Department of Information Technology
BABU BANARASI DAS
Institute of Technology & Management, Lucknow
Recognized by AICTE, Govt.of India & Affiliated to Dr. A.P.J Abdul Kalam Technical University,Lucknow,AKTU College Code=054

LAB MANUAL
OF
DATA STRUCTURE LAB

[BCS351]

B. TECH, 2nd Year Semester - 3rd

Dr. A.P.J. Abdul Kalam Tech. University
Uttar Pradesh

2025-26

Department of Information Technology

Faculty Name: Ms. Garima Jain

(Assistant Professor)

Approved by: (Head of the Department)

MANUAL CONTENTS

This manual is intended for the 2nd year students of Information Technology in the subject of Data Structure Lab . This manual typically contains practical/lab sessions related Data Structure covering various aspects related to the subject of enhanced understanding.

Students are advised to thoroughly go through this manual rather than only topics mentioned in the syllabus as practical aspects are the key to understanding and conceptual visualization of theoretical aspects covered in the books.

Good Luck for your Enjoyable Laboratory Sessions.

PREFACE

This practical manual will be helpful for students of Information Technology for understanding the course from the point of view of applied aspects. Though all the efforts have been made to make this manual error free, yet some errors might have crept in inadvertently. Suggestions from the readers for the improvement of the manual are most welcomed.

VISION OF THE INSTITUTE

"To become a leading institute of providing professionally competent and socially responsive technocrats with high moral values."

MISSION OF THE INSTITUTE

M1 : To create an eco-system for the dissemination of technical knowledge, to achieve academic excellence.

M2 : To develop technocrats with creative skills and leadership qualities, to solve local and global challenges.

M3 : To impart human values and ethics in students, to make them socially and Eco-friendly responsible

.

VISION OF THE DEPARTMENT

"To transform the students to be skilled IT professionals, Innovative leaders and environmentally receptive citizens"

MISSION OF THE DEPARTMENT

M1: To develop and deliver quality academic programs in emerging & innovative fields of engineering to empower the students to meet industry standards.

M2: To build students community with ethical standards to undertake R&D in thrust areas of national & international needs.

M3: To inculcate professional behavior, positive attitude and communication skills.

Program Educational Objectives(PEOs) of Department

PEO1: To develop the ability among students to understand the concept of core Information Technology subjects that will facilitate understanding of any new technology.

PEO2: To embed a strong foundation in the engineering fundamentals to solve analyze and design real time engineering products.

PEO3: To give exposures to emerging edge technologies adequate training and opportunities to work as a team on multi-disciplinary projects with effective communication skills and leadership qualities.

Program Outcomes:(PO)

Graduates will be able to achieve

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. Engineering 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 The World: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to professional engineering practice.

PO7.Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO8.Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO 9.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

PO10.Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO11. 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(PSOs) of the Department

PSO1: Use and apply current technical concepts and practices in the core Information Technologies of Human Machine Learning, Information Management, Programming, Cloud Computing.

PSO2: Design, develop and test software systems for worldwide network of computers to provide solutions to real world problems.

Course Evaluation Scheme

Sr No	Subject Code	Subject Name	Periods			Evaluation Scheme				Total	Credit
			L	T	P	Sessional Assessment			PE		
						CT	TA	PS			
1.	BCS351	Data Structure lab	0	0	2	-	-	50	50	100	1

Course Objectives:

The teacher will explain:

1.	Understand and implement fundamental data structures such as arrays, linked lists, stacks, queues, trees, and graphs using a programming language
2.	Analyze the efficiency of different data structure implementations and algorithms with respect to time and space complexity.
3.	Enhance problem-solving and debugging skills through practical implementation and testing of data structure algorithms.

Pre- requisite: Fundamentals of Computer Science and Programming, Basic Algorithms.

Course Outcomes (COs)

Course Outcomes: The students should be able to:		**Bloom's Level**
CO1	Implement programs using array	L3
CO2	Implement different sorting and searching algorithms	L3
CO3	Implement the Stack , Queue and their applications	L3
CO4	Implement graph using BFs and DFS	L3
CO5	Implement tree using BST and its traversal	L3

CO-PO-PSO Mapping

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PSO1	PSO2
CO1	3	2	3	-	3	-	-	-	-	-	2	2	1
CO2	3	3	2	-	2	-	-	-	-	-	2	2	2
CO3	3	3	3	-	2	-	-	-	-	-	2	2	1
CO4	3	3	3	-	2	-	-	-	-	-	2	2	1
CO5	3	3	3	-	3	-	-	-	-	-	2	2	1
Avg.													

The extent of mapping is as follows: 1 for low, 2 for moderate, 3 for high &"-" for No correlation between CO & PO.

