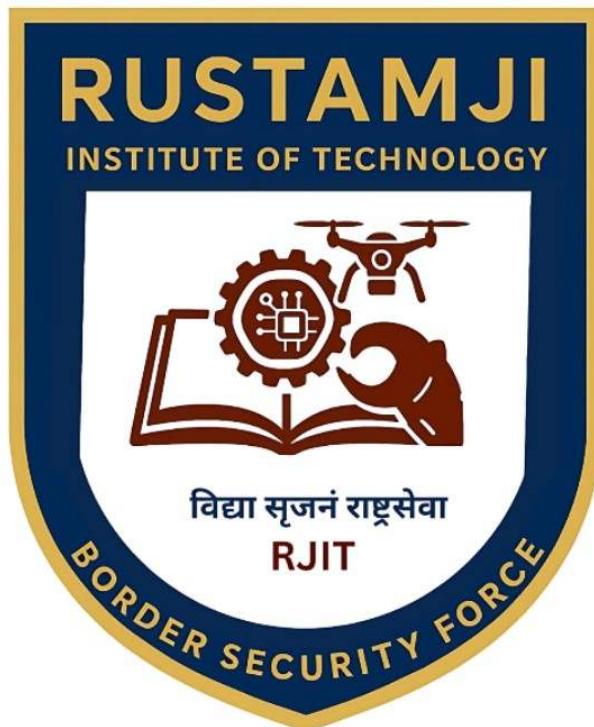


RUSTAMJI INSTITUTE OF TECHNOLOGY

BSF ACADEMY, TEKANPUR

**Lab File for
CS303 (Data Structure)**



Submitted by
Harsh Bharadwaj (0902CS241072)
B.Tech. Computer Science & Engineering 3rd
Semester
(2024-2028 batch)

Subject Teacher
Dr. Jagdish Makhijani

File Checked by
Mr. Yashwant Pathak

TABLE OF CONTENTS

Section-A (Linked List)

S. No.	Practical Description	Page Nos.	COs
1	Implementation of Linked List using array.	1 to 3	CO- 1
2	Implementation of Linked List using Pointers.	4 to 6	CO- 1
3	Implementation of Doubly Linked List using Pointers.	7 to 10	CO - 1
4	Implementation of Circular Single Linked List using Pointers.	11 to 14	CO- 1
5	Implementation of Circular Doubly Linked List using Pointers.	15 to 18	CO- 1

Section-B (Stack)

S. No.	Practical Description	Page Nos.	COs
1	Implementation of Stack using Array.	19 - 23	CO - 2
2	Implementation of Stack using Pointers.	24 - 27	CO - 2
3	Program for Tower of Hanoi using recursion.	28 - 29	C0 2
4	Program to find out factorial of given number using recursion. Also show the various states of stack using in this program.	30 - 33	C0 - 2

Section-C (Queue)

S. No.	Practical Description	Page Nos.	COs
1	Implementation of Queue using Array.	34 - 37	CO - 3
2	Implementation of Queue using Pointers.	38 - 45	CO - 3
3	Implementation of Circular Queue using Array.	46 - 52	CO - 3

Section-D (Trees)

S. No.	Practical Description	Page Nos.	COs
1	Implementation of Binary Search Tree.	53 - 60	CO- 4
2	Conversion of BST PreOrder/PostOrder/InOrder.	61 - 65	CO - 4

Section-E (Sorting & Searching)

S. No.	Practical Description	Page Nos.	COs
1	Implementation of Sorting a. Bubble b. Selection c. Insertion d. Quick e. Merge	66 - 72	CO - 5
2	Implementation of Binary Search on a list of numbers stored in an Array	73 - 74	CO - 5
3	Implementation of Binary Search on a list of strings stored in an Array	75 - 76	CO - 5
4	Implementation of Linear Search on a list of strings stored in an Array	77 - 78	CO - 5
5	Implementation of Binary Search on a list of strings stored in a Single Linked List (optional)	79 - 81	CO - 5

Section-A (Linked List)

Experiment No :1

Program Description:

Implementation of Linked List using array.

```
#include<stdio.h>
#define MAXSIZE 100
```

```
typedef struct {
    int data[MAXSIZE];
    int next[MAXSIZE];
    int head;
    int freeIndex;
} ArrayLinkedList;
```

```
void initList(ArrayLinkedList *list) {
    list->head = -1;
    list->freeIndex = 0;
    for(int i = 0; i < MAXSIZE; i++) {
        list->next[i] = -1;
    }
}
```

```
void insert(ArrayLinkedList *list, int value) {
    if(list->freeIndex >= MAXSIZE) {
        printf("List is full\n");
        return;
    }
    int newIndex = list->freeIndex;
    list->data[newIndex] = value;
    list->next[newIndex] = -1;

    if(list->head == -1) {
        list->head = newIndex;
    } else {
        int current = list->head;
        while(list->next[current] != -1) {
            current = list->next[current];
        }
        list->next[current] = newIndex;
    }
    list->freeIndex++;
}
```

```
void deleteValue(ArrayLinkedList *list, int value) {
    int current = list->head;
```

```

{
int previous = -1;
while(current != -1) {
if(list->data[current] == value) {
if(previous == -1) {
list->head = list->next[current];
} else {
list->next[previous] = list->next[current];
}
list->next[current] = -1;
printf("Value %d deleted from the list.\n", value);
return;
}
previous = current;
current = list->next[current];
}
printf("Value %d not found in the list.\n", value);
}

void display(ArrayLinkedList *list) {
int current = list->head;
while(current != -1) {
printf("%d -> ", list->data[current]);
current = list->next[current];
}
printf("NULL\n");
}

int search(ArrayLinkedList *list, int value) {
int current = list->head;
int index = 0;
while(current != -1) {
if(list->data[current] == value) {
return index;
}
current = list->next[current];
index++;
}
return -1;
}

int size(ArrayLinkedList *list) {
int count = 0;
int current = list->head;
while(current != -1) {
count++;
current = list->next[current];
}
return count;
}

```

```

int main() {
    ArrayLinkedList list;
    initList(&list);

    insert(&list, 10);
    insert(&list, 20);
    insert(&list, 30);
    display(&list);

    deleteValue(&list, 20);
    display(&list);

    insert(&list, 40);
    display(&list);

    int value = 30;
    int index = search(&list, value);
    if(index != -1) {
        printf("Value %d found at index %d\n", value, index);
    } else {
        printf("Value %d not found in the list\n", value);
    }

    printf("Size of the list: %d\n", size(&list));
    return 0;
}

```

Output:

```

Run          Output      Clear
^ 10 -> 20 -> 30 -> NULL
Value 20 deleted from the list.
10 -> 30 -> NULL
10 -> 30 -> 40 -> NULL
Value 30 found at index 1
Size of the list: 3

==== Code Execution Successful ====

```

Experiment No: 2

Program Description :

Implementation of Linked List using Pointers.

Solution:

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

typedef struct Node {
    int data;
    struct Node* next;
} Node;

typedef struct {
    Node* head;
} LinkedList;

void initList(LinkedList *list) {
    list->head = NULL;
}

Node* createNode(int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = value;
    newNode->next = NULL;
    return newNode;
}

void insert(LinkedList *list, int value) {
    Node* newNode = createNode(value);
    if(list->head == NULL) {
        list->head = newNode;
    } else {
        Node* current = list->head;
        while(current->next != NULL) {
            current = current->next;
        }
        current->next = newNode;
    }
}

void deleteValue(LinkedList *list, int value) {
    Node* current = list->head;
```

```

Node* previous = NULL;
while(current != NULL) {
if(current->data == value) {
if(previous == NULL) {
list->head = current->next;
} else {
previous->next = current->next;
}
free(current);
printf("Value %d deleted from the list.\n", value);
return;
}
previous = current;
current = current->next;
}
printf("Value %d not found in the list.\n", value);
}

void display(LinkedList *list) {
Node* current = list->head;
while(current != NULL) {
printf("%d -> ", current->data);
current = current->next;
}
printf("NULL\n");
}

Node* search(LinkedList *list, int value) {
Node* current = list->head;
while(current != NULL) {
if(current->data == value) {
return current;
}
current = current->next;
}
return NULL;
}

int size(LinkedList *list) {
int count = 0;
Node* current = list->head;
while(current != NULL) {
count++;
current = current->next;
}
return count;
}

int main() {
LinkedList list;

```

```
initList(&list);

printf("List after insertion\n");
insert(&list, 10);
insert(&list, 20);
insert(&list, 30);
display(&list);

printf("List after deletion\n");
deleteValue(&list, 20);
display(&list);

int searchValue = 30;
Node* foundNode = search(&list, searchValue);
if(foundNode) {
    printf("Value %d found in the list.\n", searchValue);
} else {
    printf("Value %d not found in the list.\n", searchValue);
}

printf("Size of the list: %d\n", size(&list));
return 0;
}
```

Output:

Run Output Clear

