<<num<<endl;

return 1;// base case factorial of 0 is 1

}

else

{

cout<<num<<endl;

return num = num \* factorial(num - 1);// sending decrementing number as arguments

}

}

*ScreenShot:*Text

Description automatically generated

**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 s1 = 0, s2 = 1;

int \_count = 2;

int next\_ = 0;

void Fibonacci(int val,int p1,int p2)

{

next\_ = p1 + p2; //add preceding two to get next number

cout << next\_ << " "; //display the new number

p1 = p2; // make the next two and current value as preceding numbers

p2 = next\_;

\_count++;

if (\_count != val) //recurrsion should occur until times n

Fibonacci(val, p1, p2);

}

int main()

{

int n = 0;

cout << " Enter a non-negative number: "; cin >> n;// input from user

cout << "The Series is: ";

cout << '0' << " 1 "; //printing the first two numbers first

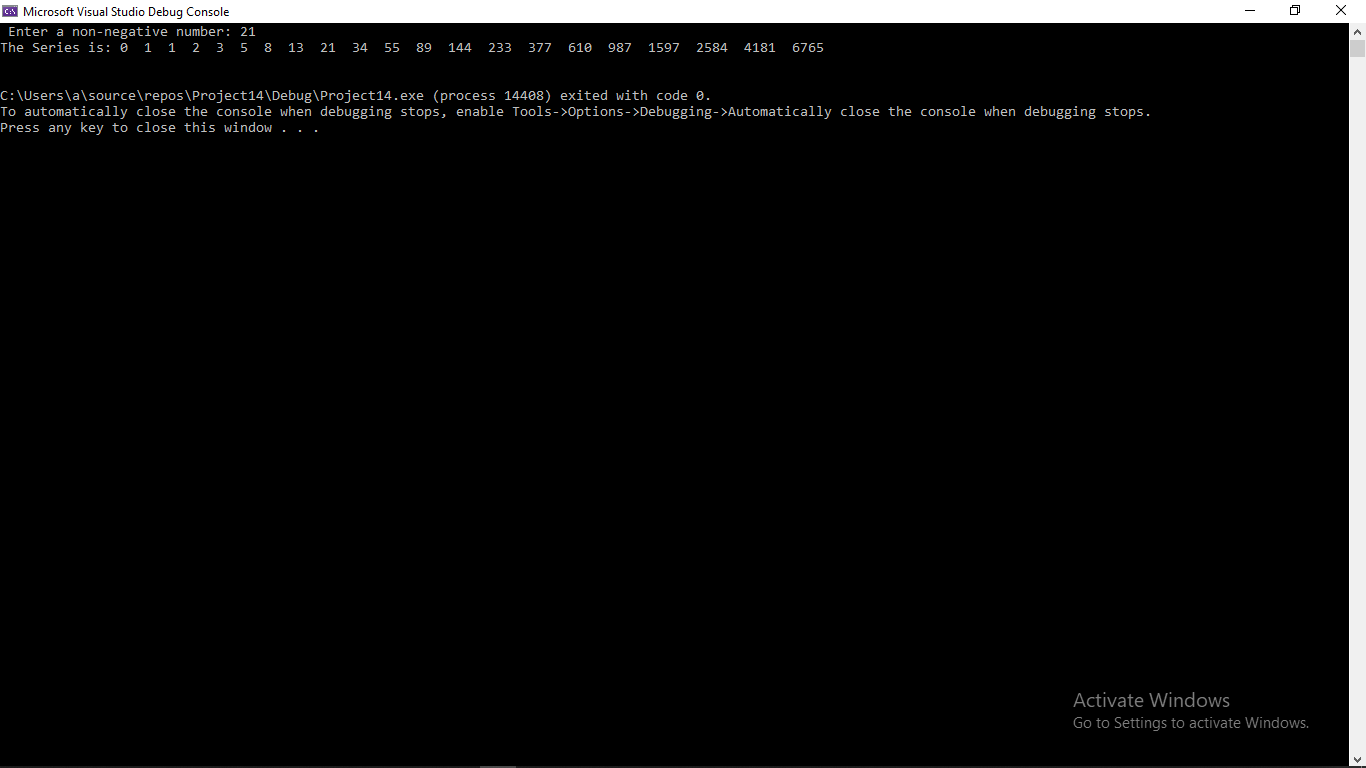
Fibonacci(n, s1, s2);

cout << endl << endl;

return 0;