List of Experiments

S. No.	Experiment
1	**Array & Linked List**
1.1	Implement a Program to Separate odd and even integers into Separate Array
1.2	Implement a Program to find out max and min element in an array
1.3	Implement a Program in C to insert an element at a specific position from an array
1.4	Implement a Program to in c to delete an element from an array
1.5	Implement a Program for adding two matrix of same size
1.6	Implement a Program in C for transpose of two matrix
1.7	Implement Singly Linked List and perform the following operations on Singly Linked list 1. Insertion 2. Deletion 3. Creation
2	**Stack & Queue**
2.1	Implement Stack using Array
2.2	Implement Stack using Linked List
2.3	Implement Queue using Array
2.4	Implement Queue using Linked List
3	**Searching & Sorting**
3.1	Implement Linear Search
3.2	Implement Binary Search
3.2	Implement Bubble Sort
3.3	Implement Selection SOrt
3.4	Implement Insertion Sort
3.5	Implement Merge Sort
3.6	Implement Quick Sort
4	**Trees**
4.1	Implement Binary Search Tree
4.2	Implement Tree Traversal method using non recursive
5	**Graph**
5.1	Implement Adjacency List
5.2	Implement Adjacency Matrix

Beyond Syllabus:

S.No.	Experiment
1	Design the algorithm that sorts elements by repeatedly reducing the gap between compared items.
2	Implement Quick Data retrieval Method

INDEX

S.No.	Name of the Experiment	CO	BTL	Lab Conduction Date	Remark/ Grade/ Marks	Faculty Signature with Date
1	**Array & Linked List**					
1.1	Implement a Program to Separate odd and even integers into Separate Array	CO1	L3	15/9		
1.2	Implement a Program to find out max and min element in an array	CO1	L3	15/9		
1.3	Implement a Program in C to insert an element at a specific position from an array	CO1	L3	15/9		
1.4	Implement a Program to in c to delete an element from an array	CO1	L3	15/9		
1.5	Implement a Program for adding two matrix of same size	CO1	L3	23/9		
1.6	Implement a Program in C for transpose of two matrix	CO1	L3	23/9		
1.7	Implement Singly Linked List and perform the following operations on Singly Linked list 4. Insertion 5. Deletion 6. Creation	CO1	L3	23/9		
2	**Stack & Queue**					
2.1	Implement Stack using Array	CO2	L3	30/09		
2.2	Implement Stack using Linked List	CO2	L3	30/09		
2.3	Implement Queue using Array	CO2	L3	28/10		
2.4	Implement Queue using Linked List	CO2	L3	28/10		
3	**Searching & Sorting**					

3.1	Implement Linear Search	CO3	L3	04/11		
3.2	Implement Binary Search	CO3	L3	04/11		
3.2	Implement Bubble Sort	CO3	L3	04/11		
3.3	Implement Selection SOrt	CO3	L3	04/11		
3.4	Implement Insertion Sort	CO3	L3	11/11		
3.5	Implement Merge Sort	CO3	L3	11/11		
3.6	Implement Quick Sort	CO3	L3	11/11		
4	**Trees**					
4.1	Implement Binary Search Tree	CO4	L3	25/11		
4.2	Implement Tree Traversal method using non recursive	CO4	L3	02/12		
5	**Graph**					
5.1	Implement Adjacency List	CO5	L3	09/12		
5.2	Implement Adjacency Matrix	CO5	L3	09/12		

Softwares & Hardwares Required:

1. **Operating System :** Windows,Linux,macOS.
2. **Software : MSDos**
3. **Hardwar Processor** : At least **Intel i3, RAM**: A minimum of **8 GB RAM**.

Do's

1. Know the location of the fire extinguisher and the first aid box how to use them in case of an emergency.
2. Read and understand how to carry out an activity thoroughly before coming to the laboratory.
3. Report files or accidents to your lecturer/laboratory technician immediately.
4. Report any broken plugs or exposed electrical wires to your lecturer/laboratory technician immediately.

Don'ts

1. Do not eat or drink in the laboratory.
2. Avoid stepping on electrical wires or any other computer cobles
3. Do not open the system unit casing or monitor casing particularly when the power is turned on. Some internal components hold electric voltages of up to 3000 volts, which can be fatal.
4. Do not insert metal objects such as clips, pins, and needles into the computer casings. They may cause fire.
5. Do not remove anything from computer laboratory without permission.
6. Do not touch, connect, or disconnect any plug or cable without your lecturer/laboratory technician's permission.

Experiment-1.1

Objective:

Implement a Program to Separate odd and even integers into Separate Array

Procedure:

```
#include <stdio.h>
int main() {
int arr[100], even[100], odd[100];
int n, i;
int eCount = 0, oCount = 0;
printf("Enter number of elements: ");
scanf("%d", &n);
printf("Enter integers separated by spaces:\n");
for(i = 0; i < n; i++) {
scanf("%d", &arr[i]);
}
for(i = 0; i < n; i++) {
if(arr[i] % 2 == 0) {
even[eCount++] = arr[i];
} else {
odd[oCount++] = arr[i];
}
}
printf("\nOriginal array: [");
for(i = 0; i < n; i++) {
printf("%d", arr[i]);
if(i < n - 1) printf(", ");
}
printf("]\n");
printf("Even integers: [");
for(i = 0; i < eCount; i++) {
printf("%d", even[i]);
if(i < eCount - 1) printf(", ");
}
printf("]\n");
printf("Odd integers: [");
for(i = 0; i < oCount; i++) {
printf("%d", odd[i]);
if(i < oCount - 1) printf(", ");
}
printf("]\n");
return 0;
}
```

Output :

```
Enter integers separated by spaces:
10 15 22 33 40 51 64 77 88 99
Original array: [10, 15, 22, 33, 40, 51, 64, 77, 88, 99]
Even integers: [10, 22, 40, 64, 88]
Odd integers: [15, 33, 51, 77, 99]
```

