-

DEPARTMENT OF
ELECTRONICS AND COMMUNICATION ENGINEERING

EC8381 - FUNDAMENTALS OF DATA STRUCTURES IN C

III SEMESTER - R 2017

LABORATORY MANUAL

Name : _____

Register No. : _____

Section : _____

VISION

is committed to provide highly disciplined, conscientious and enterprising professionals conforming to global standards through value based quality education and training.

MISSION

- To provide competent technical manpower capable of meeting requirements of the industry
- To contribute to the promotion of Academic Excellence in pursuit of Technical Education at different levels
- To train the students to sell his brawn and brain to the highest bidder but to never put a price tag on heart and soul

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

VISION

To impart professional education integrated with human values to the younger generation, so as to shape them as proficient and dedicated engineers, capable of providing comprehensive solutions to the challenges in deploying technology for the service of humanity

MISSION

- To educate the students with the state-of-art technologies to meet the growing challenges of the electronics industry
- To carry out research through continuous interaction with research institutes and industry, on advances in communication systems
- To provide the students with strong ground rules to facilitate them for systematic learning, innovation and ethical practices

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

1. Fundamentals

To impart students with fundamental knowledge in Mathematics, Science and fundamentals of Engineering that will would them to be successful professionals

2. Core Competence

To provide students with sound knowledge in engineering and experimental skills to identify complex software problems in industry and to develop practical solution for them

3. Breadth

To provide relevant training and experience to bridge the gap between theory and practice this enables to find solutions for real time problem in industry and organization and to design products requiring interdisciplinary skills

4. Professionalism skills

To bestow students with adequate training and provide opportunities to work as team that will build up their communication skills, individual leadership and supportive qualities and to develop them to adapt and work in ever changing technologies

5. Lifelong Learning

To develop the ability of students to establish themselves as professionals in Computer Science and Engineering and to create awareness about the need for lifelong learning and pursuing advanced degrees

PROGRAMME OUTCOMES (POs)

a) To apply basic knowledge of Mathematics, Science and engineering fundamentals in Computer Science and Engineering field

b) To design and conduct experiments as well as to analyze and interpret and apply the same in the career

c) To design and develop innovative and creative software applications

d) To understand a complex real world problems and develop an efficient practical solutions

e) To create, select and apply appropriate technique, resources, modern engineering and IT tools

f) To understand their roles as professionals and give the best to the soicety

g) To develop a system that will meet expected need with realistic constraints such as economical, environmental, social, political, ethical, safe and sustainable

h) To communicate effectively and make others understand exactly what they are trying to convey in both verbal and written forms

i) To engage lifelong learning and exhibit their technical skills

j) To develop and manage projects in multidisciplinary environments

EC8381 - FUNDAMENTALS OF DATA STRUCTURES IN C SYLLABUS

COURSE OBJECTIVES

- To understand and implement basic data structures using C
- To apply linear and non-linear data structures in problem solving.
- To learn to implement functions and recursive functions by means of data structures
- To implement searching and sorting algorithms

LIST OF EXPERIMENTS:

1. Basic C Programs – looping, data manipulations, arrays
2. Programs using strings – string function implementation
3. Programs using structures and pointers
4. Programs involving dynamic memory allocations
5. Array implementation of stacks and queues
6. Linked list implementation of stacks and queues
7. Application of Stacks and Queues
8. Implementation of Trees, Tree Traversals
9. Implementation of Binary Search trees
10. Implementation of Linear search and binary search
11. Implementation Insertion sort, Bubble sort, Quick sort and Merge Sort
12. Implementation Hash functions, collision resolution technique

COURSE OUTCOMES

- To Write basic and advanced programs in c.
- To Implement functions and recursive functions in c.
- To Implement data structures using c.
- To choose appropriate sorting algorithm for an application and implement it in a modularized way.

EC8381 - FUNDAMENTALS OF DATA STRUCTURES IN C SYLLABUS

CONTENTS

Sl. No.	Name of the Experiment	Page No.
1	Basic C Programs – looping, data manipulations, arrays	7
2	Programs using strings – string function implementation	13
3	Programs using structures and pointers	15
4	Programs involving dynamic memory allocations	20
5	Array implementation of stacks and queues	22
6	Linked list implementation of stacks and queues	29
7	Application of Stacks and Queues	35
8	Implementation of Trees, Tree Traversals	39
9	Implementation of Binary Search trees	41
10	Implementation of Linear search and binary search	46
11	Implementation Insertion sort, Bubble sort, Quick sort and Merge Sort	50
12	Implementation Hash functions, collision resolution technique	58

EXPT. NO. 1a BASIC C PROGRAM – LOOPING

Aim:

To write a C program to calculate the sum of first n natural numbers using for loop statements

Software requirements:

C compiler

Hardware requirements:

Server with supporting 30 terminals

Algorithm :

1. Start the program.
2. Read the variables.
3. The initialization statement is executed.
4. The test expression is evaluated, if it is false for loop is terminated, test expression is true (nonzero), codes inside the body of loop is executed and the update expression is updated.
5. This process repeats until the test expression is false.
6. Stop the program.

Output

```
Enter a positive integer: 10

Sum = 55
```

Result:

Thus the program to find the sum of n natural numbers was executed successfully.