}

*ScreenShot:*



**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;

#define size 8

int Search(int num, int sval)

{

int list[size] = { 3,5,6,7,8,2,1,4 };

int flag = 0;

if (list[num] == sval)

{

int index = num;// return found value index

return index+1;

}

else

{

if (num == size)// if list size is reached ,then return -1 (not found)

{

return -1;

}

int index;

index = Search(num + 1, sval);// using as loop sends incremented value as argument

}

}

void main()

{

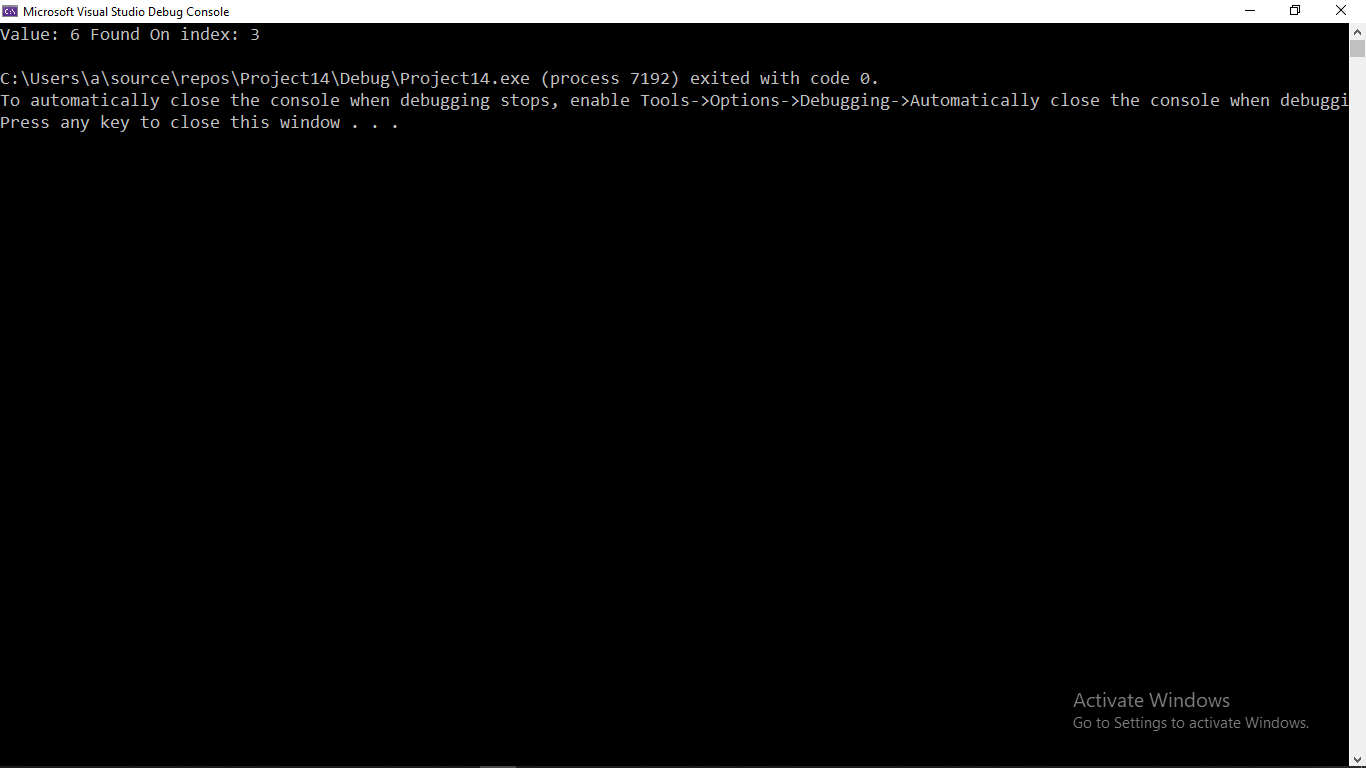
int num = 0;

int svalue=1;

cout <<"Value: "<< svalue << " Found On index: " << Search(num, svalue) << endl;// output statement

}

*ScreenShot:*



**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.

*Code:*

#include<iostream>

#include<string>

using namespace std;

char str[10] = { 'c','i','v','i','c' };// input string

int x = strlen(str); // assigning length of string

int\* flag = new int[x]; //flag to check if all corresponding values are true

int c = 0;

bool palindrome(char str[], int start, int last)

{

if (c < x) // comparing the number of loop count to length

{

if (start != last)