Experiment- 1.2

Objective:
Implement a Program to find out max and min element in an array

Procedure:

```
#include <stdio.h>
int main() {
int arr[100], n, i;
int max, min;
printf("Enter number of elements: ");
scanf("%d", &n);
printf("Enter integers separated by spaces:\n");
for(i = 0; i < n; i++) {
scanf("%d", &arr[i]);
}
max = min = arr[0];
Exp 1.3 :
Exp 1.4 :
for(i = 1; i < n; i++) {
if(arr[i] > max) max = arr[i];
if(arr[i] < min) min = arr[i];
}
printf("\nThe array is: [");
for(i = 0; i < n; i++) {
printf("%d", arr[i]);
if(i < n - 1) printf(", ");
}
printf("]\n");
printf("Maximum element: %d\n", max);
printf("Minimum element: %d\n", min);
return 0;
}
```

```
Enter integers separated by spaces:
34 12 78 4 99 23 56
The array is: [34, 12, 78, 4, 99, 23, 56]
Maximum element: 99
Minimum element: 4
```

Experiment- 1.3

Objective:

Implement a Program in C to insert an element at a specific position from an array

Procedure:

```
#include <stdio.h>
int main() {
int arr[100], n, i, pos, val;
printf("Enter number of elements: ");
scanf("%d", &n);
printf("Enter elements:\n");
for(i = 0; i < n; i++) {
scanf("%d", &arr[i]);
}
printf("Enter position to insert (1 to %d): ", n + 1);
scanf("%d", &pos);
printf("Enter value to insert: ");
scanf("%d", &val);
if(pos < 1 || pos > n + 1) {
printf("Invalid position\n");
return 0;
}
for(i = n; i >= pos; i--) {
arr[i] = arr[i - 1];
}
arr[pos - 1] = val;
n++;
printf("Array after insertion: [");
for(i = 0; i < n; i++) {
printf("%d", arr[i]);
if(i < n - 1) printf(", ");
}
printf("]\n");
return 0;
}
```

Experiment- 1.4

Objective : Implement a Program to in c to delete an element from an array

Procedure:

```
#include <stdio.h>
int main() {
int arr[100], n, i, pos;
printf("Enter number of elements: ");
scanf("%d", &n);
```

```
printf("Enter elements:\n");
for(i = 0; i < n; i++) {
scanf("%d", &arr[i]);
}
printf("Enter position to delete (1 to %d): ", n);
scanf("%d", &pos);
if(pos < 1 || pos > n) {
printf("Invalid position\n");
return 0;
}
for(i = pos - 1; i < n - 1; i++) {
arr[i] = arr[i + 1];
}
n--;
printf("Array after deletion: [");
for(i = 0; i < n; i++) {
printf("%d", arr[i]);
if(i < n - 1) printf(", ");
}
printf("\n");
return 0;
}
```

```
PS C:\Users\Dev\Projects\Arrays> .\delete_element
Enter number of elements: 6
Enter 6 elements:
10 20 30 40 50 60
Enter the position to delete (1-based index): 4

Original array: 10 20 30 40 50 60
Array after deletion: 10 20 30 50 60
```

Experiment- 1.5

Objective : Implement a Program for adding two matrix of same size

Procedure :

```
#include <stdio.h>
int main() {
int a[10][10], b[10][10], sum[10][10];
int r, c, i, j;
printf("Enter number of rows and columns: ");
scanf("%d %d", &r, &c);
printf("Enter elements of first matrix:\n");
for(i = 0; i < r; i++) {
```

```
for(j = 0; j < c; j++) {
scanf("%d", &a[i][j]);
}
}
printf("Enter elements of second matrix:\n");
for(i = 0; i < r; i++) {
for(j = 0; j < c; j++) {
scanf("%d", &b[i][j]);
}
}
for(i = 0; i < r; i++) {
for(j = 0; j < c; j++) {
sum[i][j] = a[i][j] + b[i][j];
}
}
printf("Sum of matrices:\n");
for(i = 0; i < r; i++) {
for(j = 0; j < c; j++) {
printf("%d ", sum[i][j]);
}
printf("\n");
}
return 0;
}
```

```
PS C:\Users\Dev\Projects\Matrices> .\add_matrices
Enter number of rows: 2
Enter number of columns: 3

Enter elements of first matrix (2 x 3):
1 2 3
4 5 6

Enter elements of second matrix (2 x 3):
6 5 4
3 2 1

First matrix:
1 2 3
4 5 6

Second matrix:
6 5 4
3 2 1

Sum of the two matrices:
7 7 7
7 7 7
```

Experiment- 1. 6

Objective : Implement a Program in C for transpose of two matrix

Procedure :

```
#include <stdio.h>
int main() {
int a[10][10], t[10][10];
int r, c, i, j;
printf("Enter number of rows and columns: ");
scanf("%d %d", &r, &c);
printf("Enter elements of the matrix:\n");
for(i = 0; i < r; i++) {
for(j = 0; j < c; j++) {
scanf("%d", &a[i][j]);
}
}
for(i = 0; i < r; i++) {
for(j = 0; j < c; j++) {
t[j][i] = a[i][j];
}
```

```
}
printf("Transpose of the matrix:\n");
for(i = 0; i < c; i++) {
for(j = 0; j < r; j++) {
printf("%d ", t[i][j]);
}
printf("\n");
}
return 0;
}
```

```
PS C:\Users\Dev\Projects\Matrices> .\transpose_matrix
Enter number of rows: 2
Enter number of columns: 3

Enter elements of the matrix (2 x 3):
1 2 3
4 5 6

Original matrix:
1 2 3
4 5 6

Transpose of the matrix (3 x 2):
1 4
2 5
3 6
```