EXP 1 b. BASIC C PROGRAM – DATA MANIPULATIONS

Aim:

To write a C program for data manipulations using the bitwise operators

Software requirements:

C compiler

Hardware requirements:

Server with supporting 30 terminals

Algorithm :

1. Start the program.
2. Declare the variables.
3. Use the required bitwise operators for the given input.
4. Display the result.
5. Stop the program.

Sample Output :

A = 10
B = 25

Bitwise AND operator & = 8

A = 10
B = 25

Bitwise OR operator | = 29

A= 10
B = 25

Bitwise XOR (exclusive OR) operator ^ = 21

complement = ~35
complement = ~-12

complement = -36
Output = 11

Shift Operator (>> and <<)
Num = 212

Right Shift by 0: 212
Right Shift by 1: 106
Right Shift by 2: 53

Left Shift by 0: 212
Left Shift by 1: 424
Left Shift by 2: 848

Result :

Thus the program for data manipulations using the bitwise operators is executed successfully.

EXP 1 c BASIC C PROGRAM – ARRAYS

Aim:

To write a C program using Arrays

Software requirements:

C compiler

Hardware requirements:

Server with supporting 30 terminals

Algorithm :

1. Start the program.
2. Declare the array and the variables required for the program.
3. Get the values .
4. Perform the operation based on the program inputs.
5. Display the output.
6. Stop the program.

Sample Output :

```
Enter n: 5
Enter number1: 45
Enter number2: 35
Enter number3: 38
Enter number4: 31
Enter number5: 49

Average = 39
```

Result :

Thus the program using arrays is executed successfully.

Outcome :

Thus the basic c programs for looping, Data manipulations and arrays is attained.

- C language is used to creating computer application
- Used in writing embedded software
- Firmware for various electronics ,industrial product which uses micro controller
- Developing verifications of software code ,simulator.

VIVA - VOCE

1. What is Decision making and Branching ?

2. What are all the different types of control flow statements ?

3. Mention the different types of looping statements.

4. What is the difference between while and do while loop statement ?

5. What is an operator ?

6. Mention the bitwise operators.

7. What is an array ?

8 What is the use of array ?

9. What is multi dimensional array ?

10. How to access the elements in the array.

EXP NO 2 : PROGRAMS USING STRINGS FUNCTION

AIM:

To write a c program to perform string manipulation function.

Software requirements:

C compiler

Hardware requirements:

Server with supporting 30 terminals

Algorithm :

1. Start the program.
2. Declare the variables.
3. Get the string by using gets() function.
4. Call the required string manipulation function.
5. Display the results by using the puts() function.

Sample Output :

```
Enter string: String
Length of string  = 6
```

Result :

Thus the program for string manipulation is executed successfully.

Outcome :

Thus the programs using strings- string function implementation is attained.

Applications:

- String are used in spell checker, spam filter, intrusion detection

VIVA - VOCE

1. What is a function ?
2. Mention the types of function.
3. What is a character ?
4. What is a string ?
5. Difference between character and string.
6. What are the different types of string handling functions are available.
7. Which header file is to be included while using the string functions.
8. How to declare a string and character.

EXP NO 3 a : PROGRAMS USING POINTERS

Aim:

To write a c program to perform swapping operation using pointers.

Software requirements:

C compiler

Hardware requirements:

Server with supporting 30 terminals

Algorithm :

1. Start the program.

2. Declare the pointer variables.

3. Initialise the pointer variable with an appropriate address.

4. If the content of the address stored in the pointer is required, the indirection operator * is used to obtain the value.

5. Stop the program.

Sample Output :

Before Swapping : A = 10 B = 20

After Swapping : A = 20 B = 10

Result :

Thus the program to swap two numbers using pointers is executed successfully.

EXP NO 3 b : PROGRAMS USING STRUCTURES

Aim:

To write a c program using structures.

Software Requirements:

C compiler

Hardware Requirements:

Server with supporting 30 terminals

Algorithm :

1. Start the program.
2. Declare the structure and mention the structure variables inside the structure.
3. In the main program access the member of the structure using the dot operator.
4. Call the required functions inside the main function.
5. Display the output.
6. Stop the program.

Sample Output :

```
Enter No. of Employees : 2

Enter Employee Details

Enter Employee Id  : 436
Enter Employee Name :Gopal
Enter Basic Salary  : 10000

Enter Employee Details

Enter Employee Id  : 463
Enter Employee Name :Rajesh
Enter Basic Salary  : 22000

                        XYZ & Co. Payroll

*************************************************************************
EmpId   Name          Basic    HRA     DA      IT      Gross       Net Pay
*************************************************************************
 436    Gopal         10000   200.00  100.00  500.00  10300.00    9800.00
 463    Rajesh        22000   440.00  220.00 1100.00  22660.00    21560.00
*************************************************************************
```

Result :

Thus the program using structure is executed successfully.

Outcome :

Thus the programs using structures and pointers is attained.

Applications:

- The C pointer are used to track read /write position in files in Linux/Unix OS.

VIVA - VOCE

1. What is meant by pointers?
2. What is meant by structure?
3. What is the advantage of structure?
4. What is meant by address operator?