```
^ List after insertion
10 -> 20 -> 30 -> NULL
List after deletion
Value 20 deleted from the list.
10 -> 30 -> NULL
Value 30 found in the list.
Size of the list: 2      I

==== Code Execution Successful ===
```

Experiment No : 3

Program Description:

Implementation of Doubly Linked List using Pointers.

Solution:

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

typedef struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
} Node;

typedef struct {
    Node* head;
} DoublyLinkedList;

void initList(DoublyLinkedList *list) {
    list->head = NULL;
}

Node* createNode(int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = value;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}

void insert(DoublyLinkedList *list, int value) {
    Node* newNode = createNode(value);
    if(list->head == NULL) {
        list->head = newNode;
    } else {
        Node* current = list->head;
        while(current->next != NULL) {
            current = current->next;
        }
        current->next = newNode;
        newNode->prev = current;
    }
}

void deleteValue(DoublyLinkedList *list, int value) {
```

```

}

Node* current = list->head;
while(current != NULL) {
if(current->data == value) {
if(current->prev != NULL) {
current->prev->next = current->next;
} else {
list->head = current->next;
}
if(current->next != NULL) {
current->next->prev = current->prev;
}
free(current);
printf("Deleted value %d\n", value);
return;
}
current = current->next;
}
printf("Value %d not found in the list.\n", value);
}

void displayForward(DoublyLinkedList *list) {
Node* current = list->head;
while(current != NULL) {
printf("%d -> ", current->data);
current = current->next;
}
printf("NULL\n");
}

void displayBackward(DoublyLinkedList *list) {
if(list->head == NULL) {
printf("NULL\n");
return;
}
Node* current = list->head;
while(current->next != NULL) {
current = current->next;
}
while(current != NULL) {
printf("%d -> ", current->data);
current = current->prev;
}
printf("NULL\n");
}

Node* search(DoublyLinkedList *list, int value) {
Node* current = list->head;
while(current != NULL) {
if(current->data == value) {
return current;
}
}

```

```

current = current->next;
}
return NULL;
}
int size(DoublyLinkedList *list) {
int count = 0;
Node* current = list->head;
while(current != NULL) {
count++;
current = current->next;
}
return count;
}
int main() {
DoublyLinkedList list;
initList(&list);

printf("Forward\n");
insert(&list, 10);
insert(&list, 20);
insert(&list, 30);
displayForward(&list);

printf("Backward\n");
displayBackward(&list);

printf("After deletion\n");
deleteValue(&list, 20);
printf("Forward\n");
displayForward(&list);
printf("Backward\n");
displayBackward(&list);

insert(&list, 40);
insert(&list, 50);
printf("After insertion\n");
printf("Forward\n");
displayForward(&list);
printf("Backward\n");
displayBackward(&list);

Node* found = search(&list, 30);
if(found) {
printf("Value 30 found in the list.\n");
} else {
printf("Value not found.\n");
}
}

```

```
printf("Size of the list: %d\n", size(&list));  
return 0;
```

Output :

```
Run Output Clear  
Forward  
10 -> 20 -> 30 -> NULL  
Backward  
30 -> 20 -> 10 -> NULL  
After deletion  
Deleted value 20  
Forward  
10 -> 30 -> NULL  
Backward  
30 -> 10 -> NULL  
After insertion  
Forward  
10 -> 30 -> 40 -> 50 -> NULL  
Backward  
50 -> 40 -> 30 -> 10 -> NULL  
Value 30 found in the list.  
Size of the list: 4  
  
==== Code Execution Successful ===
```

Experiment No : 4

Program Description:

Implementation of Circular Single Linked List using Pointers.

Solution:

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

typedef struct Node {
    int data;
    struct Node* next;
} Node;

typedef struct {
    Node* last;
} CircularLinkedList;

void initList(CircularLinkedList *list) {
    list->last = NULL;
}

Node* createNode(int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = value;
    newNode->next = NULL;
    return newNode;
}

void insertAtEnd(CircularLinkedList *list, int value) {
    Node* newNode = createNode(value);
    if(list->last == NULL) {
        newNode->next = newNode;
        list->last = newNode;
    } else {
        newNode->next = list->last->next;
        list->last->next = newNode;
        list->last = newNode;
    }
}

void insertAtBeginning(CircularLinkedList *list, int value) {
    Node* newNode = createNode(value);
    if(list->last == NULL) {
        newNode->next = newNode;
        list->last = newNode;
    } else {
```

```

newNode->next = list->last->next;
list->last->next = newNode;
}
}

void deleteValue(CircularLinkedList *list, int value) {
if(list->last == NULL) {
printf("List is empty.\n");
return;
}
Node* current = list->last->next;
Node* previous = list->last;

do {
if(current->data == value) {
if(current == list->last->next) {
if(list->last->next == list->last) {
list->last = NULL;
} else {
list->last->next = current->next;
}
} else {
previous->next = current->next;
}
free(current);
printf("Deleted value %d\n", value);
return;
}
previous = current;
current = current->next;
} while(current != list->last->next);

printf("Value %d not found in the list.\n", value);
}

void display(CircularLinkedList *list) {
if(list->last == NULL) {
printf("List is empty.\n");
return;
}
Node* current = list->last->next;
do {
printf("%d -> ", current->data);
current = current->next;
} while(current != list->last->next);
printf("Back to Start\n");
}

Node* search(CircularLinkedList *list, int value) {
if(list->last == NULL) return NULL;
Node* current = list->last->next;

```

```

do {
    if(current->data == value) return current;
    current = current->next;
} while(current != list->last->next);
return NULL;
}
int size(CircularLinkedList *list) {
    if(list->last == NULL) return 0;
    int count = 0;
    Node* current = list->last->next;
    do {
        count++;
        current = current->next;
    } while(current != list->last->next);
    return count;
}
int main() {
    CircularLinkedList list;
    initList(&list);

    printf("Circular Linked List\n");
    insertAtEnd(&list, 10);
    insertAtEnd(&list, 20);
    insertAtEnd(&list, 30);
    display(&list);

    printf("After inserting at the beginning\n");
    insertAtBeginning(&list, 5);
    display(&list);

    printf("After deleting 20\n");
    deleteValue(&list, 20);
    display(&list);

    Node* found = search(&list, 30);
    if(found) {
        printf("Value 30 found in the list.\n");
    } else {
        printf("Value 30 not found.\n");
    }

    printf("Size of the list: %d\n", size(&list));
    return 0;
}

```

Output :

Run	Output	Clear
	<pre>* Circular Linked List 10 -> 20 -> 30 -> Back to Start After inserting at the beginning 5 -> 10 -> 20 -> 30 -> Back to Start After deleting 20 Deleted value 20 5 -> 10 -> 30 -> Back to Start Value 30 found in the list. Size of the list: 3</pre> <p>==== Code Execution Successful ===</p>	

Experiment No : 5

Program Description:

Implementation of Circular Doubly Linked List using Pointers.

Solution:

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

typedef struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
} Node;

typedef struct {
    Node* last;
} CircularDoublyLinkedList;

void initList(CircularDoublyLinkedList *list) {
    list->last = NULL;
}

Node* createNode(int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = value;
    newNode->next = newNode;
    newNode->prev = newNode;
    return newNode;
}

void insertAtEnd(CircularDoublyLinkedList *list, int value) {
    Node* newNode = createNode(value);
    if(list->last == NULL) {
        list->last = newNode;
    } else {
        Node* first = list->last->next;
        newNode->next = first;
        newNode->prev = list->last;
        list->last->next = newNode;
        first->prev = newNode;
        list->last = newNode;
    }
}

void insertAtBeginning(CircularDoublyLinkedList *list, int value) {
    Node* newNode = createNode(value);
```

```

if(list->last == NULL) {
list->last = newNode;
} else {
Node* first = list->last->next;
newNode->next = first;
newNode->prev = list->last;
list->last->next = newNode;
first->prev = newNode;
}
}

void deleteValue(CircularDoublyLinkedList *list, int value) {
if(list->last == NULL) {
printf("List is empty.\n");
return;
}
Node* current = list->last->next;
do {
if(current->data == value) {
if(current == list->last->next) {
if(list->last->next == list->last) {
list->last = NULL;
} else {
list->last->next = current->next;
current->next->prev = list->last;
}
} else if(current == list->last) {
list->last = current->prev;
current->prev->next = list->last->next;
list->last->next->prev = list->last;
} else {
current->prev->next = current->next;
current->next->prev = current->prev;
}
free(current);
printf("Deleted value %d\n", value);
return;
}
current = current->next;
} while(current != list->last->next);

printf("Value %d not found in the list.\n", value);
}

void display(CircularDoublyLinkedList *list) {
if(list->last == NULL) {
printf("List is empty.\n");
return;
}
}