Experiment- 1.7

Objective : Implement Singly Linked List and perform the following operations on Singly Linked list
Insertion
Deletion
Creation

Procedure :

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
int data;
struct Node *next;
};
struct Node *head = NULL;
void insertAtEnd(int val) {
struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data = val;
newNode->next = NULL;
if(head == NULL) {
head = newNode;
return;
}
struct Node *temp = head;
while(temp->next != NULL) {
temp = temp->next;
}
temp->next = newNode;
}
void display() {
struct Node *temp = head;
while(temp != NULL) {
printf("%d -> ", temp->data);
temp = temp->next;
}
printf("NULL\n");
}
int main() {
int n, i, val;
printf("Enter number of nodes: ");
scanf("%d", &n);
for(i = 0; i < n; i++) {
```

```
printf("Enter value: ");
scanf("%d", &val);
insertAtEnd(val);
}
printf("Singly Linked List:\n");
display();
return 0;
}
```

```
PS C:\Users\Dev\Projects\LinkedList> .\singly_linked_list

Singly Linked List Implementation in C
--------------------------------------
1. Insert at beginning
2. Insert at end
3. Delete from beginning
4. Delete from end
5. Display list
6. Exit
Enter your choice: 1
Enter value to insert at beginning: 10
Node inserted.

Enter your choice: 2
Enter value to insert at end: 20
Node inserted.

Enter your choice: 2
Enter value to insert at end: 30
Node inserted.

Enter your choice: 5
Current List: 10 -> 20 -> 30 -> NULL

Enter your choice: 3
Node deleted from beginning.

Enter your choice: 5
Current List: 20 -> 30 -> NULL

Enter your choice: 4
Node deleted from end.

Enter your choice: 5
Current List: 20 -> NULL

Enter your choice: 6
Exiting...
```

Experiment- 2.1

Objective : Implement Stack using Array

Procedure :

```
#include <stdio.h>
#define MAX 100
int stack[MAX];
int top = -1;
void push(int val) {
if (top == MAX - 1) {
printf("Stack Overflow\n");
return;
}
stack[++top] = val;
}
void pop() {
if (top == -1) {
printf("Stack Underflow\n");
return;
}
printf("Popped element: %d\n", stack[top--]);
}
void peek() {
if (top == -1) {
printf("Stack is empty\n");
return;
}
printf("Top element: %d\n", stack[top]);
}
void display() {
if (top == -1) {
printf("Stack is empty\n");
return;
}
for (int i = top; i >= 0; i--) {
```

```
printf("%d ", stack[i]);
}
printf("\n");
}
int main() {
int choice, val;
do {
printf("\n1.Push 2.Pop 3.Peek 4.Display 5.Exit\n");
printf("Enter choice: ");
scanf("%d", &choice);
switch (choice) {
case 1:
printf("Enter value: ");
scanf("%d", &val);
push(val);
break;
case 2:
pop();
break;
case 3:
peek();
break;
case 4:
display();
break;
case 5:
break;
default:
printf("Invalid choice\n");
}
} while (choice != 5);
return 0;
}
```

```
PS C:\Users\Dev\Projects\Stack> .\stack_array

Stack Implementation using Array in C
-------------------------------------
1. Push
2. Pop
3. Peek (Top element)
4. Display
5. Exit
Enter your choice: 1
Enter value to push: 10
Pushed 10 onto the stack.

Enter your choice: 1
Enter value to push: 20
Pushed 20 onto the stack.

Enter your choice: 1
Enter value to push: 30
Pushed 30 onto the stack.

Enter your choice: 4
Stack elements (top to bottom): 30 20 10

Enter your choice: 3
Top element is: 30

Enter your choice: 2
Popped 30 from the stack.

Enter your choice: 4
Stack elements (top to bottom): 20 10

Enter your choice: 5
Exiting...
```

Experiment- 2.2

Objective : To write a program to find the Physical Address for a given IP addresses using Simulation.

Procedure :

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
int data;
struct Node* next;
};
struct Node* top = NULL;
```

```
void push(int x) {
struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
newNode->data = x;
newNode->next = top;
top = newNode;
printf("Pushed %d\n", x);
}
void pop() {
if (top == NULL) {
printf("Stack Underflow\n");
} else {
struct Node* temp = top;
printf("Popped %d\n", top->data);
top = top->next;
free(temp);
}
}
void display() {
struct Node* temp = top;
if (top == NULL) {
printf("Stack is empty\n");
} else {
printf("Stack elements: ");
while (temp != NULL) {
printf("%d ", temp->data);
temp = temp->next;
}
printf("\n");
}
```

```
}
int main() {
int choice, value;
while (1) {
printf("\n1. Push\n2. Pop\n3. Display\n4. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1:
printf("Enter value to push: ");
scanf("%d", &value);
push(value);
break;
case 2:
pop();
break;
case 3:
display();
break;
case 4:
return 0;
default:
printf("Invalid choice\n");
}
}
}
```

```
PS C:\Users\Dev\Projects\Stack> .\stack_linkedlist

Stack Implementation using Linked List in C
-------------------------------------------
1. Push
2. Pop
3. Peek (Top element)
4. Display
5. Exit
Enter your choice: 1
Enter value to push: 10
Pushed 10 onto the stack.

Enter your choice: 1
Enter value to push: 20
Pushed 20 onto the stack.

Enter your choice: 1
Enter value to push: 30
Pushed 30 onto the stack.

Enter your choice: 4
Stack elements (top to bottom): 30 20 10

Enter your choice: 3
Top element is: 30

Enter your choice: 2
Popped 30 from the stack.

Enter your choice: 4
Stack elements (top to bottom): 20 10

Enter your choice: 5
Exiting...
```

Experiment-2.3

Objective : Implement Queue using Array.