5. What are the usages of pointers?
6. Why you need pointers in C?
7. What is meant by referencing operator?
8. What are the ways to declare the structure?
9. What is meant by structure operator and arrow operator?
10. How to pass function to pointers?

Expt. No. 4 PROGRAMS INVOLVING DYNAMIC MEMORY ALLOCATION

Aim:

To write a c program to perform dynamic memory allocation.

Software requirements:

C compiler

Hardware requirements:

Server with supporting 30 terminals

Algorthim:

1. Start the program.
2. Include the required header file for using the memory allocation function.
3. Declare the required variables for the program.
4. Cal the required Dynamic Memory Allocation function depending upon the program.
5. Display the result.
6. Stop the program.

Sample Output :

```
Enter integer value: 100
Enter character value: x
Enter float value: 123.45
Inputted value are: 100, x, 123.45
```

Result:

Thus the program to perform dynamic memory allocation is executed successfully.

Outcome:

Thus the programs involving dynamic memory allocations are attained.

Applications:

The memory allocation concepts are used in operating systems present in embedded software .

VIVA - VOCE

1. What is memory allocation?
2. What is meant by dyanamic memory allocation?
3. Differentiate static memory allocation from dynamic memory allocation?
4. Give some examples for memory allocation?
5. What is the syntax for malloc()?
6. What is difference between malloc() and realloc()?
7. What is the function of calloc()?
8. What is meant by free() and what is the purpose of using free() fuction?
9. What is the significance of calloc()?
10. What is syntax for calloc()?

Expt. No. 5 a ARRAY IMPLEMENTATION OF STACK

Aim:

To write a C program to perform array implementation of stack.

Software Requirements:

C compiler

Hardware Requirements:

Server with supporting 30 terminals

Algorithm :

1. Start the program.

2. Include all the **header files** which are used in the program and define a constant **'SIZE'** with specific value.

3. Declare all the **functions** used in stack implementation.

4. Create a one dimensional array with fixed size (**int stack[SIZE]**)

5. Define a integer variable **'top'** and initialize with **'-1'**. (**int top = -1**)

6. In main method display menu with list of operations and make suitable function calls to perform operation selected by the user on the stack.

PUSH OPERATION

1. Check whether **stack** is **FULL**. (**top == SIZE-1**)

2. If it is **FULL**, then display **"Stack is FULL!!! Insertion is not possible!!!"** and terminate the function.

3. If it is **NOT FULL**, then increment **top** value by one (**top++**) and set stack[top] to value (**stack[top] = value**).

POP OPERATION

1. Check whether **stack** is **EMPTY**. (**top == -1**)

2. If it is **EMPTY**, then display **"Stack is EMPTY!!! Deletion is not possible!!!"** and terminate the function.

3. If it is **NOT EMPTY**, then delete **stack[top]** and decrement **top** value by one (**top--**)

DISPLAY

1.Check whether **stack** is **EMPTY**. (**top == -1**)

2. If it is **EMPTY**, then display **"Stack is EMPTY!!!"** and terminate the function.

3. If it is **NOT EMPTY**, then define a variable 'i' and initialize with top. Display **stack[i]** value and decrement **i** value by one (**i--**).

4. Repeat above step until **i** value becomes '0'.

Sample Output :

```
STACK      OPERATION
1.PUSH 2.POP 3.VIEW 4.QUIT Enter Choice : 1
Enter Stack element : 12

STACK      OPERATION
1.PUSH 2.POP 3.VIEW 4.QUIT Enter Choice : 1
Enter Stack element : 23

STACK      OPERATION

.PUSH 2.POP 3.VIEW 4.QUIT Enter Choice : 1
Enter Stack element : 34

STACK      OPERATION
1.PUSH     2.POP     3.VIEW     4.QUIT
Enter Choice : 1

Enter Stack element : 45
STACK      OPERATION
1.PUSH     2.POP     3.VIEW     4.QUIT
Enter Choice : 3

Top-->     45  34      23  12
STACK      OPERATION
1.PUSH     2.POP     3.VIEW     4.QUIT
Enter Choice : 2

Popped element is 45
STACK      OPERATION
1.PUSH     2.POP     3.VIEW     4.QUIT
Enter Choice : 3

Top-->  34  23  12
STACK      OPERATION
1.PUSH     2.POP     3.VIEW     4.QUIT
Enter Choice : 4
```

Result :

Thus the program for array implementation of stack is executed successfully.

Expt. No. 5 b ARRAY IMPLEMENTATION OF QUEUE

Aim:

To write a C program to perform array implementation of queue.

Software Requirements:

C compiler

Hardware Requirements:

Server with supporting 30 terminals

Algorithm :

1. Start the program.

2. Include all the **header files** which are used in the program and define a constant **'SIZE'** with specific value.

3. Declare all the **user defined functions** which are used in queue implementation.

4. Create a one dimensional array with above defined SIZE (**int queue[SIZE]**)

5. Define two integer variables **'front'** and **'rear'** and initialize both with **'-1'**. (**int front = -1, rear = -1**)

6. Then implement main method by displaying menu of operations list and make suitable function calls to perform operation selected by the user on queue.