{

if (str[start] == str[last]) // comparing the corresponding elements

{

flag[c++] = 1;

palindrome(str, start + 1, last - 1);// call function

}

else

{

c++;

palindrome(str, start + 1, last - 1);// call function

}

}

int flag1 = 1;

for (int i = 0; i < c; i++) // checking all corressponding element are same

{

flag1 = flag1 \* flag[i];

}

if (flag1 == 1)

{

return 1;

}

else return 0;

}

else

{

int flag1 = 1;

for (int i = 0; i < c; i++) // checking all corressponding element are same

{

flag1 = flag1 \* flag[i];

cout << "flag1: "<<flag1;

}

if (flag1 == 1)

{

return 1;

}

else return 0;

}

}

int main()

{

int len = strlen(str);

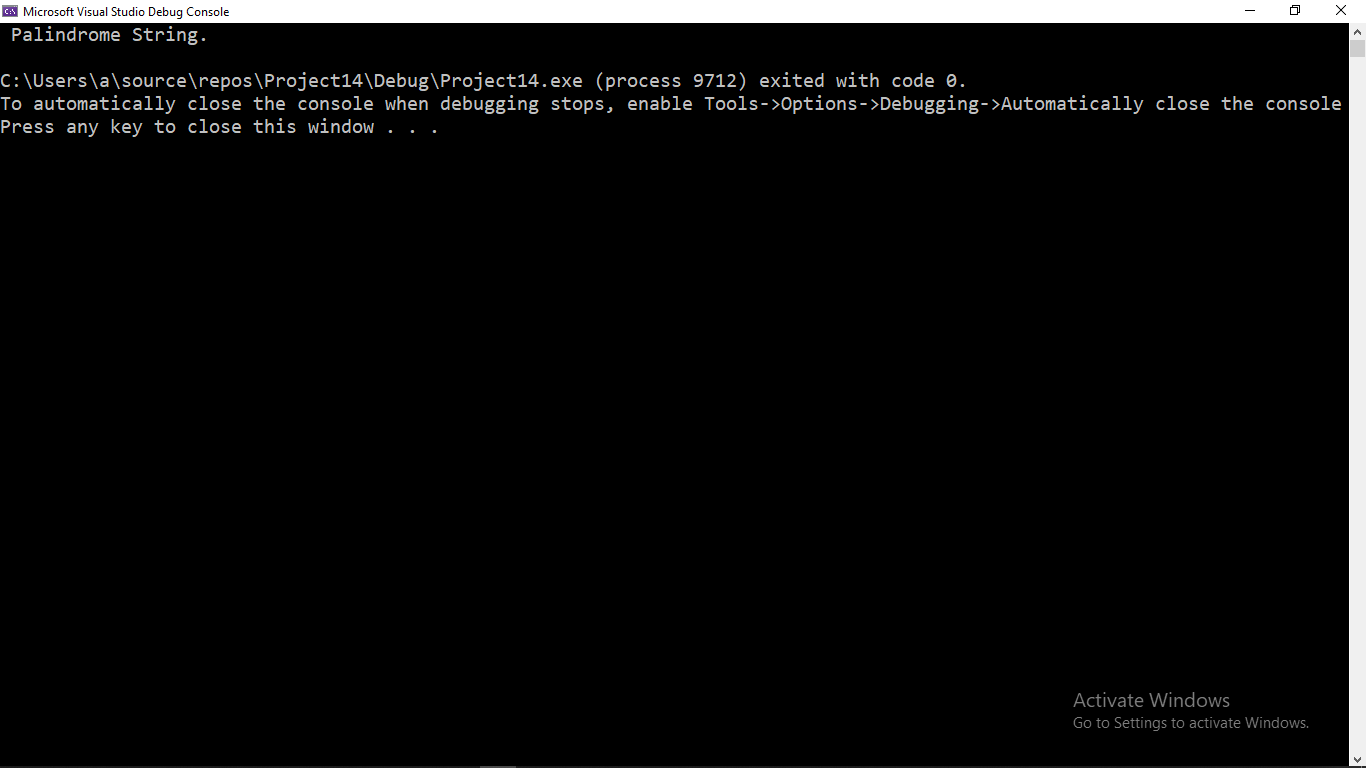
if (palindrome(str, 0, len - 1) == 1)

cout << " Palindrome String." << endl;

else cout << " Not Palindrome String." << endl;

}

*ScreenShot:*



**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 node

{

public:

int data;

node\* next;