Procedure :

```
#include <stdio.h>
#define MAX 5
int queue[MAX];
int front = -1, rear = -1;
void enqueue(int x) {
if (rear == MAX - 1) {
printf("Queue Overflow\n");
} else {
if (front == -1) {
front = 0;
}
rear++;
queue[rear] = x;
printf("Enqueued %d\n", x);
}
}
```

```
void dequeue() {
if (front == -1 || front > rear) {
printf("Queue Underflow\n");
} else {
printf("Dequeued %d\n", queue[front]);
front++;
if (front > rear) {
front = rear = -1;
}
}
}
void display() {
if (front == -1 || front > rear) {
printf("Queue is empty\n");
} else {
printf("Queue elements: ");
for (int i = front; i <= rear; i++) {
printf("%d ", queue[i]);
}
printf("\n");
}
}
void enqueue(){
if (back == -1 || rear )
}
int main() {
int choice, value;
while (1) {
printf("\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1:
printf("Enter value to enqueue: ");
scanf("%d", &value);
```

```
enqueue(value);
break;
case 2:
dequeue();
break;
case 3:
display();
break;
case 4:
return 0;
default:
printf("Invalid choice\n");
}
}
}
```

```
PS C:\Users\Dev\Projects\Queue> .\queue_array

Queue Implementation using Array in C
-------------------------------------
1. Enqueue
2. Dequeue
3. Peek (Front element)
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 10
Enqueued 10 into the queue.

Enter your choice: 1
Enter value to enqueue: 20
Enqueued 20 into the queue.

Enter your choice: 1
Enter value to enqueue: 30
Enqueued 30 into the queue.

Enter your choice: 4
Queue elements (front to rear): 10 20 30

Enter your choice: 3
Front element is: 10

Enter your choice: 2
Dequeued 10 from the queue.

Enter your choice: 4
Queue elements (front to rear): 20 30

Enter your choice: 5
Exiting...
```

Experiment-2.4

Objective : Implement Queue using Linked List

Procedure :

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
int data;
struct Node* next;
};
struct Node *front = NULL;
struct Node *rear = NULL;
void enqueue(int x) {
struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
if (newNode == NULL) {
printf("Queue Overflow\n");
return;
}
newNode->data = x;
newNode->next = NULL;
if (rear == NULL) {
front = rear = newNode;
} else {
rear->next = newNode;
rear = newNode;
}
printf("Enqueued %d\n", x);
}
void dequeue() {
if (front == NULL) {
printf("Queue Underflow\n");
} else {
struct Node* temp = front;
printf("Dequeued %d\n", front->data);
```

```
front = front->next;
if (front == NULL) {
rear = NULL;
}
free(temp);
}
}
void display() {
struct Node* temp = front;
if (front == NULL) {
printf("Queue is empty\n");
} else {
printf("Queue elements: ");
while (temp != NULL) {
printf("%d ", temp->data);
temp = temp->next;
}
printf("\n");
}
}
int main() {
int choice, value;
while (1) {
printf("\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1:
printf("Enter value to enqueue: ");
scanf("%d", &value);
enqueue(value);
break;
case 2:
dequeue();
break;
```

```
case 3:
display();
break;
case 4:
return 0;
default:
printf("Invalid choice\n");
}
}
}
```

```
PS C:\Users\Dev\Projects\Queue> .\queue_linkedlist

Queue Implementation using Linked List in C
-------------------------------------------
1. Enqueue
2. Dequeue
3. Peek (Front element)
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 10
Enqueued 10 into the queue.

Enter your choice: 1
Enter value to enqueue: 20
Enqueued 20 into the queue.

Enter your choice: 1
Enter value to enqueue: 30
Enqueued 30 into the queue.

Enter your choice: 4
Queue elements (front to rear): 10 20 30

Enter your choice: 3
Front element is: 10

Enter your choice: 2
Dequeued 10 from the queue.