ENQUEUE OPERATION

1. Check whether **queue** is **FULL**. (**rear == SIZE-1**)

2. If it is **FULL**, then display **"Queue is FULL!!! Insertion is not possible!!!"** and terminate the function.

3. If it is **NOT FULL**, then increment **rear** value by one (**rear++**) and set **queue[rear]** = **value**.

DEQUEUE OPERATION

1. Check whether **queue** is **EMPTY**. (**front == rear**)

2. If it is **EMPTY**, then display **"Queue is EMPTY!!! Deletion is not possible!!!"** and terminate the function.

3. If it is **NOT EMPTY**, then increment the **front** value by one (**front ++**). Then display **queue[front]** as deleted element. Then check whether both **front** and **rear** are equal (**front == rear**), if it **TRUE**, then set both **front** and **rear** to '**-1**' (**front = rear = -1**).

DISPLAY OPERATION

1. Check whether **queue** is **EMPTY**. (**front == rear**)

2. If it is **EMPTY**, then display **"Queue is EMPTY!!!"** and terminate the function.

3. If it is **NOT EMPTY**, then define an integer variable '**i**' and set '**i = front+1**'.

4. Display '**queue[i]**' value and increment '**i**' value by one (**i++**). Repeat the same until '**i**' value is equal to **rear** (**i <= rear**)

Sample Output :

```
QUEUE OPERATION
1.INSERT 2.DELETE 3.VIEW 4.QUIT Enter Choice : 1
Enter element to be inserted : 12

 QUEUE OPERATION
1.INSERT 2.DELETE 3.VIEW 4.QUIT Enter Choice : 1
Enter element to be inserted : 23

 QUEUE OPERATION
1.INSERT 2.DELETE 3.VIEW 4.QUIT Enter Choice : 1
Enter element to be inserted : 34

 QUEUE OPERATION
1.INSERT 2.DELETE 3.VIEW 4.QUIT Enter Choice : 1
Enter element to be inserted : 45

 QUEUE OPERATION
1.INSERT 2.DELETE 3.VIEW 4.QUIT Enter Choice : 1
Enter element to be inserted : 56

 QUEUE OPERATION
1.INSERT 2.DELETE 3.VIEW 4.QUIT Enter Choice : 1
Queue Full

 QUEUE OPERATION
1.INSERT 2.DELETE 3.VIEW 4.QUIT Enter Choice : 3
Front--> 12 23 34 45 56 <--Rear
```

Result :

Thus the program for array implementation of Queue is executed successfully.

Outcome:

Thus the program for array implementation of stack and queue is attained.

VIVA - VOCE

1. What is meant by data structure?
2. What are the types of data structure?
3. What is meant by stack?
4. What is meant by queue?
5. State the working principle of stack and queue?
6. What is difference between stack and queue?
7. What are the function of stack and queue?
8. What is the task of push and pop()?
9. Give some application of stack and queue?
10. What is stack overflow?

Expt. No. 6 a LINKED LIST IMPLEMENTATION OF STACK

Aim:

To write a C program to perform Linked List implementation of stack.

Software Requirements:

C compiler

Hardware Requirements:

Server with supporting 30 terminals

Algorithm :

1. Start the program.

2. Include all the **header files** which are used in the program. And declare all the **user defined functions**.

3. Define a '**Node**' structure with two members **data** and **next**.

4. Define a **Node** pointer '**top**' and set it to **NULL**.

5. Implement the **main** method by displaying Menu with list of operations and make suitable function calls in the **main** method.

PUSH OPERATION

1. Create a **newNode** with given value.

2. Check whether stack is **Empty** (**top** == **NULL**)

3. If it is **Empty**, then set **newNode** → **next** = **NULL**.

4. If it is **Not Empty**, then set **newNode** → **next** = **top**.

5. Finally, set **top** = **newNode**.

POP OPERATION

1. Check whether **stack** is **Empty** (**top == NULL**).

2. If it is **Empty**, then display **"Stack is Empty!!! Deletion is not possible!!!"** and terminate the function

3. If it is **Not Empty**, then define a **Node** pointer '**temp**' and set it to '**top**'.

4. Then set '**top = top → next**'.

5. Finally, delete '**temp**' (**free(temp)**).

DISPLAY OPERATION

1. Check whether stack is **Empty** (**top == NULL**).

2. If it is **Empty**, then display **'Stack is Empty!!!'** and terminate the function.

3. If it is **Not Empty**, then define a Node pointer **'temp'** and initialize with **top**.

4. Display '**temp → data --->**' and move it to the next node. Repeat the same until **temp** reaches to the first node in the stack (**temp → next != NULL**).

5. Finally! Display '**temp → data ---> NULL**'.

Sample Output :

```
  Stack using Linked List
1->Push 2->Pop 3->View 4->Exit Enter your choice : 1
Enter label for new node : 23 New node added
  Stack using Linked List
1->Push 2->Pop 3->View 4->Exit Enter your choice : 1
Enter label for new node : 34
  Stack using Linked List
1->Push 2->Pop 3->View 4->Exit Enter your choice : 3
HEAD -> 34 -> 23 -> NULL
```

Result :

Thus the program for Linked List implementation of Stack is executed successfully.

Expt. No. 6 b LINKED LIST IMPLEMENTATION OF QUEUE

Aim:

To write a C program to perform Linked List implementation of Queue.

