LAB. MANUAL

Data Structures Lab.

(CS-2091) Through C

FOR B.TECH/DUAL-M.TECH/DUAL-MBA THIRD SEMESTER STUDENTS

Version - 1.0/17.07.2017

Prepared By

Mr. Anil Kumar Swain

Asst. Professor, School of Computer Engineering

School of Computer Engineering

KIIT UNIVERSITY, BHUBANESWAR

INTRODUCTION

- Programming cannot be learned by watching others do it. Students must spend numerous hours working on programs themselves.
- This laboratory manual is a tool that will allow students to experiment with computer science. As students progress through each laboratory, they may wonder how or why something works. The best way to discover the answer is to try things out.
- The **purpose of this lab. manual** is to acquaint the students to understand about writing algorithms and step by step approach in **solving problems with the help of fundamental data structures through C language.**
- This Strengthen the ability of students to identify and apply the suitable data structure for the given real world problem.

STRUCTURE OF THIS LAB. MANUAL

- This lab. manual provides study aids from programming assignments to scheduled exercises using prepared materials.
- This lab manual is divided into **10 laboratory classes**. Each laboratory class consists of the following:
  a) **Lab. Excerise (LE):** These are the **assignments** that ask each student to independently create small programs **during the lab time**.
  b) **Home Exercise (HE):** These are the assignments to be done during lab time by each student if lab. assignments are completed before lab. time or may be assigned as **post-lab homework** and submitted in the next lab class.
  c) **Round Exercise (RE):** These are the **group assignments** to be done by each group round the time/week and submitted one copy per group at any time if asked to submit after one week of RE given.

The approach of each Lab: **LE-HE-RE**

Program Outcomes

PO1	**Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2	**Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	**Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	**Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5	**Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO6	**The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	**Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8	**Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	**Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	**Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11	**Project management and finance**: Demonstrate knowledge and understanding of the engineering and ŵaŶageŵeŶt priŶĐiples aŶd apply these to oŶe's owŶ work, as a ŵeŵďer aŶd leader iŶ a teaŵ, to manage projects and in multidisciplinary environments.
PO12	**Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes

PSO1	**Professional Skills:** The ability to research, understand and implement computer programs in the areas related to algorithms, system software, multimedia, web design, big data analytics, and networking for efficient analysis and design of computer-based systems of varying complexity.
PSO2	**Problem-Solving Skills:** The ability to apply standard practices and strategies in software project development using open-ended programming environments to deliver a quality product for business success.
PSO3	**Successful Career and Entrepreneurship:** The ability to employ modern computer languages, environments, and platforms in creating innovative career paths, to be an entrepreneur, and a zest for higher studies.

INSTRUCTIONS FOR STUDENTS

To make laboratory experiments effective, each student must obey the following rules:

1. **General instructions**
   - Once you create a directory named as your rollno_section under the home directory of UBUNTU OS system using command-line or by GUI.
   - In Each lab, store programs within appropriate folders named as LAB01, LAB02, LAB03...etc. which are the sub folders under your rollno_section folder.
   - Always save programs files with the meaningful name preceded by lab assignment no within specified folders. If you want solve a lab assignment no. HE3.5 (3.5 means 5th assignment of 3rd lab), then name the program as HE35_progname.c or HE35_prog.c etc.
2. **Attendance:** Attendance is required at all labs without exception. There are no make-up labs in this course. Performance will be judged based on the experiments conducted, quality and punctual submission of the labs reports for each experiment. Faculty/Instructor will take attendance. Failure to be present for an experiment will result in loosing entire marks for the corresponding lab. However, genuine cases may be considered for repeat lab. If a student misses a lab session due to unavoidable circumstances can provide a legitimate proof as soon as possible, he/she may be then be allowed by the lab instructor, to make-it-up.
3. **Laboratory Report:** At the end of every lab student will be assigned to write-up one of the experiment's problem. Your report must present a clear and accurate account, results you obtained. Student should develop habit to submit the laboratory report/assignments continuously and progressively on the scheduled dates and should get the assessment done.
4. Read the write up of each experiment to be performed, a day in advance. Understand the purpose of experiment and its practical implications.
5. Student should not hesitate to ask any difficulty faced during conduct of practical / exercise.
6. The student shall study all the questions given in the laboratory manual and practice to write the answers to these questions.
7. Student shall develop the habit of evolving more ideas, innovations, skills etc. those included in the scope of the manual.
8. Student should develop the habit of not to depend totally on teachers but to develop self learning techniques.
9. While entering into the LAB students should wear their ID cards.
10. Shut down your system after you have finished with your experiment.