};

class linkedlist

{

public:

node\* start;

node\* last;

node\* ctr;

node\* ptr;

int length = 4;

node\* newnode1 = new node;

node\* newnode2 = new node;

node\* newnode3 = new node;

node\* newnode4 = new node;

node\* newnode5 = new node;

linkedlist() //making a temperary list

{

start = newnode1;

last = newnode5;

newnode1->data = 1;

newnode2->data = 4;

newnode3->data = 7;

newnode4->data = 18;

newnode1->next = newnode2;

newnode2->next = newnode3;

newnode3->next = newnode4;

newnode4->next = newnode5;

}

int x = 0; node\* temp = new node;

void reverse\_print(node\* temp)

{

x++;

if (x != length) //check the list limit is not reached

reverse\_print(temp->next); //calling the function without outputing

cout <<" " << temp->data << endl;// outputing the values on addresses stored in stack

}

};

int main()

{

linkedlist x;

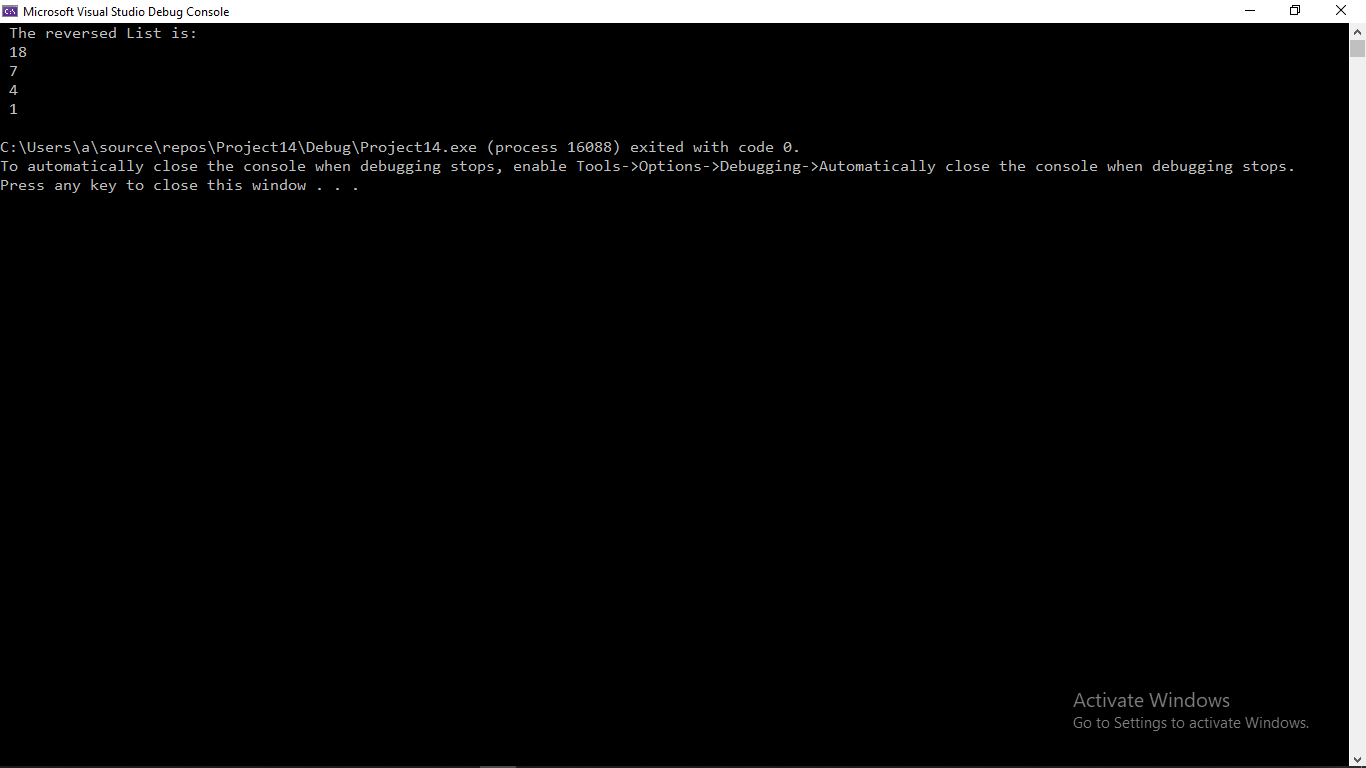
cout << " The reversed List is: " << endl;

x.reverse\_print(x.start);

return 0;