Software Requirements:

C compiler

Hardware requirements:

Server with supporting 30 terminals

Algorithm :

1. Start the program.

2. Include all the header files which are used in the program. And declare all the user defined functions.

3. Define a 'Node' structure with two members data and next.

4. Define two Node pointers 'front' and 'rear' and set both to NULL.

5. Implement the main method by displaying Menu of list of operations and make suitable function calls in the main method to perform user selected operation.

enQueue OPERATION

1. Create a newNode with given value and set 'newNode → next' to NULL.

2. Check whether queue is Empty (rear == NULL)

3. If it is Empty then, set front = newNode and rear = newNode.

4. If it is Not Empty then, set rear → next = newNode and rear = newNode.

dEQueue OPERATION

1. Check whether queue is Empty (front == NULL).

2. If it is Empty, then display "Queue is Empty!!! Deletion is not possible!!!" and terminate from the function

3. If it is Not Empty then, define a Node pointer 'temp' and set it to 'front'.

4. Then set 'front = front → next' and delete 'temp' (free(temp)).

DISPLAY OPERATION

1. Check whether queue is Empty (front == NULL).

2. If it is Empty then, display 'Queue is Empty!!!' and terminate the function.

3. If it is Not Empty then, define a Node pointer 'temp' and initialize with front.

4. Display 'temp → data --->' and move it to the next node. Repeat the same until 'temp' reaches to 'rear' (temp → next != NULL).

5. Finally! Display 'temp → data ---> NULL'.

Sample Output

```
Queue using Linked List
1->Push 2->Pop 3->View 4->Exit Enter your choice : 1
Enter label for new node : 23 New node added
Stack using Linked List
1->Push 2->Pop 3->View 4->Exit Enter your choice : 1
Enter label for new node : 34
Stack using Linked List
1->Push 2->Pop 3->View 4->Exit Enter your choice : 3
HEAD -> 34 -> 23 -> NULL
```

Result :

Thus the program for Linked List implementation of Queue is executed successfully.

Outcome :

Thus the program for Linked List implementation of stack and queue is attained.

Applications:

- Thread scheduler which maintains process in the memory .Linked list implementation is used move running process in queue which is used thread scheduler.

VIVA - VOCE

1. What is the meant by linear data structure?
2. What is meant by linked list?
3. What is stack implementation of linked list?
4. What is meant by node?
5. What are the operations of stack in the linked list?
6. How to add the node and delete the node in the linked list using stack?
7. How to access the element from the stack in linked list?
8. What are the operations of queue using linked list?
9. How to allocate and deallocate the memory for linked list?
10. What is meant by singly linked list?

Expt. No. 7 a APPLICATION OF STACKS

Aim:

To write a C program to perform infix to postfix expression

Software Requirements:

C compiler

Hardware Requirements:

Server with supporting 30 terminals

Algorithm :

1. Start the program.

2. Read all the symbols one by one from left to right in the given Infix Expression.

3. If the reading symbol is operand, then directly print it to the result (Output).

4. If the reading symbol is left parenthesis '(', then Push it on to the Stack.

5. If the reading symbol is right parenthesis ')', then Pop all the contents of stack until respective left parenthesis is poped and print each poped symbol to the result.

6. If the reading symbol is operator (+ , - , * , / etc.), then Push it on to the Stack. However, first pop the operators which are already on the stack that have higher or equal precedence than current operator and print them to the result.

Sample Output :

INFIX EXPRESSION : (A + B) * (C - D)

POSTFIX EXPRESSION : *A B + C D - **

Result :

Thus the program for performing infix expression to postfix expression is executed successfully.

Expt. No. 7 b APPLICATION OF STACKS

Aim:

To write a C program to perform postfix to infix expression

Software Requirements:

C compiler

Hardware Requirements:

Server with supporting 30 terminals

Algorithm :

1. Start the program.

2. Read all the symbols one by one from left to right in the given Postfix Expression

3. If the reading symbol is operand, then push it on to the Stack.

4. If the reading symbol is operator (+ , - , * , / etc.,), then perform TWO pop operations and store the two popped oparands in two different variables (operand1 and operand2). Then perform reading symbol operation using operand1 and operand2 and push result back on to the Stack.

5. Finally! perform a pop operation and display the popped value as final result.

Sample Output :

POSTFIX EXPRESSION : (5+3)*(8-2)

INFIX EXPRESSION : 53+82-*

Result :

Thus the program for performing postfix expression to infix expression is executed successfully.

Outcome :

Thus the program for application of stack and queue is attained.

Applications:

- In high level programming language, the arithmetic expressions are calculated. The stacks are used to convert the expression which is understood by the computer.

VIVA - VOCE

1. What are the applications of stack?
2. What is meant by infix expression, Give example?
3. Give some real application of stack?
4. What is the procedure to evaluate the infix to postfix using stack?
5. Which data structure is needed to convert infix notation to postfix notation?
6. What is the result of Top (Push (S, X))?
7. Whether Linked List is linear or Non-linear data structure?
8. What is linked list?
9. List out the types of linked list.
10. What is circular linked list?

Expt. No. 8 IMPLEMENTATION OF TREES, TREE TRAVERSALS

Aim:

To write a C program for performing Binary Tree Traversal.