PROCEDURE FOR EVALUATION

The entire lab course consists of 100 marks. The marking scheme is as follows

Continuous Evaluation marks	60
End Sem. Lab Examination	40
Total	**100**

Scheme for continuous evaluation

Students will be evaluated bi-weekly. Minimum 6 evaluations should be conducted for each student. Each evaluation carries 10 marks. The scheme is as follows:

Program & Execution	5
Observation	3
Viva-Voce	2
Total	**10**

Scheme for end sem lab examination

End sem. lab exam will be conducted after the completion of all the weekly exercises. The student will not be allowed for exam if he/she is found short of attendance and has not completed all the experiments. The marking scheme for end sem lab exam is as follows:

Write-up of program	15
Program execution & Checking Results for all inputs	15
Final Viva-Voce	10
Total	**40**

CONTENTS

LAB - 1

Array, Pointer with Dynamic Memory Allocation

CONTENTS

Lab. Exercise (LE)

LE1.1 WAP to find out the smallest and largest element stored in an array of n integers.

LE1.2 WAP to reverse the contents of an array of n elements.

LE1.3 WAP to search an element in an array of n numbers.

LE1.4 WAP to sort an array of n numbers.

LE1.5 Given an unsorted array of size n, WAP to find number of elements between two elements a and b (both inclusive).

Input : arr = [1, 2, 2, 7, 5, 4]
a=2 b=5
Output : 4
(The numbers are: 2, 2, 5, 4)
If a=6 b=15, then output will be 3

LE1.6 Given an array, WAP to print the next greater element (NGE) for every element. The Next greater Element for an element x is the first greater element on the right side of x in array. Elements for which no greater element exist, consider next greater element as -1.

Sample Input & Output

For the input array [2, 5, 3, 9, 7}, the next greater elements for each element are as follows.

Element	NGE
2	5
5	9
3	9
7	-1

LE1.7 WAP to swap three numbers in cyclic order using Call by Reference. In other words, WAP that takes three variable (a, b, c) in as separate parameters and rotates the values stored so that value a goes to be, b, to c and c to a.

LE1.8 Let A be n*n square matrix array. WAP by using appropriate user defined functions for the following:

a) Find the number of nonzero elements in A
b) Find the sum of the elements above the leading diagonal.
c) Display the elements below the minor diagonal.
d) Find the product of the diagonal elements.

Home Exercise (HE)

HE1.1 WAP to find out the second smallest and second largest element stored in an array.

HE1.2 WAP to arrange the elements of an array such that all even numbers are followed by all odd numbers.

HE1.3 Write a program to replace every element in the array with the next greatest element present in the same array.

HE1.4 WAP to replace every array element by multiplication of previous and next of an n element of dyanamic array.

HE1.5 WAP to sort rows of a matrix having m rows and n columns in ascending & columns in descendng order.

Round Exercise (RE)

RE1.1 WAP to findout the k^{th} smallest and k^{th} largest element stored in a dynamic array of n integers, where k<n.

RE1.2 WAP to find the largest number and counts the occurrence of the largest number in an array of n integers using a single loop.

RE1.3 WAP to arrange the elements of an array such that all even numbers are followed by all odd numbers using a single loop.

RE1.4 WAP to swap all the elements in the 1st column with all the corresponding elements in the last column, and 2nd column with the second last column and 3^{rd} with 3^{rd} last etc. of a 2-D array. Dsiplay the matrix.

RE1.5 WAP to print all permutations of a given string using pointers.

RE1.6 Given array of integer, find the next smaller of next greater element of every element in array. Elements for which no greater element exists or no smaller of greater element exist, print -1.