}

*ScreenShot:*



**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 node

{

public:

int data;

node\* next;

};

class linkedlist

{

public:

node\* start;

node\* last;

node\* ctr;

node\* ptr;

node\* sctr;

int length = 6;

node\* newnode1 = new node;

node\* newnode2 = new node;

node\* newnode3 = new node;

node\* newnode4 = new node;

node\* newnode5 = new node;

node\* newnode6 = new node;

node\* newnode7 = new node;

node\* newnode = new node;

linkedlist() //making a temperary list

{

start = newnode1;

last = newnode5;

newnode1->data = 1;

newnode2->data = 4;

newnode3->data = 7;

newnode4->data = 18;

newnode5->data = 20;

newnode6->data = 40;

newnode1->next = newnode2;

newnode2->next = newnode3;

newnode3->next = newnode4;

newnode4->next = newnode5;

newnode5->next = newnode6;

newnode6->next = newnode7;

start = newnode1;

last = newnode6;

newnode->data = 0;

}

int x = 0; node\* temp = new node;

void reverse\_pointer(node\* pptr, node\* cctr, node\* sstr)

{

ptr = pptr; //stores the previous node

ctr = cctr; //stores the current node

sctr = sstr;//stores the successor node

if (x == 0) //when first time

ptr->next = newnode;

x++;

ctr->next = ptr;

if (x != length) //until size of list

reverse\_pointer(ctr, sctr, sctr->next); //sends the next node pointers as arguments

}

void print() //prints the list through addresses

{

start = last;

temp = start;

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

{

cout << " Value " << i + 1 << ": " << temp->data << endl;

temp = temp->next;

}

}

};

int main()

{

linkedlist x;

cout << " The reversed List is: " << endl;

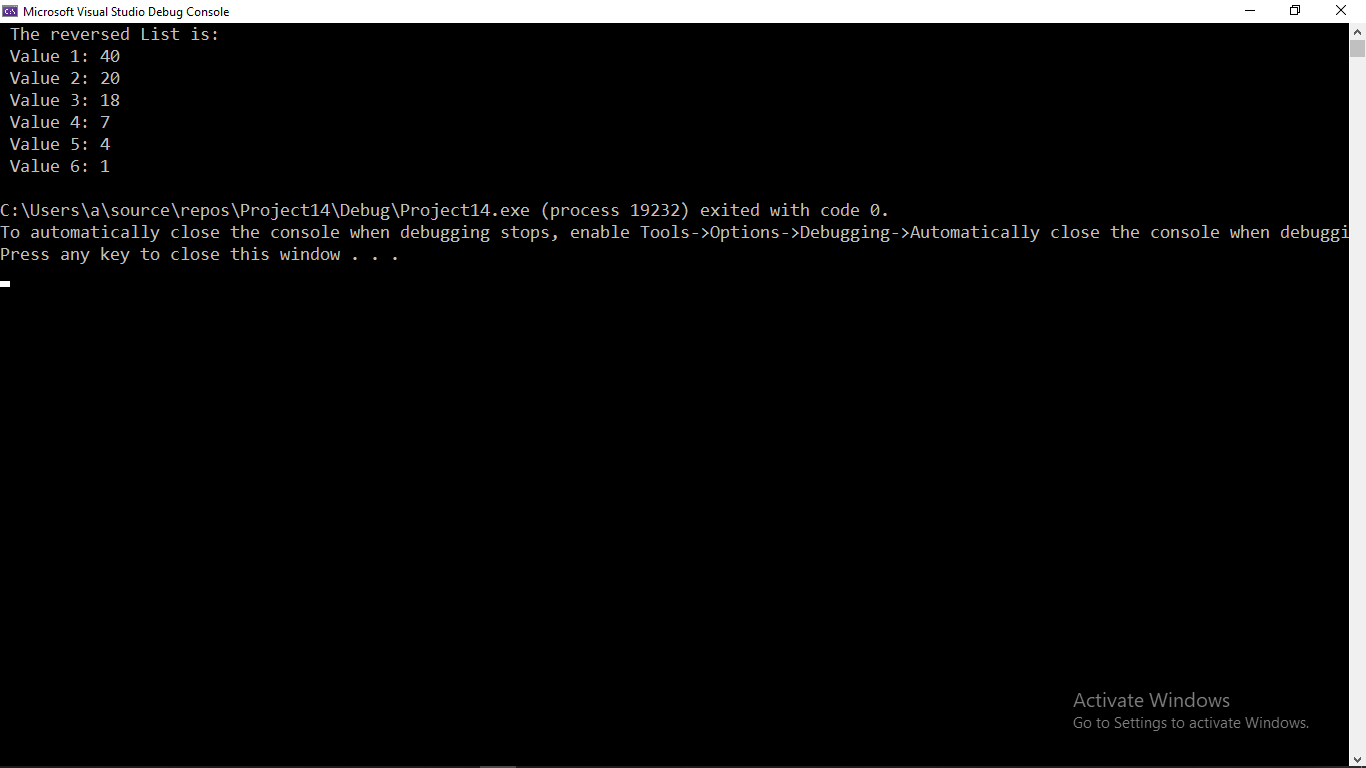
x.reverse\_pointer(x.start,x.start->next,x.start->next->next);