```

```

Node* current = list->last->next;
do {
printf("%d -> ", current->data);
current = current->next;
} while(current != list->last->next);
printf("Back to Start\n");
}

Node* search(CircularDoublyLinkedList *list, int value) {
if(list->last == NULL) return NULL;
Node* current = list->last->next;
do {
if(current->data == value) return current;
current = current->next;
} while(current != list->last->next);
return NULL;
}
int size(CircularDoublyLinkedList *list) {
if(list->last == NULL) return 0;
int count = 0;
Node* current = list->last->next;
do {
count++;
current = current->next;
} while(current != list->last->next);
return count;
}
int main() {
CircularDoublyLinkedList list;
initList(&list);

printf("Circular Doubly Linked List\n");
insertAtEnd(&list, 10);
insertAtEnd(&list, 20);
insertAtEnd(&list, 30);
display(&list);

printf("After inserting at the beginning\n");
insertAtBeginning(&list, 5);
display(&list);

printf("After deleting 20\n");
deleteValue(&list, 20);
display(&list);

Node* found = search(&list, 30);
if(found) {

```

Section-B (Stack)

Experiment No: 1

Program Description:

Implementation of Stack using Array.

Solution:

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

```
#include<stdlib.h>
```

```
#define MAX 100
```

```
typedef struct Stack{
```

```
    int arr[MAX];
```

```
    int top;
```

```
}Stack;
```

```
void initStack(Stack* stack){
```

```
    stack->top =-1;
```

```
}
```

```
int isEmpty(Stack* stack){
```

```
    return stack->top ==-1;
```

```
}
```

```
int isFull(Stack* stack){
```

```
    return stack->top == MAX-1;
```

```
}
```

```
voidpush(Stack* stack, int item){
```

```
    if(isFull(stack)){
```

```
        printf("Stack Overflow! Cannot push %d\n", item);
```

```
        return;
```

```
}
```

```
    stack->arr[++stack->top] = item;
```

```
    printf("%d pushed to stack\n", item);
```

```
}
```

```
intpop(Stack* stack){
```

```
    ifisEmpty(stack)){
```

```
        printf("Stack Underflow! Cannot pop from an empty stack\n");
```

```
        return -1;
```

```
}
```

```
    return stack->arr[stack->top];
```

```
}
```

```
intpeek(Stack* stack){
```

```
    ifisEmpty(stack)){
```

```
printf("Stack is empty! Cannot peek\n");
```

```
return -1;
```

```
}
```

```
return stack->arr[stack->top];
```

```
}
```

```
void display(Stack* stack){
```

```
if(isEmpty(stack)){
```

```
printf("Stack is empty!\n");
```

```
return;
```

```
}
```

```
for(int i = 0; i<=stack->top; i++){
```

```
printf("%d", stack->arr[i]);
```

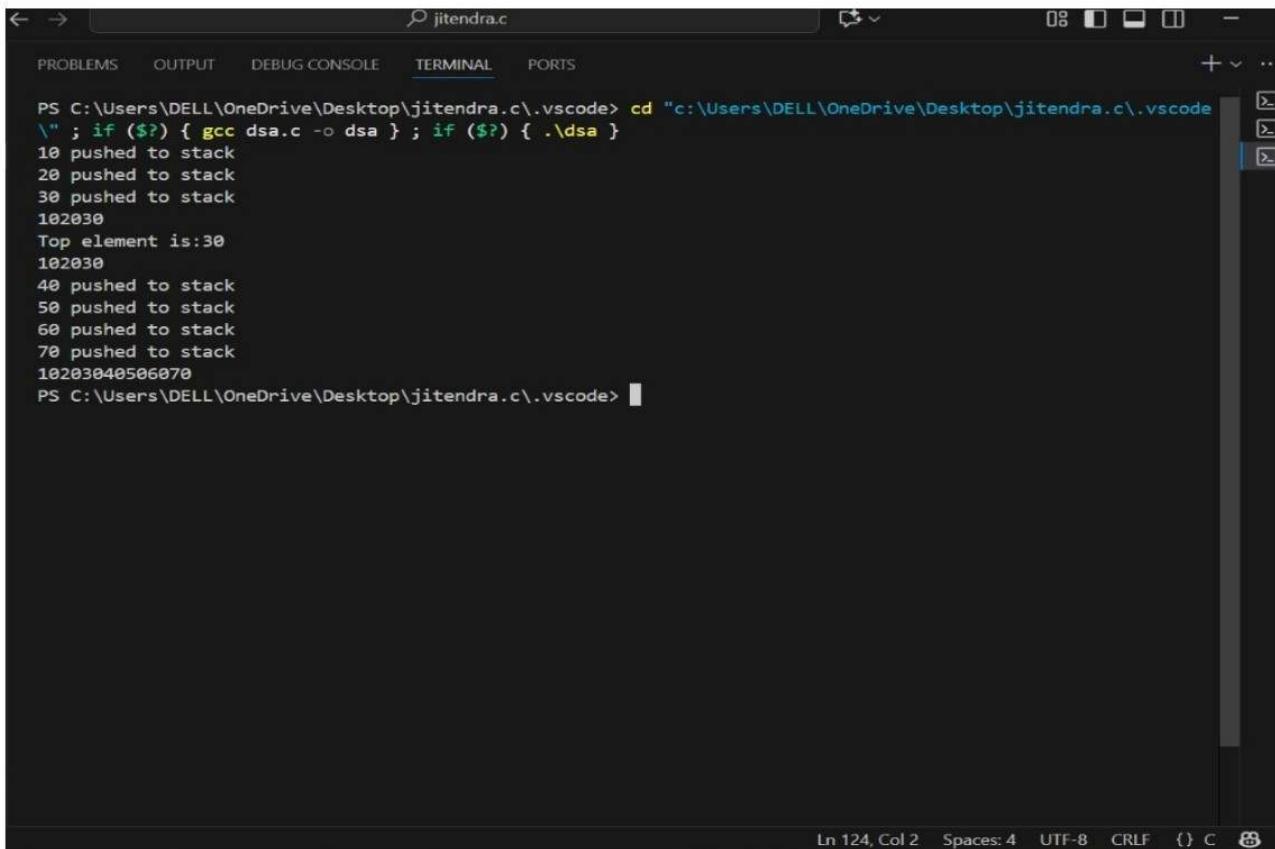
```
}
```

```
printf("\n");
```

```
}
```

```
intmain(){  
  
    Stack stack;  
  
    initStack(&stack);  
  
    push(&stack, 10);  
  
    push(&stack, 20);  
  
    push(&stack, 30);  
  
    display(&stack);  
  
    printf("Top element is:%d\n",peek(&stack));  
  
    display(&stack);  
  
    push(&stack,40);  
  
    push(&stack,50);  
  
    push(&stack,60);  
  
    push(&stack,70);  
  
    display(&stack);  
  
    return 0;  
}
```

OUTPUT:



```
PS C:\Users\DELL\OneDrive\Desktop\jitendra.c\.vscode> cd "c:\Users\DELL\OneDrive\Desktop\jitendra.c\.vscode"
\" ; if ($?) { gcc dsa.c -o dsa } ; if (?) { ./dsa }
10 pushed to stack
20 pushed to stack
30 pushed to stack
102030
Top element is:30
102030
40 pushed to stack
50 pushed to stack
60 pushed to stack
70 pushed to stack
10203040506070
PS C:\Users\DELL\OneDrive\Desktop\jitendra.c\.vscode>
```

Ln 124, Col 2 Spaces: 4 UTF-8 CRLF {} C

EXPERIMENT No : 2

Program Description:

Implementation of Stack using Pointers.