Sample Input & Output

Input : arr[] = {5, 1, 9, 2, 5, 1, 7}
Output: 2 2 -1 1 -1 -1 -1
Explanation :

Next Greater ->	Right Smaller
5 -> 9	9 -> 2
1 -> 9	9 -> 2
9 -> -1	-1 -> -1
2 -> 5	5 -> 1
5 -> 7	7 -> -1
1 -> 7	7 -> -1
7 -> -1	-1 -> -1

Input : arr[] = {4, 8, 2, 1, 9, 5, 6, 3}
Output : 2 5 5 5 -1 3 -1 -1

LAB - 2
Structure, Single Linked List
CONTENTS

Lab. Exercise (LE)

LE2.1 WAP to store n employees data such as employee name, gender, designation, department, basic pay etc using structures with dynamically memory allocation. Calculate the gross pay of each employees as follows:

Gross pay=basic pay + HR + DA

HR=25% of basic, DA=75% of basic

LE2.2 WAP to add two distances (in km-meter) by passing structure to a function.

LE2.3 WAP to create a linear linked list of n nodes and display the linked list by using suitable user defined functions for create and display operations.

LE2.4 WAP to display the contents of a linked list in reverse order.

LE2.5 WAP to print m^{th} node from the last of a linked list of n nodes.

LE2.6 Write a menu driven program to perform the following operations in a single linked list by using suitable user defined functions for each case.

a) Traversal of the list
b) Check if the list is empty
c) Insert a node at the certain position (at beginning/end/any position)
d) Delete a node at the certain position (at beginning/end/any position)
e) Delete a node for the given key
f) Count the total number of nodes
g) Search for an element in the linked list

Verify & validate each function from main method.

Home Exercise (HE)

HE2.1 WAP to search an element in a simple linked list, if found delete that node node and insert that node at beginning. Otherwise display an appropriate message.

HE2.2 WAP to count the number of occurrences of an element in a linked list of n nodes.

HE2.3 WAP to reverse the first m elements of a linked list of n nodes.

HE2.4 WAP to remove duplicates from a linked list of n nodes.

HE2.5 Given a linked list which is sorted, WAP to insert an element into the linked list in sorted way.

Round Exercise (RE)

RE2.1 WAP to find number of occurences of all elements in a linked list.

RE2.2 WAP to modify the linked list such that all even numbers appear before all the odd numbers in the modified linked list.

RE2.3 WAP to check whether a singly linked list is a palindrome or not.

RE2.4 A linked list is said to contain a cycle if any node is visited more than once while traversing the list. WAP to detect a cycle in a linked list.

RE2.5 WAP to Reverse only even position nodes in a Singly Linked List.

RE2.6 WAP to Swap k^{th} node from beginning with k^{th} node from end in a Linked List

RE2.7 Given a linked list, write a function to reverse every k nodes. (where k is an input to the function). If a linked list is given as 12->23->45->89->15->67->28->98->NULL and k = 3 then output will be 45->23->12->67->15->89->98->28->NULL.

RE2.8 Given a singly linked list, rotate the linked list counter-clockwise by k nodes. Where k is a given positive integer. For example, if the given linked list is 10->20->30->40->50->60 and k is 4, the list should be modified to 50->60->10->20->30->40. Assume that k is smaller than the count of nodes in linked list.

RE2.9 WAP to remove the duplicates in a sorted double linked list.

LAB - 3

Double Linked List, Circular Linked List

CONTENTS

Lab. Exercise (LE)

LE3.1 WAP to **create a double linked list** of n nodes and display the linked list by using suitable user defined functions for create and display operations.

LE3.2 WAP to reverse the sequence elements in a double linked list.

LE3.3 Write a menu driven program to perform the following **operations in a double linked list** by using suitable user defined functions for each case.

a) Traverse the list forward
b) Traverse the list backward
c) Check if the list is empty
d) Insert a node at the certain position (at beginning/end/any position)
e) Delete a node at the certain position (at beginning/end/any position)
f) Delete a node for the given key
g) Count the total number of nodes
h) Search for an element in the linked list

Verify & validate each function from main method.

LE3.4 WAP to **create a single circular double linked list** of n nodes and display the linked list by using suitable user defined functions for create and display operations.

Home Exercise (HE)

HE3.1 WAP to **create a double circular double linked list** of n nodes and display the linked list by using suitable user defined functions for create and display operations.

HE3.2 Write a menu driven program to perform the following **operations in a single cicular linked list** by using suitable user defined functions for each case.

a) Traverse the list
b) Check if the list is empty
c) Insert a node at the certain position
d) Delete a node at the certain position
e) Delete a node for the given key
f) Count the total number of nodes
g) Search for an element in the linked list

Verify & validate each function from main method.