x.print();

return 0;

}

*ScreenShot:*



**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.

*Code:*

#include<iostream>

using namespace std;

class node

{

public:

int data;

node\* next;

};

class linkedlist

{

public:

node\* start;

node\* last;

node\* ctr;

node\* ptr;

node\* sctr;

int length = 6;

node\* newnode1 = new node;

node\* newnode2 = new node;

node\* newnode3 = new node;

node\* newnode4 = new node;

node\* newnode5 = new node;

node\* newnode6 = new node;

node\* newnode7 = new node;

node\* newnode = new node;

linkedlist() //making a temperary list

{

start = newnode1;

last = newnode5;

newnode1->data = 1;

newnode2->data = 2;

newnode3->data = 3;

newnode4->data = 4;

newnode5->data = 5;

newnode6->data = 6;

newnode1->next = newnode2;

newnode2->next = newnode3;

newnode3->next = newnode4;

newnode4->next = newnode5;

newnode5->next = newnode6;

newnode6->next = newnode7;

start = newnode1;

last = newnode6;

newnode->data = 10;

}

int x = 0; node\* temp = new node;

void reverse\_pointer(node\* pptr, node\* cctr, node\* sstr,int len)

{

ptr = pptr; //stores the previous node

ctr = cctr; //stores the current node

sctr = sstr;//stores the successor node

if (x == 0) //when first time

ptr->next = newnode;

x++;

ctr->next = ptr;

if (x != len) //until size of list

reverse\_pointer(ctr, sctr, sctr->next,len); //sends the next node pointers as arguments

}

void print() //prints the list through addresses

{

temp = s1\_;

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

{

cout << " Value " << i + 1 << ": " << temp->data << endl;

temp = temp->next;

}

}

node\* s1\_ = new node; //nodes to store temperary start node

node\* s2\_ = new node; //nodes to store temperary start.next node

int c = 0;

void OddEvenArrange(node\* s1, node\* s2)

{

if (c == 0) //first time run stores initial addresses

{

s1\_ = start;

s2\_ = start->next;

}

c++;

s1->next = s2->next; //pointing odd nodes to the next odd node

s2->next = s1->next->next; //pointing even node to next even nodes

if (c == (length / 2)-1)

{

reverse\_pointer(s2\_, s2\_->next, s2\_->next->next, length / 2); //calling to reverse the even list

s2\_ = s2->next; //replacing start with last node / s2\_ now contains the last node of even list

}

if (c != (length / 2)-1)

OddEvenArrange(s1->next, s2->next);

}

};

int main()

{

linkedlist x;

cout << " The reversed List is: " << endl;