Enter your choice: 4
Queue elements (front to rear): 20 30

Enter your choice: 5
Exiting...
```

Experiment-3.1

Objective : Implement Linear Search

Procedure :

```
#include <stdio.h>
int linearSearch(int arr[], int size, int key) {
int i;
for (i = 0; i < size; i++) {
if (arr[i] == key) {
return i;
}
}
return -1;
int main() {
int arr[100], size, key, index;
printf("Enter size of array: ");
scanf("%d", &size);
printf("Enter %d elements:\n", size);
for (int i = 0; i < size; i++) {
scanf("%d", &arr[i]);
}
printf("Enter element to search: ");
scanf("%d", &key);
index = linearSearch(arr, size, key);
if (index != -1) {
printf("Element %d found at position %d\n", key, index + 1);
} else {
printf("Element %d not found in array\n", key);
}
return 0;
}
```

```
PS C:\Users\Dev\Projects\Search> .\linear_search
Enter number of elements: 7
Enter 7 elements:
5 12 7 25 19 3 10
Enter element to search: 25

Array: 5 12 7 25 19 3 10
Element 25 found at position 4
```

Experiment-3.2

Objective : Implement Binary Search

Procedure :

```
#include <stdio.h>
int binary_search(int arr[], int size, int key) {
int low = 0;
int high = size - 1;
while (low <= high) {
int mid = low + (high - low) / 2;
if (arr[mid] == key) {
return mid;
}
if (arr[mid] < key) {
low = mid + 1;
}
else {
high = mid - 1;
}
}
return -1;
}
int main() {
int arr[100], size, key, position;
printf("Enter the size of the array (max 100): ");
scanf("%d", &size);
printf("Enter %d elements in SORTED order:\n", size);
```

```
for (int i = 0; i < size; i++) {
scanf("%d", &arr[i]);
}
printf("Enter the element to search: ");
scanf("%d", &key);
position = binary_search(arr, size, key);
if (position != -1) {
printf("Element %d found at position %d (index %d)\n", key, position + 1,
position);
} else {
printf("Element %d not found in the array.\n", key);
}
return 0;
}
```

```
PS C:\Users\Dev\Projects\Search> .\binary_search
Enter number of elements: 8
Enter 8 elements in sorted order:
3 7 10 15 21 28 36 42
Enter element to search: 21

Array: 3 7 10 15 21 28 36 42
Element 21 found at position 5
```

Experiment-3.3

Objective : Implement Bubble Sort

Procedure :

```
#include <stdio.h>
void bubbleSort(int arr[], int size) {
int i, j, temp;
for (i = 0; i < size - 1; i++) {
for (j = 0; j < size - i - 1; j++) {
if (arr[j] > arr[j+1]) {
temp = arr[j];
arr[j] = arr[j+1];
arr[j+1] = temp;
}
```

```
}
}
}
void printArray(int arr[], int size) {
int i;
for (i = 0; i < size; i++) {
printf("%d ", arr[i]);
}
printf("\n");
}
int main() {
int arr[100], size, i;
printf("Enter the number of elements: ");
scanf("%d", &size);
printf("Enter %d elements:\n", size);
for (i = 0; i < size; i++) {
scanf("%d", &arr[i]);
}
bubbleSort(arr, size);
printf("Sorted array using Bubble Sort:\n");
printArray(arr, size);
return 0;
}
```

```
PS C:\Users\Dev\Projects\Sorting> .\bubble_sort
Enter number of elements: 7
Enter 7 elements:
64 25 12 22 11 90 34

Unsorted array: 64 25 12 22 11 90 34
Sorted array using Bubble Sort: 11 12 22 25 34 64 90
```

Experiment-3.4

Objective : Implement Selection SOrt

Procedure :

```
#include <stdio.h>
void selectionSort(int arr[], int size) {
```

```
int i, j, minIndex, temp;
for (i = 0; i < size - 1; i++) {
minIndex = i;
for (j = i + 1; j < size; j++) {
if (arr[j] < arr[minIndex]) {
minIndex = j;
}
}
temp = arr[i];
arr[i] = arr[minIndex];
arr[minIndex] = temp;
}
}
void printArray(int arr[], int size) {
int i;
for (i = 0; i < size; i++) {
printf("%d ", arr[i]);
}
printf("\n");
}
int main() {
int arr[100], size, i;
printf("Enter the number of elements: ");
scanf("%d", &size);
printf("Enter %d elements:\n", size);
for (i = 0; i < size; i++) {
scanf("%d", &arr[i]);
}
selectionSort(arr, size);
printf("Sorted array using Selection Sort:\n");
printArray(arr, size);
return 0;
}
```

```
PS C:\Users\Dev\Projects\Sorting> .\selection_sort
Enter number of elements: 7
Enter 7 elements:
64 25 12 22 11 90 34

Unsorted array: 64 25 12 22 11 90 34
Sorted array using Selection Sort: 11 12 22 25 34 64 90
```

Experiment-3.5

Objective : Implement Insertion Sort

Procedure :

```
#include <stdio.h>
void insertionSort(int arr[], int size) {
int i, j, key;
for (i = 1; i < size; i++) {
key = arr[i];
j = i - 1;
while (j >= 0 && arr[j] > key) {
arr[j + 1] = arr[j];
j--;
}
arr[j + 1] = key;
}
}
void printArray(int arr[], int size) {
int i;
for (i = 0; i < size; i++) {
printf("%d ", arr[i]);
}
printf("\n");
}
int main() {
int arr[100], size, i;
printf("Enter the number of elements: ");
scanf("%d", &size);
```

```
printf("Enter %d elements:\n", size);
for (i = 0; i < size; i++) {
scanf("%d", &arr[i]);
}
insertionSort(arr, size);
printf("Sorted array using Insertion Sort:\n");
printArray(arr, size);
return 0;
}
```

```
PS C:\Users\Dev\Projects\Sorting> .\insertion_sort
Enter number of elements: 7
Enter 7 elements:
64 25 12 22 11 90 34