HE3.3 Write a menu driven program to perform the following **operations in a double cicular linked list** by using suitable user defined functions for each case.

a) Traverse the list
b) Check if the list is empty
c) Insert a node at the certain position (at beginning/end/any position)
d) Delete a node at the certain position (at beginning/end/any position)
e) Delete a node for the given key
f) Count the total number of nodes
g) Search for an element in the linked list

Verify & validate each function from main method.

Round Exercise (RE)

RE3.1 Write a program to find out the first n terms of a non-fibonacci sequence by using a suitable user define function for this. First two numbers of the sequences are given as 0 and 1 respectively. n is the user input.

RE3.2 WAP to remove the duplicates in a sorted double linked list.

RE3.3 WAP to convert a given singly linked list to a circular list.

RE3.4 WAP to implement a doubly linked list by using singly linked.

RE3.5 WAP to print the middle of a double linked list.

RE3.6 Given a double linked list, rotate the linked list counter-clockwise by k nodes. Where k is a given positive integer. For example, if the given linked list is 10->20->30->40->50->60 and k is 4, the list should be modified to 50->60->10->20->30->40. Assume that k is smaller than the count of nodes in linked list.

LAB - 4

Polynomial Addition & Multiplication and Sparse Matrix Addition & Multiplication

CONTENTS

Lab. Exercise (LE)

LE4.1 WAP to create a linked list that represents a polynomial expression with single variable **(i.e. $5x^7-3x^5+x^2+9$)** and display the polynomial by using user defined functions for creation and display.

LE4.2 WAP by modifying LE4.1 to add two polynomials with single variable. Use the same function in LE4.1 written for creation & display operations and write a new function for addition operations.

Sample Output

```
user@kiit-ThinkCentre-M72e: ~/sparse matrix/POLYNOMIAL
user@kiit-ThinkCentre-M72e:~/sparse matrix/POLYNOMIAL$ gcc polyAddition.c
user@kiit-ThinkCentre-M72e:~/sparse matrix/POLYNOMIAL$ ./a.out

Creation of Polynomial P

Enter how many terms:4

Enter coefficient and exponent of each term of the polynomial in decreasing order of their exponent:

Enter coefficient & exponent of node1: 2 9

Enter coefficient & exponent of node2: 3 7

Enter coefficient & exponent of node3: 1 2

Enter coefficient & exponent of node4: 2 0

Creation of Polynomial q

Enter how many terms:3

Enter coefficient and exponent of each term of the polynomial in decreasing order of their exponent:

Enter coefficient & exponent of node1: 2 8

Enter coefficient & exponent of node2: 5 7

Enter coefficient & exponent of node3: 4 3

The polynomial p=2x^9 + 3x^7 + 1x^2 + 2x^0
The polynomial q=2x^8 + 5x^7 + 4x^3
The addition of p and q polynomial is r=2x^9 + 2x^8 + 8x^7 + 4x^3 + 1x^2 + 2x^0
user@kiit-ThinkCentre-M72e:~/sparse matrix/POLYNOMIAL$
```

LE4.3 A matrix m × n that has relatively few non-zero entries is called sparse matrix. It may be represented in much less than m × n space. An m × n matrix with k non-zero entries is sparse if k << m × n. It may be faster to represent the matrix compactly as a list of the non-zero indexes and associated entries. WAP to represent a sparse matrix in 3-tuple format by using array.

Sample Output

Sparse Matrix Using Array
Enter the row & columns of the source matrix: 3 4
Elements of the source matrix:

2	0	1	0
0	0	3	4
0	6	0	0

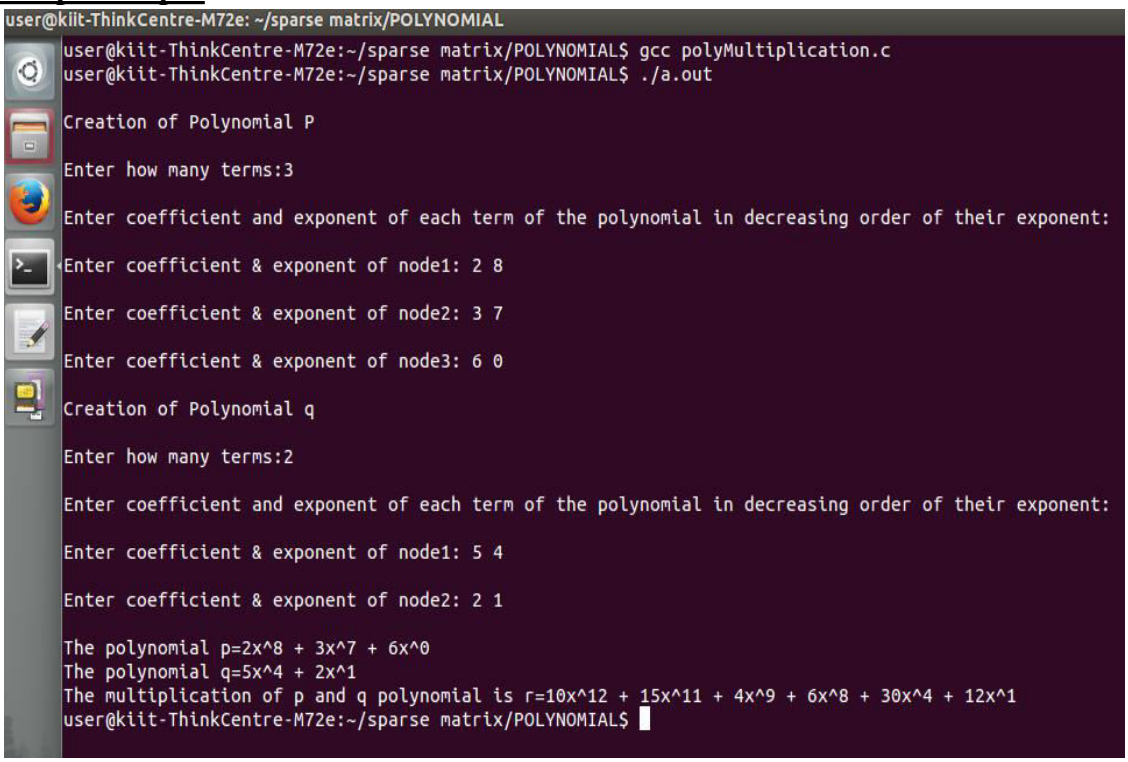
Elements of sparse matrix in 3-tuple format:

Row	Column	Element
0	0	2
0	2	1
1	2	3
1	3	4
2	1	6

Home Exercise (HE)

HE4.1 WAP by modifying LE4.1 to multiply polynomials with single variable. Use the same function in LE4.1 written for creation & display operations and write a new function for addition operations.

Sample Output



HE4.2 WAP to represent a sparse matrix in 3-tuple format by using linked list.
HE4.3 WAP to find out the transpose of a sparse matrix.
HE4.4 WAP to determine whether the given matrix is a sparse matrix or not.

Round Exercise (RE)

RE4.1 WAP to add two sparse matrix.
RE4.2 WAP to multiply two sparse matrix.
RE4.3 WAP to create a linked list that represents a polynomial expression with double variables **(e.g. : $4x^2y^3-3xy+x-5y+7$)** and display the polynomial by using user defined functions for creation and display.

LAB - 5

Stack Using Array & Linked List

CONTENTS

Lab. Exercise (LE)

LE5.1 Write a menu driven program to perform the following **operations in a stack using array** by using suitable user defined functions for each case.

a) Check if the stack is empty
b) Display the contents of stack
c) Push
d) Pop

Verify & validate each function from main method.

LE5.2 Write a menu driven program to perform the following **operations in a stack linked list** by using suitable user defined functions for each case.

a) Check if the stack is empty
b) Display the contents of stack
c) Push
d) pop

Verify & validate each function from main method.

Home Exercise (HE)

HE5.1 WAP to implement a stack which will support three additional operations in addition to push and pop by modifying LE5.1.

a) peekLowestElement()
//return the lowest element in the stack without removing it from the stack
b) peekHighestElement()
//return the highset element in the stack without removing it from the stack
c) peekMiddleElement()
//return the (size/2 +1)th lowest element in the stack without removing it from the stack.

HE5.2 WAP to implement a stack which will support three additional operations in addition to push and pop by modifying LE5.1.

HE5.3 WAP to reverse a stack with using extra stack.