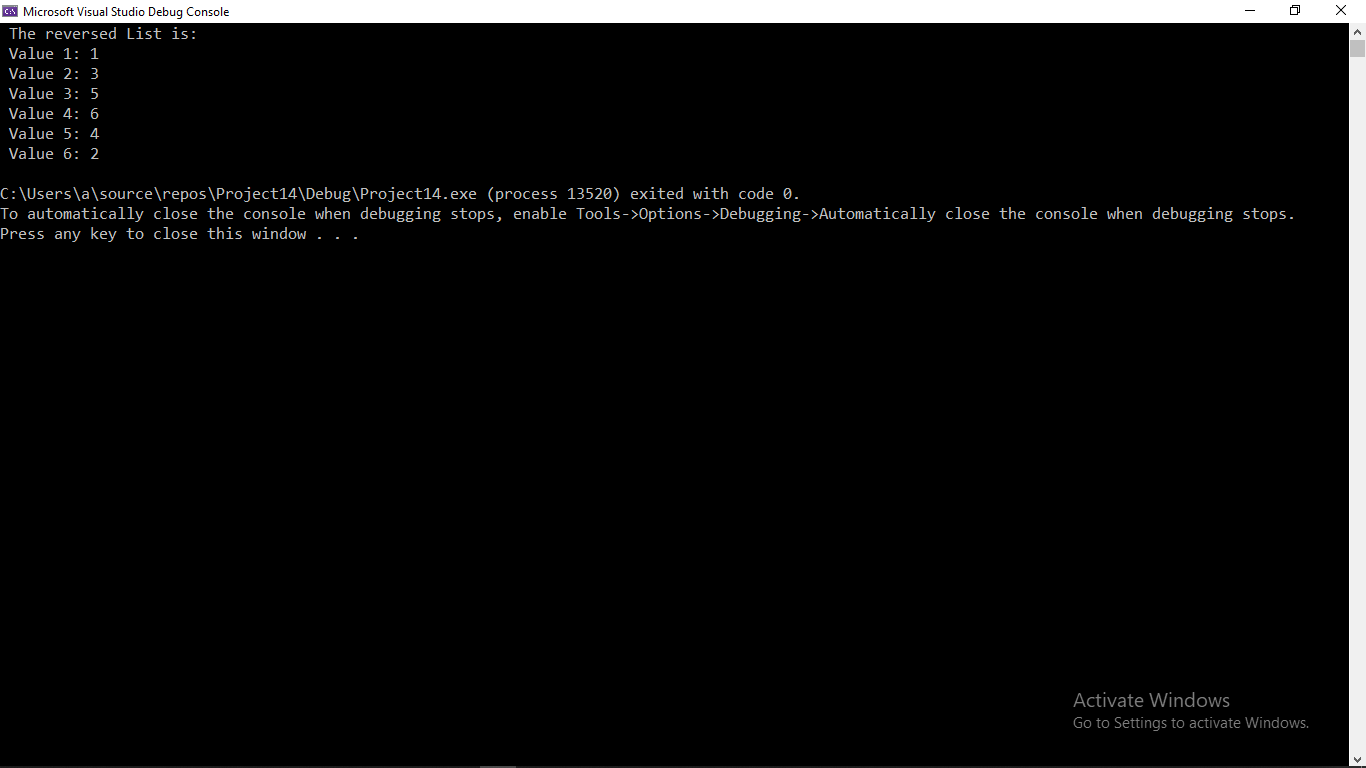
x.OddEvenArrange(x.start, x.start->next);

x.print();

return 0;

}

*ScreenShot:*



**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:

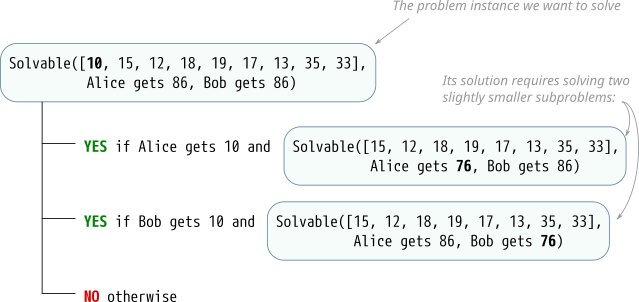
int prices [] = {10, 15, 12, 18, 19, 17, 13, 35, 33};

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!***

*Code:*

#include<iostream>

using namespace std;

#define siz 10 //fixed size of paintings

int list[siz] = { 10, 15, 12, 18, 19, 17, 13, 35, 33 }; //the set of values

int temp[siz] = { 0 }; //list will get new values for new group

int sum = 0;

int sum\_ = 0;

int c = 0;

void sort\_dcc(int arr[], int size)// sorting array in decending order

{

int temp = 0;

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

{

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

{

if (arr[i] < arr[j])

{

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

}

void sort\_acc(int arr[], int size)// sorting array in Accending order

{

int temp = 0;

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

{

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

{

if (arr[i] > arr[j])

{

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

}

void divisible(int price[], int size,int sum)

{

if (price[c] <= sum) //compares the maximum value in the list

{ //if fit in sum then deducts from sum

sum = sum - price[c];

temp[c] = price[c]; //stores deducted value in temperary array

price[c] = 0; //index position in arry price overwritten to 0

}

c++; //increment count

if (sum != 0)

divisible(price, size, sum);

}

int main()