Software Requirements:

C compiler

Hardware Requirements:

Server with supporting 30 terminals

Algorithm :

1. Start the program.

PREORDER TRAVERSAL

2. Process the root node (N).
3. Traverse the left subtree of N (L).
4. Traverse the right subtree of N (R).

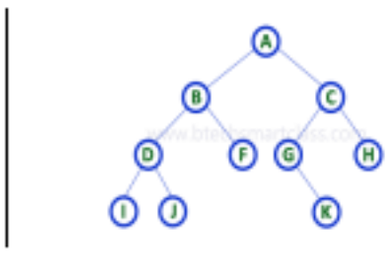
INORDER TRAVERSAL

5. Traverse the left subtree of N (L).
6. Process the root node (N).
7. Traverse the right subtree of N (R).

POSTORDER TRAVERSAL

8. Traverse the left subtree of N (L).
9. Traverse the right subtree of N (R).
10. Process the root node (N).

Sample Output :



In order traversal sequence : I - D - J - B - F - A - G - K - C - H

Post order traversal sequence : I - J - D - F - B - K - G - H - C - A

Pre order traversal sequence : A - B - D - I - J - F - C - G - K - H

Result :

Thus the program for performing binary tree traversal is executed successfully.

Outcome :

Thus the program for implementation of Trees, Tree Traversals is attained.

Applications:

- Manipulate hierarchical data
- Manipulate sorted list of data
- As a workflow for computing digital image for visual effects

Expt. No. 9 IMPLEMENTATION OF BINARY SEARCH TREES

Aim:

To write a C program to performing Binary Search Tree Operations.

Software Requirements:

C compiler

Hardware Requirements:

Server with supporting 30 terminals

Algorithm :

1. Start the program.

Search Operation in BST

2. Read the search element from the user

3. Compare, the search element with the value of root node in the tree.

4. If both are matching, then display "Given node found!!!" and terminate the function

5. If both are not matching, then check whether search element is smaller or larger than that node value.

6. If search element is smaller, then continue the search process in left subtree.

7. If search element is larger, then continue the search process in right subtree.

8. Repeat the same until we found exact element or we completed with a leaf node

9. If we reach to the node with search value, then display "Element is found" and terminate the function.

10. If we reach to a leaf node and it is also not matching, then display "Element not found" and terminate the function.

Insertion Operation in BST

1. Create a newNode with given value and set its **left** and **right** to **NULL**.

2. Check whether tree is Empty.

3. If the tree is **Empty**, then set set **root** to **newNode**.

4. If the tree is **Not Empty**, then check whether value of newNode is **smaller** or **larger** than the node (here it is root node).

5. If newNode is **smaller** than **or equal** to the node, then move to its **left** child. If newNode is **larger** than the node, then move to its **right** child.

6. Repeat the above step until we reach to a **leaf** node (e.i., reach to NULL).

7. After reaching a leaf node, then isert the newNode as **left child** if newNode is **smaller or equal** to that leaf else insert it as **right child**.

Deletion Operation in BST

Case 1: Deleting a Leaf node (A node with no children)

Case 2: Deleting a node with one child

Case 3: Deleting a node with two children

Case 1: Deleting a leaf node

1. **Find** the node to be deleted using **search operation**

2. Delete the node using **free** function (If it is a leaf) and terminate the function.

Case 2: Deleting a node with one child

1. **Find** the node to be deleted using **search operation**

2. If it has only one child, then create a link between its parent and child nodes.

3. Delete the node using **free** function and terminate the function.

Case 3: Deleting a node with two children

1. **Find** the node to be deleted using **search operation**

2. If it has two children, then find the **largest** node in its **left subtree** (OR) the **smallest** node in its **right subtree**.

3. **Swap** both **deleting node** and node which found in above step.

4. Then, check whether deleting node came to **case 1** or **case 2** else goto steps 2

5. If it comes to **case 1**, then delete using case 1 logic.

6. If it comes to **case 2**, then delete using case 2 logic.

7. Repeat the same process until node is deleted from the tree.

Sample Output :

```
Select an option
1- insert.
2- Search.
3- Delete.
4 – Display.
5 - Exit

Enter your choice : 1
Enter the node value : 6

Enter your choice : 1
Enter the node value : 1

Enter your choice : 1
Enter the node value : 5

Enter your choice : 1
Enter the node value : 2

Enter your choice : 1
Enter the node value : 4

Enter your choice : 1
Enter the node value : 3

Select an option
1- insert.
2- Search.
3- Delete.
4 – Display.
5 - Exit

Enter your choice : 4
1
2
3
4
5
6
```

Result :

Thus the program for performing Binary Search Tree Operations is executed successfully.

Outcome :

Thus the program for implementation of Binary Search Trees is attained.

Applications:

- Used in search application where data is constantly entered and stored
- Binary search partition used in 3D video game.
- Binary tries – Used in high bandwidth router for storing routing data.
- Syntax tree- Used for construction of compiler and calculator to parse expression

Expt. No. 10 a IMPLEMENTATION OF LINEAR SEARCH

Aim:

To write a C program to perform Linear Search Operation.