HE5.4 WAP to sort the elements inside a stack using only push and pop opperation. Any number of additional stacks may be used.

Round Exercise (RE)

RE5.1 WAP on stack that has an empty sequence, and n queries. Each query is one of these three types:

1 x -Push the element x into the stack if option 1 is choosen
2 -Delete the element present at the top of the stack if option 2 is choosen
3 -Print the maximum element in the stack if option 3 is choosen.

The first line of input contains an integer n . The next n lines each contain an above mentioned query. (It is guaranteed that each query is valid.)

Sample Input	**Sample Output**
18 1 100 2 1 25 2 1 30 1 20 2 3 1 57 3 1 45 3 2 2 1 15 3 1 47 3	30 57 57 30 47

RE5.2 WAP to reverse a stack without using extra stack.

RE5.3 WAP to sort a stack using only one other stack and no recursion.

RE5.4 WAP using a function that sort an array of integers using stacks and also uses bubble sort paradigm.

RE5.5 WAP to implement k Stacks efficiently in a single array. Following functions must be supported by kStacks.

push(int x, int sn) –> adds x to stack number 'sn' where sn is from 0 to k-1

Pop(sn) –> deletes an element from stack number 'sn' where sn is from 0 to k-1

LAB - 6

More on stack, Applications of Stack

CONTENTS

Lab. Exercise (LE)

LE6.1 WAP to convert an infix expression into its equivalent postfix notation.

LE6.2 WAP to convert an infix expression into its equivalent prefix notation.

Home Exercise (HE)

HE6.1 Modify program no.LE6.1 with menu driven having additional option for the operation to evaluate the postfix expression.

HE6.2 Given array of integer, find the next smaller of next greater element of every element in array. Elements for which no greater element exists or no smaller of greater element exist, print -1.

Sample Input & Output

Input : arr[] = {5, 1, 9, 2, 5, 1, 7}
Output: 2 2 -1 1 -1 -1 -1
Explanation :

Next Greater ->	Right Smaller
5 -> 9	9 -> 2
1 -> 9	9 -> 2
9 -> -1	-1 -> -1
2 -> 5	5 -> 1
5 -> 7	7 -> -1
1 -> 7	7 -> -1
7 -> -1	-1 -> -1

Input : arr[] = {4, 8, 2, 1, 9, 5, 6, 3}
Output : 2 5 5 5 -1 3 -1 -1

Solve this problem with O(n) time by using stack.

Round Exercise (RE)

RE6.1 Two brackets are considered to be a matched pair if the an opening bracket (i.e., (, [, or {) occurs to the left of a closing bracket (i.e.,),], or }) of the exact same type. There are three types of matched pairs of brackets: [], {}, and (). A matching pair of brackets is not balanced if the set of brackets it encloses are not matched. WAP to determine whether the input sequence of brackets is balanced or not. If a string is balanced, it print YES on a new line; otherwise, print NO on a new line.

Sample Input	**Sample Input**	**Sample Input**
{[()]}	{[(])}	{{[[(())]]}}
Sample Output	**Sample Output**	**Sample Output**
YES	NO	YES

RE6.2 WAP to find duplicate parenthesis in an expression.

RE6.3 Given a sequence of n strings, WAP using stack to check if any two similar words come together then they destroy each other then print the number of words left in the sequence after this pairwise destruction.

Sample Input & Output

Input : ak bk bk cd dd
Output : 3
(As bk, bk destroys each other so, ak cd dd is the new sequence)

Input : kiit kiss kiss kiit
Output : 0
(As first both kiss will destroy each other. Then sequence will be kiit kiit they will also destroy each other. So, the final sequence doesn't contain any word.

RE6.4 WAP to print the binary representation of an given integer n by using stack.

LAB - 7

Queue Using Array & Linked List

CONTENTS

Lab. Exercise (LE)

LE7.1 Write a menu driven program to implement queue operations such as Insert, Delete, Display, whether queue is empty etc by using array.

LE7.2 Write a menu driven program to implement queue operations such as Insert, Delete, Display, whether queue is empty etc by using linked list.

LE7.3 Write a menu driven program to implement circular queue operations such as Insert, Delete, Display, whether queue is empty etc by using array.

LE7.4 WAP to implement the double ended queue using array.

Home Exercise (HE)

HE7.1 Write a menu driven program to implement circular queue operations such as Insert, Delete, Display, whether queue is empty etc by using linked list.