{

for (int i = 0; i < siz; i++) //totalling of list array

sum += list[i];

cout << " The Sum is the List is: " << sum << endl;

if (sum % 2 == 0) //if even sum then call function

{

sort\_dcc(list, siz);

divisible(list, siz, sum/2);

}

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

sum\_ += list[i];

cout << " The Fair division Would be Group of " << sum\_ << endl;

if (sum\_ == sum / 2)

{

cout << " The List Has Fair Division." << endl;

// displaying of both groups in accending order respectively

sort\_acc(list, siz);

cout << " The First Group is" << endl;

cout << " { ";

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

{

if (list[i] != 0)

cout << list[i] << ' ';

}

cout << " } " << endl;

sort\_acc(temp, siz);

cout << " The Second Group is" << endl;

cout << " { ";

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

{

if (temp[i] != 0)

cout << temp[i] << ' ';

}

cout << " } " << endl;

}

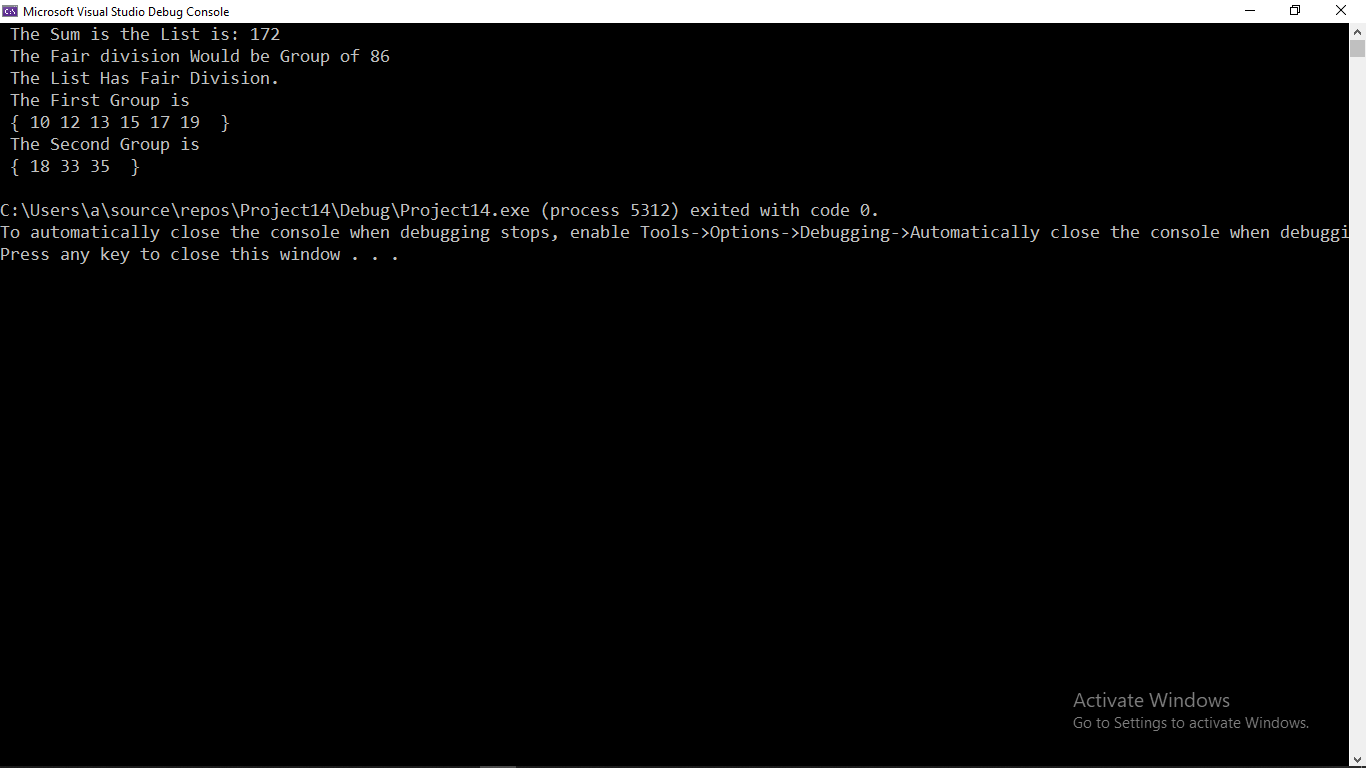
else

cout << " The List Has No Fair Division." << endl << endl;

return 0;

}

*ScreenShot:*



**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.