Unsorted array: 64 25 12 22 11 90 34
Sorted array using Insertion Sort: 11 12 22 25 34 64 90
```

Experiment-3.6

Objective : Implement Merge Sort

Procedure :

```
#include <stdio.h>
void merge(int arr[], int left, int mid, int right) {
int i, j, k;
int n1 = mid - left + 1;
int n2 = right - mid;
int leftArr[n1], rightArr[n2];
for (i = 0; i < n1; i++)
leftArr[i] = arr[left + i];
for (j = 0; j < n2; j++)
rightArr[j] = arr[mid + 1 + j];
i = 0;
j = 0;
k = left;
while (i < n1 && j < n2) {
if (leftArr[i] <= rightArr[j]) {
```

```
arr[k] = leftArr[i];
i++;
} else {
arr[k] = rightArr[j];
j++;
}
k++;
}
while (i < n1) {
arr[k] = leftArr[i];
i++;
k++;
}
while (j < n2) {
arr[k] = rightArr[j];
j++;
k++;
}
}
void mergeSort(int arr[], int left, int right) {
if (left < right) {
int mid = left + (right - left) / 2;
mergeSort(arr, left, mid);
mergeSort(arr, mid + 1, right);
merge(arr, left, mid, right);
}
}
void printArray(int arr[], int size) {
int i;
for (i = 0; i < size; i++) {
printf("%d ", arr[i]);
}
printf("\n");
}
int main() {
```

```
int arr[100], size, i;
printf("Enter the number of elements: ");
scanf("%d", &size);
printf("Enter %d elements:\n", size);
for (i = 0; i < size; i++) {
scanf("%d", &arr[i]);
}
mergeSort(arr, 0, size - 1);
printf("Sorted array using Merge Sort:\n");
printArray(arr, size);
return 0;
}
```

```
PS C:\Users\Dev\Projects\Sorting> .\merge_sort
Enter number of elements: 8
Enter 8 elements:
38 27 43 3 9 82 10 15

Unsorted array: 38 27 43 3 9 82 10 15
Sorted array using Merge Sort: 3 9 10 15 27 38 43 82

PS C:\Users\Dev\Projects\Sorting> _
```

Experiment-3.7

Objective : Implement Quick Sort

Procedure :

```
#include <stdio.h>
void swap(int *a, int *b) {
int temp = *a;
*a = *b;
*b = temp;
}
int partition(int arr[], int low, int high) {
int pivot = arr[high];
int i = (low - 1);
for (int j = low; j <= high - 1; j++) {
if (arr[j] <= pivot) {
```

```
i++;
swap(&arr[i], &arr[j]);
}
}
swap(&arr[i + 1], &arr[high]);
return (i + 1);
}
void quickSort(int arr[], int low, int high) {
if (low < high) {
int pi = partition(arr, low, high);
quickSort(arr, low, pi - 1);
quickSort(arr, pi + 1, high);
}
}
void printArray(int arr[], int size) {
for (int i = 0; i < size; i++) {
printf("%d ", arr[i]);
}
printf("\n");
}
int main() {
int arr[100], size;
printf("Enter the number of elements: ");
scanf("%d", &size);
printf("Enter %d elements:\n", size);
for (int i = 0; i < size; i++) {
scanf("%d", &arr[i]);
}
quickSort(arr, 0, size - 1);
printf("Sorted array using Quick Sort:\n");
printArray(arr, size);
return 0;
}
```

```
PS C:\Users\Dev\Projects\Sorting> .\quick_sort
Enter number of elements: 8
Enter 8 elements:
38 27 43 3 9 82 10 15

Unsorted array: 38 27 43 3 9 82 10 15
Sorted array using Quick Sort: 3 9 10 15 27 38 43 82

PS C:\Users\Dev\Projects\Sorting> _```
```

Experiment-4.1

Objective : Implement Binary Search Tree

Procedure :

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
int data;
struct Node *left;
struct Node *right;
};
struct Node* createNode(int value) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data = value;
newNode->left = NULL;
newNode->right = NULL;
return newNode;
}
struct Node* insert(struct Node* root, int value) {
if (root == NULL) {
return createNode(value);
}
if (value < root->data) {
root->left = insert(root->left, value);
} else if (value > root->data) {
root->right = insert(root->right, value);
}
return root;
}
void inorder(struct Node* root) {
if (root == NULL)
return;
inorder(root->left);
printf("%d ", root->data);
```

```
inorder(root->right);
}
int main() {
struct Node* root = NULL;
int n, value, i;
printf("number of nodes : ");
scanf("%d", &n);
printf("Enter %d values:\n", n);
for (i = 0; i < n; i++) {
scanf("%d", &value);
root = insert(root, value);
}
printf("Inorder traversal of the BST:\n");
inorder(root);
printf("\n");
return 0; }
```