Solution:

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

```
#include<stdlib.h>
```

```
typedef struct Node {
```

```
    int data;
```

```
    struct Node* next;
```

```
}Node;
```

```
typedef struct Stack{
```

```
    Node* top;
```

```
}Stack;
```

```
Stack* createStack(){
```

```
    Stack* stack = (Stack*)malloc(sizeof(Stack));
```

```
    stack->top = NULL;
```

```
    return stack;
```

```
}
```

```
int isEmpty(Stack* stack){
```

```
    return stack->top == NULL;
```

```
}
```

```
void push(Stack* stack, int item){
```

```

Node* newNode = (Node*)malloc(sizeof(Node));
newNode->data = item;
newNode->next = stack->top;
stack->top = newNode;
printf("%d pushed to stack\n", item);

}

intpop(Stack* stack){
if(isEmpty(stack)){
    printf("Stack Underflow! Cannot pop from an empty stack\n");
    return -1;
}
Node* temp = stack->top;
intpopped = temp->data;
stack->top = stack->top->next;
free(temp);
return popped;
}

intpeek(Stack* stack){
if(isEmpty(stack)){
    printf("Stack is empty! Cannot peek\n");
    return -1;
}
return stack->top->data;
}

```

```
void display(Stack* stack){  
    if(isEmpty(stack)){  
        printf("Stack is empty!\n");  
        return;  
    }  
  
    Node* current = stack->top;  
    printf("Stack elements:");  
    while (current!= NULL){  
        printf("%d", current->data);  
        current = current->next;  
    }  
    printf("\n");  
}  
  
int main(){  
    Stack* stack = createStack();  
    push(stack, 10);  
    push(stack, 20);  
    push(stack, 30);  
    display(stack);  
    printf("Top element is:%d\n", peek (stack));  
    printf("Popped element is:%d\n",pop(stack));  
    display(stack);  
    push(stack, 40);  
    push(stack, 50);  
    display(stack);  
    printf("Popped element is:%d\n", pop(stack));  
    display(stack);  
    return 0;  
}
```

OUTPUT:

A screenshot of the Visual Studio Code interface, specifically the Terminal tab. The terminal window has a dark background and displays the output of a C program. The code in the terminal window is as follows:

```
PS C:\Users\DELL\OneDrive\Desktop\jitendra.c\.vscode> cd "c:\Users\DELL\OneDrive\Desktop\jitendra.c\.vscode"
\" ; if ($?) { gcc dsa2.c -o dsa2 } ; if ($?) { ./dsa2 }

10 pushed to stack
20 pushed to stack
30 pushed to stack
Stack elements:302010
Top element is:30
Popped element is:30
Stack elements:2010
40 pushed to stack
50 pushed to stack
Stack elements:50402010
Popped element is:50
Stack elements:402010
PS C:\Users\DELL\OneDrive\Desktop\jitendra.c\.vscode>
```

The terminal shows the execution of a script that compiles a C program named `dsa2.c` and runs it. The program performs stack operations, pushing values 10, 20, and 30 onto the stack, then popping them off. The stack elements are printed as pairs of values separated by colons, such as `302010`. The terminal window also includes status indicators at the bottom like "Ln 28, Col 30" and file encoding settings.

EXPERIMENT No : 3

Program Description:

Program for Tower of Hanoi using recursion.

Solution:

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

```
voidtowerOfHanoi(int n, char source, char destination, char auxiliary){

    if(n==1){

        printf("Move disk 1 from %c to %c\n", source, destination);

        return;

    }

    towerOfHanoi(n-1, source, auxiliary, destination);

    printf("Moves disk %d from %c to %c\n",n, source,destination);

    towerOfHanoi(n-1, auxiliary, destination, source);

}

intmain(){

    intn;

    printf("Enter the number of disks:");

    scanf("%d",&n);

    printf("The sequence of moves for Tower of Hanoi with %d disks is:\n",n);

    towerOfHanoi(n,'A','c','B');

    return 0;

}
```

OUTPUT:

The screenshot shows a terminal window in VS Code. The terminal tab is selected at the top. The command entered was "PS C:\Users\DELL\OneDrive\Desktop\jitendra.c\.vscode> cd "c:\Users\DELL\OneDrive\Desktop\jitendra.c\.vscode\n"; if (\$?) { gcc dsa3.c -o dsa3 } ; if (\$?) { .\dsa3 }". The output shows the program asking for the number of disks (4) and then listing the sequence of moves for the Tower of Hanoi problem with 4 disks, starting from disk 1 on A to disk 4 on C.

```
PS C:\Users\DELL\OneDrive\Desktop\jitendra.c\.vscode> cd "c:\Users\DELL\OneDrive\Desktop\jitendra.c\.vscode\n"; if ($?) { gcc dsa3.c -o dsa3 } ; if ($?) { .\dsa3 }
Enter the number of disks:4
The sequence of moves for Tower of Hanoi with 4 disks is:
Move disk 1 from A to B
Moves disk 2 from A to c
Move disk 1 from B to c
Moves disk 3 from A to B
Move disk 1 from c to A
Moves disk 2 from c to B
Move disk 1 from A to B
Moves disk 4 from A to c
Move disk 1 from B to c
Moves disk 2 from B to A
Move disk 1 from c to A
Moves disk 3 from B to c
Move disk 1 from A to B
Moves disk 2 from A to c
Move disk 1 from B to c
PS C:\Users\DELL\OneDrive\Desktop\jitendra.c\.vscode>
```

EXPERIMENT No : 4

Program Description:

Program to find out factorial of given number using recursion. Also show the various states of stack using in this program.

Solution:

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

typedef struct Stack {

    char stackState[100][100];
    int top;

}Stack;

void initStack(Stack* stack){

    stack->top = -1;

}

voidpush(Stack* stack, const char* state){

    if(stack->top<99){

        strcpy(stack->stackState[++stack->top],state);

    }

}

voidpop(Stack* stack){

    if(stack->top>-1){

        stack->top--;

    }

}
```

```

    }

}

void displayStack(Stack* stack){
    if(stack->top == -1){
        printf("Stack is empty\n");
        return;
    }

    printf("\n---Current Stack State---\n");
    for(int i = stack->top;i>=0;i--){
        printf("%s\n",stack->stackState[i]);
    }

    printf("-----\n");
}

int factorial(int n, Stack* stack){
    char state[100];
    sprintf(state,"factorial(%d)-Entering",n);
    push(stack, state);
    displayStack(stack);

    if(n==0 || n==1){
        sprintf(state,"Returning 1 from factorial(%d)",n);
        push(stack, state);
        displayStack(stack);
    }
}

```

```
pop(stack);
return 1;
}

intresult = n * factorial(n-1,stack);
sprintf(state,"returning %d from factorial(%d)", result,n);
push(stack, state);
displayStack(stack);
pop(stack);
return result;
```

```
}
```

```
intmain(){
intnum; Stack
stack;

initStack(&stack);

printf("Enterb a number:");
scanf("%d",&num);

if(num<0){

printf("Factorial is not defined for negative numbers.\n");
return 0;
}

printf("\nCalculating Factorial of %d with Stack Simulation:\n",num);
intresult = factorial(num,&stack);
printf("\nFactorial of %d is\n", num, result);
```

```
return 0;  
}
```

Output:

The screenshot shows a terminal window in VS Code with the following output:

```
PS C:\Users\DELL\OneDrive\Desktop\jitendra.c\vscode> cd "c:\Users\DELL\OneDrive\Desktop\jitendra.c\vscode"  
\" ; if ($?) { gcc dsa4.c -o dsa4 } ; if ($?) { ./dsa4 }  
Enter a number:3  
Calculating Factorial of 3 with Stack Simulation:  
---Current Stack State---  
factorial(3)-Entering  
-----  
---Current Stack State---  
factorial(2)-Entering  
factorial(3)-Entering  
-----  
---Current Stack State---  
factorial(1)-Entering  
factorial(2)-Entering  
factorial(3)-Entering  
-----  
---Current Stack State---  
Returning 1 from factorial(1)  
factorial(1)-Entering  
factorial(2)-Entering  
factorial(3)-Entering  
-----  
---Current Stack State---  
returning 2 from factorial(2)  
factorial(1)-Entering  
factorial(2)-Entering  
factorial(3)-Entering
```

The terminal interface includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is selected), and PORTS. The status bar at the bottom shows "Ln 74, Col 1" and other settings like "Spaces: 4", "UTF-8", "CRLF", and "Win3".

Section-C (QUEUE)

ExperimentNo : 1

Program Description:

Implementation of queue using an array.

Solution:

```
#include <stdio.h>

#define MAX 5 int
queue[MAX]; int
front=-1; int
rear=-1; void
enqueue(int x) {

    if((rear+1)%MAX==front)
    {
        printf("Overflow condition\n");
    }
    else if(front==-1)
    {
        front=rear=0;
    }
    else
    {
        rear=(rear+1)%MAX;
        queue[rear]=x;
    }
}

void dequeue()
{
```

```

}

}

if(front==-1)
{
    printf("Underflow condition\n");
}
else
{
    printf("Deleted: %d\n", queue[front]);
    if(front==rear)
    {
        front=rear=-1;
    }
    else
        front=(front+1)%MAX;
}
}

void display()
{
    if(front==-1)
    {
        printf("Queue is empty\n");
    }
    else
    {

```

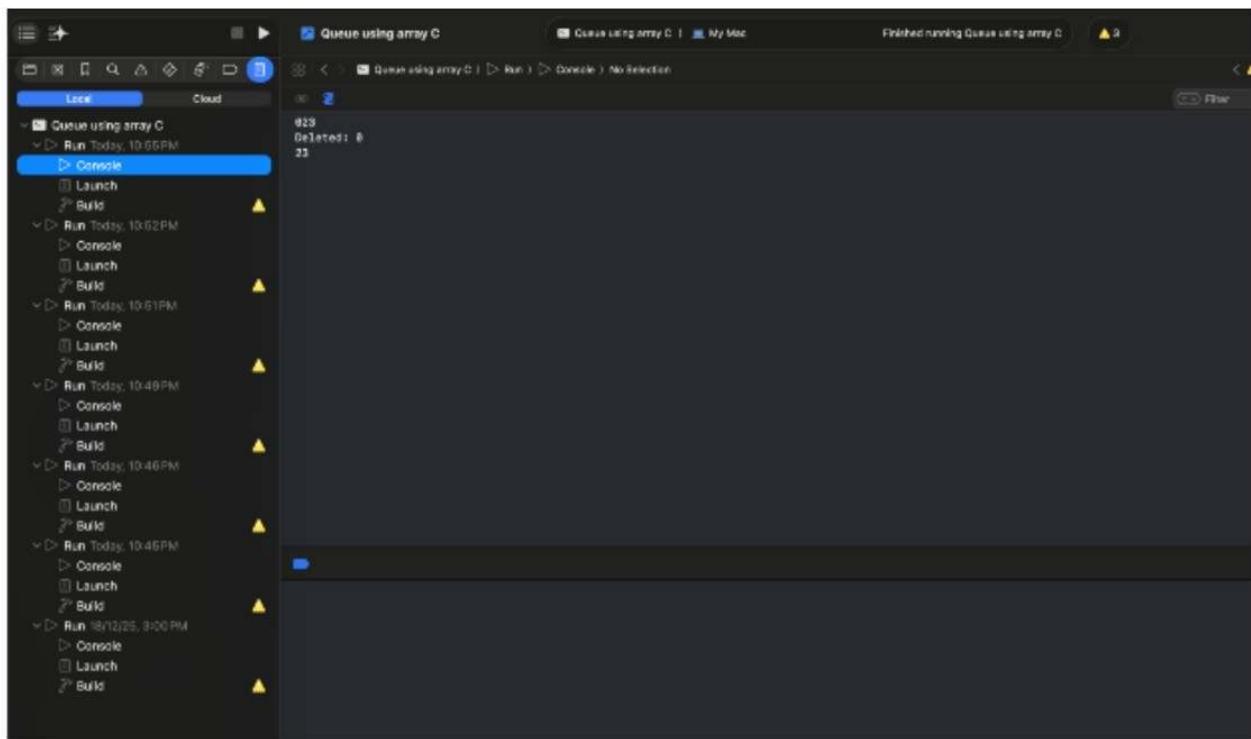
```

inti=front;
while(1)
{
    printf("%d", queue[i]);
    if(i==rear)
        break;
    i=(i+1)%MAX;
}
printf("\n");
}

intmain()
{
    enqueue(1);
    enqueue(2);
    enqueue(3);
    display();
    dequeue();
    display();
    return 0;
}

```

Output:



Experiment No : 2

Program Description:

Implementation of queue using pointers.

Solution:

```
#include <stdio.h>

#include<stdlib.h>

typedef struct Node {
    int data;
    struct Node*next;
}

}Node;

typedef struct Queue{
    Node*front;
    Node*rear;
}

}Queue;

void initializeQueue(Queue*q){
    q->front=NULL;
    q->rear=NULL;
}

intempty(Queue*q){
    return q->front==NULL;
}
```

```

void enqueue(Queue*q, int value){

    Node*newNode=(Node*)malloc(sizeof(Node));

    if(newNode==NULL)
    {
        printf("Memory allocation FAILED. Cannot enqueue%d\n", value);
        return;
    }

    newNode->data=value;
    newNode->next=NULL;

    if(q->rear==NULL)
    {
        q->front=q->rear=newNode;
    }
    else
    {
        q->rear->next=newNode;
        q->rear=newNode;
    }

    printf("Enqueued:%d\n", value);
}

int dequeue(Queue*q){

    if(empty(q))
    {
        printf("Queue underflow. Cannot dequeue\n ");
        return -1;
    }
}

```

```
}
```

```
Node*temp=q->front;
```

```
intvalue=temp->data;
```

```
q->front=q->front->next;
```

```
if(q->front==NULL)
```

```
{
```

```
q->rear=NULL;
```

```
}
```

```
free(temp);
```

```
return value;
```

```
}
```

```
int peek(Queue*q)
```

```
{
```

```
if(empty(q))
```

```
{
```

```
printf("Queue is Empty\n");
```

```
return -1;
```

```
}
```

```
return q->front->data;
```

```
}
```

```
void displayQueue(Queue*q)
```

```
{
```

```
if(empty(q))
```

```
{
```

```

printf("Queue is Empty\n");
return;
}

Node*current=q->front;
printf("Queue elements: ");

while(current!=NULL)
{
    printf("%d", current->data);
    current=current->next;
}
printf("\n");
}

```

int countElements(Queue*q)

```

{
if(empty(q))
{
    return 0;
}

```

intcount=0;

Node*current=q->front;

while(current!=NULL)

```

{
    count++;
    current=current->next;
}

return count;
}

```

```
void clearQueue(Queue*q)
{
    while(!empty(q))
    {
        dequeue(q);
    }
    printf("Queue is cleared successfully.\n");
}
```

```
void menu()
{
    printf("\nQueue Operations:\n");
    printf("1. Enqueue\n");
    printf("2. Dequeue\n");
    printf("3. Peek\n");
    printf("4. Display Queue\n");
    printf("5. Count Elements\n");
    printf("6. Clear\n");
    printf("7. Exit\n");
    printf("Enter your choice: ");
```

```
}
```

```
intmain(){
    Queue q1,q2;
    initializeQueue(&q1);
    initializeQueue(&q2);
    intchoice,value, position;
```

```
while(1)
{
}
```

```
menu();
scanf("%d", &choice);

switch(choice)

{
    case1:
        printf("Enter the value to enqueue: ");
        scanf("%d", &value);
        enqueue(&q1,value);
        break;

    case2:
        value=dequeue(&q1);
        if(value!=-1)
        {
            printf("Dequeue: %d\n", value);
        }
        break;

    case3:
        value=peek(&q1);
        if(value!=-1);
        {
            printf("Front element: %d\n", value);
        }
        break;

    case4:
        displayQueue(&q1);
        break;
}
```

```
case 5:  
    printf("Number of elements in the queue: %d\n ", countElements(&q1));  
    break;  
  
case 6:  
    clearQueue(&q1);  
    break;  
    exit(0);  
  
}case 7:  
    printf("Exit from program\n");  
  
default:  
    printf("Invalid operation\n");  
  
}  
return0;  
}
```

Output:

```
Queue Operations:
1. Enqueue
2. Dequeue
3. Peek
4. Display Queue
5. Count Elements
6. Clear
7. Exit
Enter your choice: 1
Enter the value to enqueue: 10
Queue[10]

Queue Operations:
1. Enqueue
2. Dequeue
3. Peek
4. Display Queue
5. Count Elements
6. Clear
7. Exit
Enter your choice: 5
Number of elements in the queue: 1

Queue Operations:
1. Enqueue
2. Dequeue
3. Peek
4. Display Queue
5. Count Elements
6. Clear
7. Exit
```

Experiment No : 3

Program Description:

Implementation of circular queue using array.

Solution:

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

#define MAX 100

typedef struct CircularQueue{

    int data[MAX];
    int front;  int
    rear;
    int size;
}CircularQueue;

void initializeQueue(CircularQueue*q){

    q->front=-1;
    q->rear=-1;
    q->size=0;
}

int isFull(CircularQueue*q)
```

{

```

return q->size==MAX;
}

int empty(CircularQueue*q){
    return q->size==0;
}

void enqueue(CircularQueue*q, int value)
{
    if(isFull(q))
    {
        printf("Queue Oveerflow. Cannot dequeue\n");
        return;
    }

    if(q->front==-1)
    {
        q->front=0;
        q->rear=(q->rear+1)%MAX;
        q->data[q->rear]=value;
        q->size++;
        printf("Enqueued: %d\n", value);
    }
}

int dequeue(CircularQueue*q)
{
    if(empty(q))
    {
        printf("Queue Underflow. Cannot dequeue\n");
    }
}