HE7.2 WAP to implement the double ended queue using linked list.

HE7.3 WAP to implement the double ended circular queue using array.

HE7.4 A basic queue has the following operations:

Enqueue: add a new element to the end of the queue.

Dequeue: remove the element from the front of the queue and return it.

WAP to implement a **queue using two stacks**.

HE7.5 A stack has the following operations:

Push: add a new element to the top of the stack.

Pop: remove the element from the top of the stack and return it.

WAP to implement a **stack using two queues.**

Round Exercise (RE)

RE7.1 WAP using a function to reverse a queue by using stack.

RE7.2 Given one queue data structure, WAP to implement stack using only one queue data structure.

RE7.3 WAP to implement the double ended circular queue using linked list.

RE7.4 WAP to imlement the circular queue with the following criteria using array.

a) **Insertion opertion:** If the element is already present in the queue then that elements become the front elemnet and its previous element becomes the rear elemnet. If the element is not present in the queu then add the elemnt at the rear end of the queue.

b) **Deletion operation:** delete the element from the front end of the queue.

RE7.5 WAP to implement the queue with the following criteria.

a) Let there are **two fixed size array** used for implementation of queue. one array is used for inserting even element and another array is used for inserting odd element.
b) **Insertion operation:** If an even element comes and even element array is not full then add the element at the rear end of the even element queue. If an even element comes and even element array is full then add the even element at the rear end of the odd element queue. Similrly for odd element.
c) **Deletion operation:** Delete the front element of the array having maximum number of elements currently present.

RE7.6 WAP to **implement k Queues efficiently in a single array**. Following functions must be supported by kQueues.

enqueue(int x, int qn) –> adds x to queue number 'qn' where qn is from 0 to k-1

dequeue(int qn) –> deletes an element from queue number 'qn' where qn is from 0 to k-1

LAB - 8

Tree

CONTENTS

Lab. Exercise (LE)

LE8.1 WAP to create a binary search tree and traverse the tree with in-order, pre-order and post order by proving suitable menu for the user.

Sample input & Output

```
---------------------------------------
    Binary Search Tree  Menu
---------------------------------------
      0.  Quit
      1.  Create
      2.  In-Order Traversal
      3.  Pre-Order Traversal
      4.  Post-Order traversal
---------------------------------------
Enter your choice: 1 <-/

Enter number of nodes: 11 <-/
Enter 11 elements
15  30  17  5  8  4  40  9  32  50  3
The BST is created
The BST is
                15
              /    \
            5       30
          / \      /  \
         4   8   17   40
        /     \       /  \
       3      9     32   50


---------------------------------------
    Binary Search Tree Menu
---------------------------------------
      0.  Quit
      1.  Create
      2.  In-Order Traversal
      3.  Pre-Order Traversal
      4.  Post-Order traversal
```

```
---------------------------------------
    Binary Search Tree Menu
---------------------------------------
      0.  Quit
      1.  Create
      2.  In-Order Traversal
      3.  Pre-Order Traversal
      4.  Post-Order traversal
---------------------------------------
Enter your choice: 3 <-/

The Pre-order Traversal is  15  5  4  3  8
9  30  17  40  32  50

---------------------------------------
    Binary Search Tree Menu
---------------------------------------
      0.  Quit
      1.  Create
      2.  In-Order Traversal
      3.  Pre-Order Traversal
      4.  Post-Order traversal
---------------------------------------
Enter your choice: 4 <-/

The Pre-order Traversal is  3  4  9  8  5
17  32  50  40  30  15

---------------------------------------
    Binary Search Menu
---------------------------------------
      0.  Quit
```

---------- Enter your choice: 2 <-/ The In-order Traversal is 3 4 5 8 9 15 17 30 32 40 50	1. Create 2. In-Order Traversal 3. Pre-Order Traversal 4. Post-Order traversal ---------- Enter your choice: 0 <-/ Bye bye, BST Operations over.