Experiment-4.2

Objective : Implement Tree Traversal method using non recursive

Procedure :

```
#include <stdio.h>
```

```
#include <stdlib.h>
struct Node {
int data;
struct Node *left, *right;
};
struct Node* createNode(int data) {
struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
newNode->data = data;
newNode->left = newNode->right = NULL;
return newNode;
}
void inorder(struct Node* root) {
struct Node* stack[100];
int top = -1;
struct Node* current = root;
while (current != NULL || top != -1) {
while (current != NULL) {
stack[++top] = current;
current = current->left;
}
current = stack[top--];
printf("%d ", current->data);
current = current->right;
}
}
void preorder(struct Node* root) {
if (root == NULL) return;
struct Node* stack[100];
int top = -1;
stack[++top] = root;
while (top != -1) {
struct Node* node = stack[top--];
printf("%d ", node->data);
if (node->right != NULL)
stack[++top] = node->right;
```

```
if (node->left != NULL)
stack[++top] = node->left;
}
}
void postorder(struct Node* root) {
if (root == NULL) return;
struct Node* stack1[100], *stack2[100];
int top1 = -1, top2 = -1;
stack1[++top1] = root;
while (top1 != -1) {
struct Node* node = stack1[top1--];
stack2[++top2] = node;
if (node->left != NULL)
stack1[++top1] = node->left;
if (node->right != NULL)
stack1[++top1] = node->right;
}
while (top2 != -1) {
printf("%d ", stack2[top2--]->data);
}
}
```

```
PS C:\Users\Dev\Projects\Trees> .\nonrecursive_traversal

Non-recursive Tree Traversal in C (using Stack)
-----------------------------------------
Creating sample tree with nodes: 50, 30, 70, 20, 40, 60, 80

Inorder traversal (non-recursive):   20 30 40 50 60 70 80
Preorder traversal (non-recursive):  50 30 20 40 70 60 80
Postorder traversal (non-recursive): 20 40 30 60 80 70 50
```

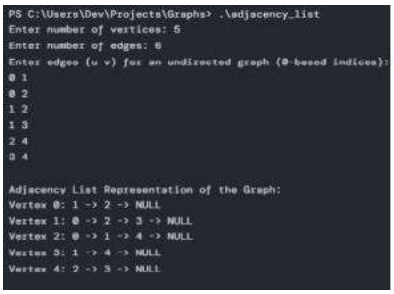
Experiment-5.1

Objective : Implement Adjacency List

Procedure :

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
int vertex;
struct Node* next;
};
struct Node* createNode(int v) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->vertex = v;
newNode->next = NULL;
return newNode;
}
void addEdge(struct Node* adj[], int src, int dest) {
struct Node* newNode = createNode(dest);
newNode->next = adj[src];
adj[src] = newNode;
newNode = createNode(src);
newNode->next = adj[dest];
adj[dest] = newNode;
}
void printGraph(struct Node* adj[], int V) {
printf("\nAdjacency List Representation of the Graph:\n");
for (int i = 0; i < V; i++) {
struct Node* temp = adj[i];
printf("Vertex %d:", i);
while (temp) {
printf(" %d ->", temp->vertex);
temp = temp->next;
}
printf(" NULL\n");
```

```
}
}
int main() {
int V, E, u, v;
printf("Enter number of vertices: ");
scanf("%d", &V);
printf("Enter number of edges: ");
scanf("%d", &E);
struct Node* adj[V];
for (int i = 0; i < V; i++) {
adj[i] = NULL;
}
printf("Enter edges (u v) for an undirected graph (0-based indices):\n");
for (int i = 0; i < E; i++) {
scanf("%d %d", &u, &v);
addEdge(adj, u, v);
}
printGraph(adj, V);
return 0;
}
```



Experiment-5.2

Objective : Implement Adjacency Matrix

Procedure :

```
#include <stdio.h>
```

```
int main() {
int V, E, u, v;
printf("Enter number of vertices: ");
scanf("%d", &V);
int adj[V][V];
for (int i = 0; i < V; i++) {
for (int j = 0; j < V; j++) {
adj[i][j] = 0;
}
}
printf("Enter number of edges: ");
scanf("%d", &E);
printf("Enter edges (u v) for an undirected graph (0-based indices):\n");
for (int i = 0; i < E; i++) {
scanf("%d %d", &u, &v);
adj[u][v] = 1;
adj[v][u] = 1;
}
printf("\nAdjacency Matrix Representation of the Graph:\n ");
for (int i = 0; i < V; i++) {
printf("%d ", i);
}
printf("\n");
for (int i = 0; i < V; i++) {
printf("%d : ", i);
for (int j = 0; j < V; j++) {
printf("%d ", adj[i][j]);
}
printf("\n");
}
return 0;
}
```

```
PS C:\Users\Dev\Projects\Graphs> .\adjacency_matrix
Enter number of vertices: 5
Enter number of edges: 6
Enter edges (u v) for an undirected graph (0-based indices):
0 1
0 2
1 2
1 3
2 4
3 4

Adjacency Matrix Representation of the Graph:
    0 1 2 3 4
0 : 0 1 1 0 0
1 : 1 0 1 1 0
2 : 1 1 0 0 1
3 : 0 0 0 0 1
4 : 0 0 1 1 0
```

Text books and References:

1. Behrouz Forouzan, “Data Communication and Networking”, McGraw Hill
2. Andrew Tanenbaum “Computer Networks”, Prentice Hall.
3. William Stallings, “Data and Computer Communication”, Pearson.
4. Kurose and Ross, “Computer Networking- A Top-Down Approach”, Pearson.
5. Peterson and Davie, “Computer Networks: A Systems Approach”, Morgan Kaufmann
6. W. A. Shay, “Understanding Communications and Networks”, Cengage Learning.
7. D. Comer, “Computer Networks and Internets”, Pearson.
8. Behrouz Forouzan, “TCP/IP Protocol Suite”, McGraw Hill.