Software Requirements:

C compiler

Hardware Requirements:

Server with supporting 30 terminals

Algorithm :

1. Start the program.

2. Read the search element from the user

3. Compare, the search element with the first element in the list.

4. If both are matching, then display "Given element found!!!" and terminate the function

5. If both are not matching, then compare search element with the next element in the list.

6. Repeat steps 3 and 4 until the search element is compared with the last element in the list.

7. If the last element in the list is also doesn't match, then display "Element not found!!!" and terminate the function.

Sample Output :

```
Enter number of elements : 7
 Enter Array Elements :
23 6 12 5 0 32 10
 Enter element to locate : 5
 Element found at position 3
```

Result :

Thus the program to perform Linear search operation is executed successfully.

Expt. No. 10 b IMPLEMENTATION OF BINARY SEARCH

Aim:

To write a C program to perform Binary Search Operation.

Algorithm :

1. Start the program.

2. Read the search element from the user

3. Find the middle element in the sorted list

4. Compare, the search element with the middle element in the sorted list.

5. If both are matching, then display "Given element found!!!" and terminate the function

6. If both are not matching, then check whether the search element is smaller or larger than middle element.

7. If the search element is smaller than middle element, then repeat steps 2, 3, 4 and 5 for the left sublist of the middle element.

8. If the search element is larger than middle element, then repeat steps 2, 3, 4 and 5 for the right sublist of the middle element.

9. Repeat the same process until we find the search element in the list or until sublist contains only one element.

10. If that element also doesn't match with the search element, then display "Element not found in the list!!!" and terminate the function.

Sample Output :

```
Enter array size : 10
Elements in Sorted Order
      0 2 4 6 8 10 12 14 16 18
Enter element to locate : 16
  Found at index 8 in 2 attempts
```

Result :

Thus the program to perform binary search operation is executed successfully.

Outcome :

Thus the implementation of Linear search and Binary search is attained.

Applications:

- Searching used in information retrieval.

Expt. No. 11 a IMPLEMENTATION OF INSERTION SORT

Aim:

To write a C program to perform Insertion Sorting.

Software Requirements:

C compiler

Hardware Requirements:

Server with supporting 30 terminals

Algorithm :

1. Start the program.

2. Asume that first element in the list is in sorted portion of the list and remaining all elements are in unsorted portion.

3. Consider first element from the unsorted list and insert that element into the sorted list in order specified.

4. Repeat the above process until all the elements from the unsorted list are moved into the sorted list.

Sample Output :

Enter total elements: 6

Enter array elements: 34 8 64 51 32 21

After Pass 1:	8	34	64	51	32	21
After Pass 2:	8	34	64	51	32	21
After Pass 3:	8	34	51	64	32	21
After Pass 4:	8	32	34	51	64	21
After Pass 5:	8	21	32	34	51	64

Sorted List : 8 21 32 34 51 64

Result :

Thus the program to perform insertion sort is executed successfully.

Expt. No. 11 b IMPLEMENTATION OF BUBBLE SORT

Aim:

To write a C program to perform Bubble Sorting.

Software Requirements:

C compiler

Hardware Requirements:

Server with supporting 30 terminals

Algorithm :

1. Start the program.
2. Set the size of the array and declare the variables.
3. Repeat the steps 4 – 9 while i > 1
4. set j=0.
5. Repeat steps 6 – 8 while j < i-1.
6. if arr[j] is greater than arr[j+1] goto step 7 else goto step 8.
7. Swap the elements stored at arr[j] and arr[j+1] and perform the operations.
8. set j=j + 1.
9. set i= i -1.
10. stop.

Sample Output :

```
Enter the number of elements in the array :
5

Enter the 5 elements to sort :
18
3
2
33
21

The sorted elements are :

2
3
18
21
33
```

Result :

Thus the program to perform bubble sort is executed successfully.

Expt. No. 11 c IMPLEMENTATION OF QUICK SORT

Aim:

To write a C program to perform Quick Sorting.

Software Requirements:

C compiler

Hardware Requirements:

Server with supporting 30 terminals

Algorithm :

1. Start the program.

2. 1. If n < = 1, then return.

2. Pick any element V in a[]. This is called the pivot.

3. Rearrange elements of the array by moving all elements xi > V right of V and all elements xi < = V left of V. If the place of the V after re-arrangement is j, all elements with value less than V, appear in a[0], a[1] . . . . a[j – 1] and all those with value greater than V appear in a[j + 1] . . . . a[n – 1].

4. Apply quick sort recursively to a[0] . . . . a[j – 1] and to a[j + 1] . . . . a[n – 1].

5. Stop the program.

Sample Output :

```
Enter the number of elements in the array :
6

Enter the 6 elements to sort :
34
99
5
2
57
40

The sorted elements are :

2
5
34
40
57
99
```

Result :

Thus the program to perform Quick sort is executed successfully.

Expt. No. 11 d IMPLEMENTATION OF MERGE SORT

Aim:

To write a C program to perform Merge Sorting.

Algorithm :

1. Start the program.

2. Divide the unsorted list into N sublists, each containing 1 element.

3. Take adjacent pairs of two singleton lists and merge them to form a list of 2 elements. N will now convert into N/2 lists of size 2.