```

```

return -1;
}

int value=q->data[q->front];
q->front=(q->front+1)%MAX;
q->size--;
if(q->size==0)
{
    q->front=-1;
    q->rear=-1;
}
return value;
}

int peek(CircularQueue*q)
{
    if(empty(q))
    {
        printf("Queue is Empty\n");
        return -1;
    }
    return q->data[q->front];
}

void dispalyQueue(CircularQueue*q)
{
    if(empty(q))
    {
        printf("Queue is Empty\n");
    }
}

```

```

return;
}

printf("Queue elements: ");
for(int i=0, index=q->front; i<q->size; i++, index=(index+1)%MAX)
{
    printf("%d", q->data[index]);
}
printf("\n");
}

int countElements(CircularQueue*q)
{
    return q->size;
}

void clearQueue(CircularQueue*q)
{
    q->front=-1;
    q->rear=-1;
    q->size=0;
    printf("Queue is cleared successfully\n");
}

void menu()
{
    printf("\nCircular Queue Operations:\n");
    printf("1. ENQUEUE\n");
    printf("2. DEQUEUE\n");
    printf("3. PEEK\n");
}

```

```
printf("4. Display Queue\n");
printf("5. Count Elements\n");
printf("6. Clear Queue\n");
printf("7. EXIT\n");
printf("Enter your choice:");
}
```

```
int main()
{
    CircularQueue q1, q2;
    initializeQueue(&q1);
    initializeQueue(&q2);
    int choice, value, position;
```

```
while(1)
{
    menu();
    scanf("%d", &choice);

    switch(choice)
    {
        case 1:
            printf("Enter value to Enqueue: ");
            scanf("%d", &value);
            enqueue(&q1, value);
            break;

        case 2:
```

```
        value=dequeue(&q1);
```

```
if(value!=-1)
{
    printf("Dequeue operation successful %d\n", value);
}

break;
```

case 3:

```
value=peek(&q1);

if(value!=-1)
{
    printf("Front element: %d\n", value);
}

break;
```

case 4:

```
displayQueue(&q1);

break;
```

case 5:

```
printf("Number of elements in queue: %d\n", countElements(&q1));
break;
```

case 6:

```
clearQueue(&q1);

break;
```

case 7:

```
printf("Exit from program\n");
```

```

    exit(0);

default:
    printf("Invalid operation\n");

}

return 0;
}

```

Output:

```

Circular Queue Operations:
1. ENQUEUE
2. DEQUEUE
3. PEEK
4. Display Queue
5. Count Elements
6. Clear Queue
7. EXIT
Enter your choice:1
Enter value to Enqueue: 20
Enqueued: 20

Circular Queue Operations:
1. ENQUEUE
2. DEQUEUE
3. PEEK
4. Display Queue
5. Count Elements
6. Clear Queue
7. EXIT
Enter your choice:5
Number of elements in queue: 1

Circular Queue Operations:
1. ENQUEUE
2. DEQUEUE
3. PEEK
4. Display Queue
5. Count Elements
6. Clear Queue
7. EXIT
Enter your choice:

```

Section-D (Trees)

ExperimentNo : 1

Program Description:

Implementation of binary search tree .

Solution:

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

struct Node{
    int data;
    struct Node*left;
    struct Node*right;
};

struct Node*createNode(int data)
{
    struct Node*newNode=(struct Node*)malloc(sizeof(struct Node));
    newNode->data=data;
    newNode->left=NULL;
    newNode->right=NULL;
    return newNode;
}

struct Node*insert(struct Node*root, int data)
{
    if(root==NULL)
    {
```

```

    return createNode(data);
}

if(data<root->data)
{
    root->left=insert(root->left, data);
}

else if(data>root->data)
{
    root->right=insert(root->right, data);
}

return root;
}

```

```

struct Node*search(struct Node*root, int key)
{
    if(root==NULL || root->data==key)
    {
        return root;
    }

    if(key<root->data)
    {
        return search(root->left, key);
    }

    else
    {
        return search(root->right, key);
    }
}

```

```
struct Node*findMin(struct Node*root)
```

```
{  
    while(root!=NULL && root->left!=NULL)  
    {  
        root=root->left;  
    }  
  
    return root;  
}
```

```
struct Node*findMax(struct Node*root)  
{  
    while(root!=NULL && root->right!=NULL)  
    {  
        root=root->right;  
    }  
  
    return root;  
}
```

```
struct Node*deleteNode(struct Node*root, int key)  
{  
    if (root==NULL)  
    {  
        return root;  
    }  
  
    if (key<root->data)  
    {  
        root->left=deleteNode(root->left, key);  
    }  
  
    else if(key>root->data)
```

```

{
    root->right=deleteNode(root->right, key);
}
else
{
    if(root->left==NULL)
    {
        struct Node*temp=root->right;
        free(root);
        return temp;
    }
    else if(root->right==NULL)
    {
        struct Node*temp=root->left;
        free(root);
        return temp;
    }
}

struct Node*temp=findMin(root->right);
root->data=temp->data;
root->right=deleteNode(root->right, temp->data);
}

return root;
}


```

```

void inOrder(struct Node*root)
{
    if (root!=NULL)
    {
        inOrder(root->left);
        inOrder(root->right);
    }
}
```

```
    printf("%d", root->data);

    inOrder(root->right);

}

}
```

```
void preOrder(struct Node*root)

{
    if(root!=NULL)

    {
        printf("%d", root->data);

        preOrder(root->left);

        preOrder(root->right);

    }

}
```

```
void postOrder(struct Node*root)

{
    if(root!=NULL)

    {
        postOrder(root->left);

        postOrder(root->right);

        printf("%d", root->data);

    }

}
```

```
int countNodes(struct Node*root)

{
    if(root==NULL)
```

```
    return 0;  
  
    return 1+countNodes(root->left)+countNodes(root->right);  
  
}
```

```
int height(struct Node*root)  
{  
    if(root==NULL)  
  
        return 0;  
  
    int leftHeight=height(root->left);  
  
    int rightHeight=height(root->right);  
  
    return (leftHeight > rightHeight ? leftHeight:rightHeight)+1;  
  
}
```

```
int main()  
{  
    struct Node*root=NULL;  
  
    root=insert(root,50);  
  
    root=insert(root,30);  
  
    root=insert(root,70);  
  
    root=insert(root,20);  
  
    root=insert(root,40);  
  
    root=insert(root,60);  
  
    root=insert(root,80);  
  
    printf("In-order Traversal: ");
```

```

inOrder(root);
printf("\n");

printf("Pre-order Traversal: ");
preOrder(root);
printf("\n");

printf("Post-order Traversal: ");
postOrder(root);
printf("\n");

printf("Number of nodes: %d\n", countNodes(root));
printf("Height of tree: %d\n", height(root));

int key=40;
struct Node*found=search(root,key);
if(found!=NULL)
{
    printf("Elements %d found.\n", key);
}
else
{
    printf("Elements %d not found.\n", key);
}

struct Node*minNode=findMin(root);
struct Node*maxNode=findMax(root);

```

```

printf("Minimum value: %d\n",minNode->data);

printf("Maximum vlaue: %d\n",maxNode->data);

printf("Deleting node 50...\n");

root=deleteNode(root, 50);

printf("In-order after deletion: ");

inOrder(root);

printf("\n");

return 0;
}

```

Output:

```

Binary Search Tree
Binary Search Tree > Binary Search Tree > My Mac
Finished running Binary Search Tree ▲ 1

In-order Traversal: 20304050607080
Pre-order Traversal: 50302040706080
Post-order Traversal: 20403060807050
Number of nodes: 7
Height of tree: 3
Elements 40 found.
Minimum value: 20
Maximum vlaue: 80
Deleting node 50...
In-order after deletion: 203040607080
Program ended with exit code: 0

```

Experiment No : 2

Program Description:

Solution:

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

struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

struct Node* buildBSTFromPreOrderUtil(int preOrder[], int* preIndex, int key, int min, int max, int n)

{



    if(*preIndex >= n)
        return NULL;
```

```

struct Node* root = NULL;

if(key > min && key < max)
{
    root = createNode(key);
    (*preIndex)++;
}

if(*preIndex < n)
{
    root->left = buildBSTFromPreOrderUtil(preOrder, preIndex, preOrder[*preIndex], min, key, n);
}

if(*preIndex < n)
{
    root->right = buildBSTFromPreOrderUtil(preOrder, preIndex, preOrder[*preIndex], key, max, n);
}

return root;
}