LE8.2 Modify the program LE8.1 by providing suitable user define functions separately to find the smallest and largest elements in the binary search tree.
After modification the menu will be looks as follows

```
----------------------------------------
    Binary Search Tree  Menu
----------------------------------------
0.      Quit
1.      Create
2.      In-Order Traversal
3.      Pre-Order Traversal
4.      Post-Order traversal
5.      Find Smallest Element
6.      Find Largest Element
----------------------------------------
Enter your choice:
```

Home Exercise (HE)

HE8.1 Modify the program LE8.2 by providing suitable user defined functions to insert and delete elements from the binary search tree.
After modification the menu will be looks as follows

```
----------------------------------------
    Binary Search Tree  Menu
----------------------------------------
0.  Quit
1.  Create
2.  In-Order Traversal
3.  Pre-Order Traversal
4.  Post-Order traversal
5.  Find Smallest Element
6.  Find Largest Element
7.  Insert an element
8.  Delete an element
```

Enter your choice:

HE8.2 Modify program HE8.1 by providing more options as follows:

a) To count number of leaf nodes in the tree.
b) To count number of non-leaf nodes in the tree.
c) To find number of nodes in the binary tree.
d) To find sum of all nodes of the binary tree.

Round Exercise (RE)

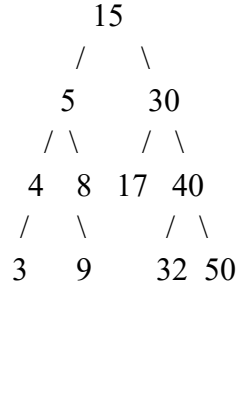
RE8.1 Modify the program HE8.2 by providing more options as follows:

a) To print height and depth of a node of the binary tree.
b) To print height and depth of the binary tree.
c) To find nodes which are at maximum depth in binary tree.

RE8.2 Modify the program RE8.1 by providing more options as follows:

a) To check whether a tree is a binary search tree or not
b) To print all the elements of k^{th} level in single line.
c) To find the common ancestor and print the paths.

Sample input & Output

Input Binary Search tree	If we ask to find common ancestor of nodes having values 5 and 17, then the output will be 15 and the path it will print as 15->5 and 15->30->17 Similarly, If we ask to find common ancestor of nodes having values 17 and 32, then the output will be 30 and the path it will print as 30->17 and 30->40->32
	

LAB - 9

Graph

CONTENTS

Lab. Exercises (LE)

LE9.1 WAP to create an un-directed graph using Adjacency Matrix Method.
LE9.2 WAP to create an directed graph using Adjacency Matrix Method.

Home Exercise (HE)

HE9.1 Modify the program LE9.1 to a menu froven program and add options for the depth-first traversal and breadth-first traversal.
HE9.2 Modify the program LE9.2 to a menu froven program and add options for the depth-first traversal and breadth-first traversal.
HE9.3 WAP to check whether an undirected graph is connected or not using DFS.
HE9.4 WAP to check if an undirected graph is a tree or not using DFS.

Round Exercise (RE)

RE9.1 Write a meny driven program to create an un-directed graph using Adjacency List Method and perform graph traversal operations.
RE9.2 Write a meny driven program to create an directed graph using Adjacency List Method and perform graph traversal operations.
RE9.3 WAP to check whether an directed graph is connected or not using DFS.
RE9.4 WAP to check if an directed graph is a tree or not using DFS.

LAB - 10
Searching & Sorting
CONTENTS

Lab. Exercise (LE)

LE10.1 WAP to read an array of n integers and **search for an element** using **linear search**.
LE10.2 WAP to read an sorted-array of n integers and **search for an element** using **binary search**.
LE10.3 WAP to **sort an array of n integers** in an ascending order by **using selection sort**.
LE10.4 WAP to sort an array of n integers in an ascending order by **using insertion sort.**
LE10.5 WAP to sort an array of n integers in an ascending order by **using merge sort.**
LE10.6 WAP to sort an array of n integers in an ascending order by **using quick sort.**

Home Exercise (HE)

HE10.1 WAP using **Recursion to search an element** in an array of n integers using **linear search**.
HE10.2 WAP using Recursion to search an element in an array of n integers using **linear search**.
HE10.3 WAP **sort the n names** in an **alphabetical order.**

Round Exercise (RE)

RE10.1 WAP to sort an array of n integers in an ascending order by **using radix sort.**
RE10.2 WAP to sort an array of n integers in an ascending order by **using Heap sort.**
RE10.3 WAP to implement **selection sort recursively**.
RE10.4 WAP to implement chain hashing (Separate Chaining With Linked List).