4. Repeat the process till a single sorted list of obtained.

Sample Output :

```
Enter the number of elements in the array :
8

Enter the 6 elements to sort :
34
99
5
2
57
40
8
29

The sorted elements are :

2
5
8
29
34
40
57
99
```

Result :

Thus the program to perform Merge sort is executed successfully.

Outcome :

Thus the implementation of Insertion sort, Bubble sort, Quick sort, Merge sort is attained.

Applications:

- Merge sort – Merge sort are used in external database.
- Bubble Sort-Used in programming for TV remote application
- Heap sort –Used in reading bar codes in plastic cards
- Quick Sort –Sports scores are organized by quick sort.

Expt. No. 12 a IMPLEMENTATION OF HASH FUNCTION

Aim:

To write a C program to perform Hash Functions.

Software Requirements:

C compiler

Hardware Requirements:

Server with supporting 30 terminals

Algorithm :

1. Start the program.

2. Declare the variables.

3. An element is converted into an integer by using a hash function. This element can be used as an index to store the original element, which falls into the hash table.

4. The element is stored in the hash table where it can be quickly retrieved using hashed key.

```
hash = hashfunc(key)
index = hash % array_size
```

Sample Output :

Enter the next key : 10765

The hash key generarted for the key 10765 is 765.

Result :

Thus the program to perform hash function is executed successfully.

Expt. No. 12 a COLLISION RESOLUTION TECHNIQUES

Aim:

To write a C program to perform Linear probing.

Software Requirements:

C compiler

Hardware Requirements:

Server with supporting 30 terminals

Algorithm :

1. Start the program.

2. Create an array of structure (i.e a hash table).

3. Take a key and a value to be stored in hash table as input.

4. Corresponding to the key, an index will be generated i.e every key is stored in a particular array index.

5. Using the generated index, access the data located in that array index.

6. In case of absence of data, create one and insert the data item (key and value) into it and increment the size of hash table.

7. In case the data exists, probe through the subsequent elements (looping back if necessary) for free space to insert new data item.
Note: This probing will continue until we reach the same element again (from where we began probing)

8. To display all the elements of hash table, element at each index is accessed (via for loop).

9. To remove a key from hash table, we will first calculate its index and delete it if key matches, else probe through elements until we find key or an empty space where not a single data has been entered (means data does not exist in the hash table).

10. Exit

Sample Output :

```
MENU-:
1. Inserting item in the Hashtable
2. Removing item from the Hashtable
3. Check the size of Hashtable
4. Display Hashtable

Please enter your choice-: 3
Size of Hashtable is-: 0

Do you want to continue-:(press 1 for yes) 1
Implementation of Hash Table in C with Linear Probing
MENU-:
1. Inserting item in the Hashtable
2. Removing item from the Hashtable
3. Check the size of Hashtable
4. Display Hashtable

Please enter your choice-: 1
Inserting element in Hashtable
Enter key and value-:   12        10

 Key (12) has been inserted

Do you want to continue-:(press 1 for yes) 1
Implementation of Hash Table in C with Linear Probing
MENU-:
1. Inserting item in the Hashtable
2. Removing item from the Hashtable
3. Check the size of Hashtable
4. Display Hashtable

Please enter your choice-: 1
Inserting element in Hash table
Enter key and value-:   122       4

 Key (122) has been inserted

Do you want to continue-:(press 1 for yes) 1
Implementation of Hash Table in C with Linear Probing
MENU-:
1. Inserting item in the Hashtable
2. Removing item from the Hashtable
3. Check the size of Hashtable
4. Display Hashtable

Please enter your choice-: 3
Size of Hashtable is-: 2
```

```
Do you want to continue-:(press 1 for yes) 1
Implementation of Hash Table in C with Linear Probing
MENU-:
1. Inserting item in the Hashtable
2. Removing item from the Hashtable
3. Check the size of Hashtable
4. Display Hashtable

Please enter your choice-: 4
Array[0] has  no elements

Array[1] has no elements

Array[2] has elements-:
12 (key) and 10 (value)

Array[3] has elements-:
122(key) and 5(value)

Array[4] has no elements

Array[5] has no elements

Array[6] has no elements

Array[7] has no elements

Array[8] has no elements

Array[9] has no elements


Do you want to continue-:(press 1 for yes) 1
Implementation of Hash Table in C with Linear Probing
MENU-:
1. Inserting item in the Hashtable
2. Removing item from the Hashtable
3. Check the size of Hashtable
4. Display Hashtable

Please enter your choice-: 2
Deleting in Hashtable
Enter the key to delete-: 122

 Key (122) has been removed

Do you want to continue-:(press 1 for yes) 1
Implementation of Hash Table in C with Linear Probing
MENU-:
1. Inserting item in the Hashtable
```

2. Removing item from the Hashtable
3. Check the size of Hashtable
4. Display Hashtable

Please enter your choice-: 2
Deleting in Hashtable
Enter the key to delete-: 56

This key does not exist

Do you want to continue-:(press 1 for yes) 2

Result :

Thus the program to perform linear probing is executed successfully.

Outcome :

Thus the implementation of Hash functions, collision resolution techniques is initiated.

Applications:

- Constructs a message Authentication code
- Used to construct digital signature
- Used in message digest as a cryptographic function
- Time stamping