```

```
struct Node* buildBSTFromPreOrder(int preOrder[], int n)
```

```
{
int preIndex = 0;

return buildBSTFromPreOrderUtil(preOrder, &preIndex, preOrder[0], INT_MIN, INT_MAX, n);
}
```

```
struct Node* buildBSTFromPostOrderUtil(int postOrder[], int* postIndex, int key, int min, int max, int n)
```

```
{
```

```

if(*postIndex < 0) return NULL;

struct Node* root = NULL;
if(key > min && key < max)
{
    root = createNode(key);
    (*postIndex)--;

    if(*postIndex >= 0)
    {
        root->right = buildBSTFromPostOrderUtil(postOrder, postIndex, postOrder[*postIndex], key, max, n);
    }
    if(*postIndex >= 0)
    {
        root->left = buildBSTFromPostOrderUtil(postOrder, postIndex, postOrder[*postIndex], min, key, n);
    }
}
return root;
}

struct Node* buildBSTFromPostOrder(int postOrder[], int n)
{
    int postIndex = n - 1;
    return buildBSTFromPostOrderUtil(postOrder, &postIndex, postOrder[n-1], INT_MIN, INT_MAX, n);
}

void inOrder(struct Node* root)
{
    if (root != NULL)
    {

```

```

inOrder(root->left);
printf("%d ", root->data);
inOrder(root->right);

}

void preOrder(struct Node* root) {
    if (root != NULL)
    {
        printf("%d ", root->data);
        preOrder(root->left);
        preOrder(root->right);
    }
}

void postOrder(struct Node* root) {
    if (root != NULL)
    {
        postOrder(root->left);
        postOrder(root->right);
        printf("%d ", root->data);
    }
}

int main()
{
    int preOrderArr[] = {50, 30, 20, 40, 70, 60, 80};
    int n = sizeof(preOrderArr) / sizeof(preOrderArr[0]);

    struct Node* rootFromPreOrder = buildBSTFromPreOrder(preOrderArr, n);
}

```

```

printf("Tree built from PreOrder Traversal\n");

printf("In-order: "); inOrder(rootFromPreOrder); printf("\n");

printf("Post-order: "); postOrder(rootFromPreOrder); printf("\n\n");

int postOrderArr[] = {20, 40, 30, 60, 80, 70, 50};

struct Node* rootFromPostOrder = buildBSTFromPostOrder(postOrderArr, n);

printf("Tree built from PostOrder Traversal\n");

printf("In-order: "); inOrder(rootFromPostOrder); printf("\n");

printf("Pre-order: "); preOrder(rootFromPostOrder); printf("\n");

return 0;
}

```

Output:

The screenshot shows the Xcode IDE interface with the project 'Conversion of BST' open. The left sidebar shows the file structure with 'main.c' selected. The main editor window displays C code for creating a binary search tree from pre-order traversal. The bottom right pane shows the terminal output of the program execution.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <limits.h>
4
5 struct Node {
6     int data;
7     struct Node* left;
8     struct Node* right;
9 };
10
11 struct Node* createNode(int data) {
12     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
13     newNode->data = data;
14     newNode->left = NULL;
15     newNode->right = NULL;
16     return newNode;
17 }
18
19 struct Node* buildBSTFromPreOrderUtil(int preOrder[], int* preIndex, int key, int min, int max, int n) {

```

Tree built from PreOrder Traversal
In-order: 20 30 40 50 60 70 80
Post-order: 20 40 30 60 80 70 50

Tree built from PostOrder Traversal
In-order: 20 30 40 50 60 70 80
Pre-order: 50 30 20 40 70 60 80
Program ended with exit code: 0

Section-E (Sorting & Searching)

Experiment No : 1

Program description:

1). SELECTION-

Solution:

```
#include <stdio.h>

void selection_sort(int arr[], int n) {
    for (int i = 0; i < n-1; i++) {
        int min_idx = i;
        for (int j = i+1; j < n; j++) {
            if (arr[j] < arr[min_idx]) {
                min_idx = j;
            }
        }
        int temp = arr[min_idx];
        arr[min_idx] = arr[i];
        arr[i] = temp;
    }
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr)/sizeof(arr[0]);

    printf("Original: ");
    for (int i = 0; i < n; i++) printf("%d ", arr[i]);
    printf("\n");

    int copy[] = {64, 34, 25, 12, 22, 11, 90};
    selection_sort(copy, n);

    printf("Selection: ");
    for (int i = 0; i < n; i++) printf("%d ", copy[i]);
    printf("\n\n");
    return 0;
}
```

Output:

```
Output
Original: 64 34 25 12 22 11 90
Selection: 11 12 22 25 34 64 90
==== Code Execution Successful ===
```

Experiment No : 2

Program description

2.) INSERTION SORT PROGRAM

Solution:

```
#include <stdio.h>

void insertion_sort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr)/sizeof(arr[0]);

    printf("Original: ");
    for (int i = 0; i < n; i++) printf("%d ", arr[i]);
    printf("\n");

    int copy[] = {64, 34, 25, 12, 22, 11, 90};
    insertion_sort(copy, n);

    printf("Insertion: ");
    for (int i = 0; i < n; i++) printf("%d ", copy[i]);
    printf("\n\n");
    return 0;
}
```

Output:

```
Output
Original: 64 34 25 12 22 11 90
Insertion: 11 12 22 25 34 64 90
==== Code Execution Successful ====
```

Experiment No : 3

Program description

3.) QUICK SORT PROGRAM

Solution :

```
#include <stdio.h>

int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1;

    for (int j = low; j < high; j++) {
        if (arr[j] <= pivot) {
            i++;
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;
    return i + 1;
}

void quick_sort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quick_sort(arr, low, pi - 1);
        quick_sort(arr, pi + 1, high);
    }
}

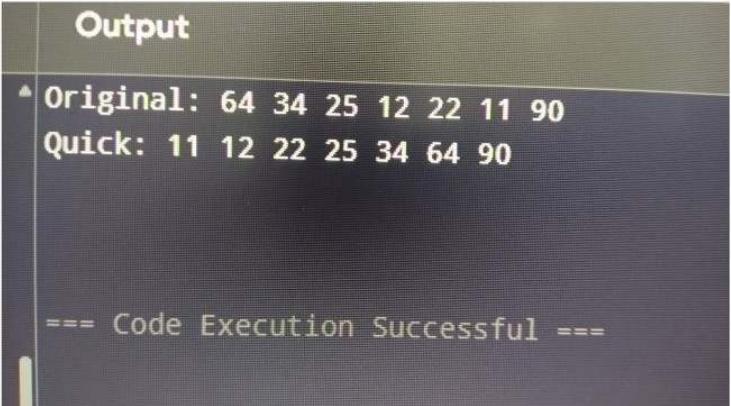
int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr)/sizeof(arr[0]);

    printf("Original: ");
    for (int i = 0; i < n; i++) printf("%d ", arr[i]);
    printf("\n");

    int copy[] = {64, 34, 25, 12, 22, 11, 90};
    quick_sort(copy, 0, n-1);
```

```
printf("Quick: ");
for (int i = 0; i < n; i++) printf("%d ", copy[i]);
printf("\n\n");
return 0;
}
```

Output



The screenshot shows a terminal window with a dark background and light-colored text. At the top, the word "Output" is displayed in a bold, white font. Below it, the text "Original: 64 34 25 12 22 11 90" is shown in white. Underneath that, the sorted array "Quick: 11 12 22 25 34 64 90" is displayed in white. At the bottom of the window, the message "==== Code Execution Successful ===" is printed in white. The terminal window has a vertical scroll bar on the left side.

```
Original: 64 34 25 12 22 11 90
Quick: 11 12 22 25 34 64 90
==== Code Execution Successful ===
```

Experiment No : 4

Program description

4.) MERGE SORT PROGRAM :

Solution :

```
#include <stdio.h>

void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int L[n1], R[n2];
    for (int i = 0; i < n1; i++) L[i] = arr[left + i];
    for (int j = 0; j < n2; j++) R[j] = arr[mid + 1 + j];

    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

if (arr[mid] == key)
    return mid;

if (arr[mid] < key)
    low = mid + 1;
else
    high = mid - 1;
}
```

```

return -1;
}

int main() {
    int arr[] = {11, 12, 22, 25, 34, 64, 90};
    int n = sizeof(arr)/sizeof(arr[0]);
    int key = 25;

    printf("Sorted array: ");
    for (int i = 0; i < n; i++) printf("%d ", arr[i]);
    printf("\n");

    int result = binary_search(arr, n, key);

    if (result != -1)
        printf("Element %d found at index %d\n", key, result);
    else
        printf("Element %d not found\n", key);

    return 0;
}

void merge_sort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        merge_sort(arr, left, mid);
        merge_sort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr)/sizeof(arr[0]);

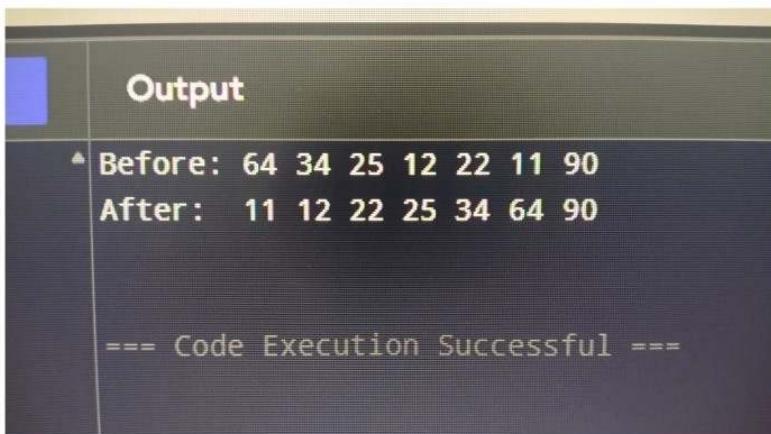
    printf("Original: ");
    for (int i = 0; i < n; i++) printf("%d ", arr[i]);
    printf("\n");

    int copy[] = {64, 34, 25, 12, 22, 11, 90};
    merge_sort(copy, 0, n-1);

    printf("Merge: ");
    for (int i = 0; i < n; i++) printf("%d ", copy[i]);
    printf("\n");
    return 0;
}

```

Output:



The screenshot shows a terminal window with a blue header bar. The main area is dark gray with white text. It displays two lines of data preceded by an asterisk (*), followed by a success message.

```
* Before: 64 34 25 12 22 11 90
* After: 11 12 22 25 34 64 90
--- Code Execution Successful ---
```

Experiment No : 5

Program description

implementation of Binary Search on a list of numbers stored in an Array.

Solution :

```
#include <stdio.h>

int binary_search(int arr[], int n, int x) {
    int l = 0, r = n - 1;

    while (l <= r) {
        int m = l + (r - l) / 2;

        if (arr[m] == x) return m;
        if (arr[m] < x) l = m + 1;
        else r = m - 1;
    }
    return -1;
}

int main() {
    int a[] = {2, 5, 8, 12, 16, 23, 38, 45, 57};
    int n = 9;

    printf("Array: ");
    for(int i=0; i<n; i++) printf("%d ", a[i]);
    printf("\n");

    int found = binary_search(a, n, 23);
    if(found != -1) printf("23 found at %d\n", found);
    else printf("Not found\n");

    return 0;
}
```

Output:

```
Output
^ Array: 2 5 8 12 16 23 38 45 67 89
  23 found at position 5

    === Code Execution Successful ===
```

Experiment No : 6

Program Description

Implementation of Binary Search on a list of strings stored in an Array

Solution:

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

int binary_search_str(char *arr[], int size, char *target) {
    int low = 0;
    int high = size - 1;
    while (low <= high) {
        int mid = low + (high - low) / 2;

        if (strcmp(arr[mid], target) == 0)
            return mid;

        if (strcmp(arr[mid], target) < 0)
            low = mid + 1;
        else
            high = mid - 1;
    }

    return -1;
}

int main() {
    char *words[] = {"apple", "banana", "cherry", "date", "elderberry",
                    "fig", "grape", "honeydew", "kiwi", "lemon"};
    int count = 10;
    char *search_word = "kiwi";

    printf("Array: ");

    for (int i = 0; i < count; i++)
        printf("%s ", words[i]);
    printf("\n");

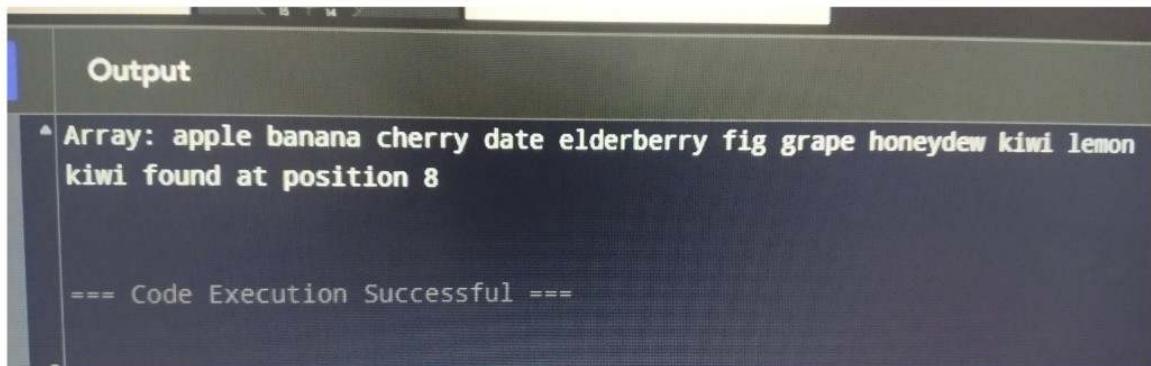
    int position = binary_search_str(words, count, search_word);

    if (position != -1)
        printf("%s found at position %d\n", search_word, position);
}
```

```
else
printf("%s not found\n", search_word);

return 0;
}
```

Output:



```
Output
^ Array: apple banana cherry date elderberry fig grape honeydew kiwi lemon
kiwi found at position 8
==== Code Execution Successful ====
```

Experiment No : 7

Program Description

Implementation of Binary Search on a list of strings stored in an Array.

Solution:

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

int bin_search_str(char *arr[], int n, char *key) {
    int left = 0, right = n - 1;

    while (left <= right) {
        int middle = left + (right - left) / 2;

        if (strcmp(arr[middle], key) == 0)
            return middle;

        if (strcmp(arr[middle], key) < 0)
            left = middle + 1;
        else
            right = middle - 1;
    }
    return -1;
}

int main() {
    char *names[] = {"alice", "bob", "charlie", "david", "emma",
                    "frank", "grace", "harry", "ivy", "jack"};
    int size = 10;
    char *find_name = "grace";

    printf("List: ");
    for (int i = 0; i < size; i++)
        printf("%s ", names[i]);
    printf("\n");

    int found = bin_search_str(names, size, find_name);

    if (found != -1)
        printf("%s is at position %d\n", find_name, found);
    else
        printf("%s not in list\n", find_name);

    return 0;
}
```

Output:

```
Output
List: alice bob charlie david emma frank grace harry ivy jack
grace is at position 6
--- Code Execution Successful ---
```

Experiment No : 8

Program Description

Implementation of Binary Search on a list of strings stored in a Single Linked List (optional).

Solution:

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

struct Node {
    char *data;
    struct Node* next;
};

struct Node* head = NULL;

void insert(char *str) {
    struct Node* newnode = (struct Node*)malloc(sizeof(struct Node));
    newnode->data = str;
    newnode->next = head;
    head = newnode;
}

int binary_search_str(struct Node* head, int n, char *key) {
    struct Node* left = head;
    struct Node* right = head;
    int left_count = 0;

    while (right->next) {
        right = right->next;
        if (left_count % 2 == 0 && left->next)
            left = left->next;
        left_count++;
    }
    struct Node* l_ptr = head;
    struct Node* r_ptr = right;
    int l_idx = 0;
```

```

while (l_idx <= n/2) {
    struct Node* mid_ptr = l_ptr;
    int steps = (n/2 - l_idx) / 2;
    for (int i = 0; i < steps; i++) {
        if (mid_ptr && mid_ptr->next) mid_ptr = mid_ptr->next;
    }

    if (strcmp(mid_ptr->data, key) == 0)
        return l_idx + steps;

    if (strcmp(mid_ptr->data, key) < 0) {
        l_ptr = mid_ptr->next;
        l_idx += steps + 1;
    } else {
        r_ptr = mid_ptr;
        n = l_idx + steps;
    }
}

return -1;
}

int main() {
    insert("zebra");
    insert("yellow");
    insert("xray");
    insert("wolf");
    insert("violet");
    insert("uncle");
    insert("tiger");
    insert("snake");
    insert("rabbit");
    insert("queen");

printf("List: ");
    struct Node* temp = head;
    int count = 0;
    while (temp) {
        printf("%s ", temp->data);
        temp = temp->next;
        count++;
    }
    printf("\n");
}

```

```
int pos = binary_search_str(head, count, "snake");
if (pos != -1)
    printf("snake found at position %d\n", pos);
else
    printf("snake not found\n");

return 0;
}
```

Output:

Run

Output

List: queen rabbit snake tiger uncle violet wolf xray yellow zebra
snake found at position 2

--- Code Execution Successful ---