DATA STRUCTURES LAB

(AI & DS 251)

Submitted To:

Dr Soumi Ghosh
Gupta

Submitted By:

Student Name: Yugansh

Roll no: 02814811921

Semester: 3

Group: AI DS 2

Maharaja Agrasen Institute of Technology, PSP Area,Sector – 22, Rohini, New Delhi – 110086

DATA STRUCTURE LAB

PRACTICAL RECORD

Paper Code : AIDS-251

Name of the student : Yugansh Gupta

University Roll No. : 02814811921

Branch : Artificial Intelligence and Data Science

Section/ Group : AI DS2

Exp. no	Experiment Name	Date of performance	Date of checking	Remarks	Marks
1	Perform Linear Search and binary search in an array using menu driven program.				
2	Implement sparse matrix using arrays				
3	Program to implement stack using an array.				
4	Create a Stack and perform Pop, push, and Traverse operations on the stack using a linear Linked List.				
5	Create a Linked List with nodes having information about students insert node at the specified position				
6	Write a code to implement the multiplication of two matrix				
7	Create a Linear Queue using Linked List and implement different operations such as insert, delete, and display the queue elements				
8	Create a doubly linked list with nodes having information about an employee and perform Insertion at front of doubly linked list and perform deletion at end of that doubly linked list.				
9	Implement the following sorting techniques: a. Insertion sort b. Merge sort c. Bubble sort d. Selection sort				
10	Create a circular linked list having information about a college and perform Insertion at the front end and perform deletion at the end.				
11	Create a Binary Tree and perform Tree Traversals (Preorder, Postorder, Inorder) using the concept of recursion.				
12	Implement insertion, deletion, and display (Inorder, Preorder, Postorder) on binary search tree with the information in the tree about the details of an automobile (type, company, year of make).				

PROGRAM-1

AIM

Perform Linear Search and binary search in an array using menu driven program.

THEORY

A linear search is the simplest method of searching a data set. Starting at the beginning of the data set, each item of data is examined until a match is made. Once the item is found, the search ends.

Binary search is a 'divide and conquer' algorithm which requires the initial array to be sorted before searching. It is called binary because it splits the array into two halves as part of the algorithm. Initially, a binary search will look at the item in the middle of the array and compare it to the search terms.

CODE

```
#include<stdio.h>
int binary_search(){
    int n;
    printf("Enter the number of elements do you want to enter:");
    scanf("%d",&n);
    int arr[n];
    printf("enter the elements\n");
    int i;
    for(i=0;i<n;i=i+1){
        int c;
        scanf("%d",&c);
        arr[i]=c;
    }
    int d,a,j;
    for (i = 0; i < n; ++i) {
        for (j = i + 1; j < n; ++j){
            if (arr[i] > arr[j]) {
                a =  arr[i];
                arr[i] = arr[j];
                arr[j] = a;
            }
        }
    }
    printf("Sorted array:");
    for(i=0;i<n;i=i+1){
        printf("%d ",arr[i]);
    }
    printf("\nEnter the target value:");
    scanf("%d",&d);
    printf("Index:");
    int start=0;
    int end=n-1;
    int mid;
    int ans=-1;
    while(end>=start){
        int mid=((start+end)/2);
        if(arr[mid]>d){
            end=mid-1;
```

```
        }
        else if(arr[mid]<d){
            start=mid+1;
        }
        else{
            printf("%d\n",mid);
            ans=1;
            break;
        }
    }
    if(ans==-1){
        printf("-1\n");
    }
}
int linear_search(){
    int n;
    printf("Enter the number of elements do you want to enter:");
    scanf("%d",&n);
    int arr[n];
    printf("enter the elements\n");
    int i;
    for(i=0;i<n;i=i+1){
        int c;
        scanf("%d",&c);
        arr[i]=c;
    }
    int d;
    int ans=-1;
    printf("Enter the target value:");
    scanf("%d",&d);
    for(int i=0;i<n;i=i+1){
        if(arr[i]==d){
            ans=i;
            break;
        }
    }
    printf("index:%d\n",ans);
}
int main(){
    int a=1;
    int ans;
    while(a==1){
        printf("\n");
        printf("1.linear search\n");
        printf("2.binary search\n");
        printf("3.Exit\n");
        printf("Enter your choice:");
        scanf("%d",&ans);
        if(ans==1){
            linear_search();
        }
        else if(ans==2){
            binary_search();
        }
```

```
        else if(ans==3){
            break;
        }
        else{
            printf("Invalid syntex");
        }
    }
}
```

OUTPUT

```
1.linear search
2.binary search
3.Exit
Enter your choice:2
Enter the number of elements do you want to enter:4
enter the elements
2 7 5 3
Sorted array:2 3 5 7
Enter the target value:3
Index:1

1.linear search
2.binary search
3.Exit
Enter your choice:1
Enter the number of elements do you want to enter:5
enter the elements
2 8 4 9 7
Enter the target value:0
index:-1

1.linear search
2.binary search
3.Exit
Enter your choice:3
PS D:\C\search>
```

PROGRAM-2

AIM

Implement sparse matrix using arrays

THEORY

Sparse matrices are those matrices that have the majority of their elements equal to zero. In other words, the sparse matrix can be defined as the matrix that has a greater number of zero elements than the non-zero elements.

CODE

```
#include<stdio.h>
int main(){
    int n,m;
    printf("Enter the number of rows:");
    scanf("%d",&m);
    printf("Enter the number of columns:");
    scanf("%d",&n);
    int arr[m][n];
    printf("enter the elements\n");
    int i;
    int j;
    for(i=0;i<m;i=i+1){
        for(j=0;j<n;j=j+1){
            int c;
            scanf("%d",&c);
            arr[i][j]=c;
        }
    }
    int ans=0;
    printf("Matrix is\n");
    for(i=0;i<m;i=i+1){
        for(j=0;j<n;j=j+1){
            printf("%d ",arr[i][j]);
            if(arr[i][j]==0){
                ans=ans+1;
            }
        }
        printf("\n");
    }
    printf("row column value");
    printf("\n");
    for(i=0;i<m;i=i+1){
        for(j=0;j<n;j=j+1){
            if(arr[i][j]!=0){
                printf("%d   %d      %d",i,j,arr[i][j]);
                printf("\n");
            }
        }
    }
    if(ans>(m*n)-ans){
        printf("matrix is sparse matrix");
    }
    else{
        printf("matrix is not a sparse matrix");}
}
```

OUTPUT

```
PS C:\Users\91921\Desktop\C> cd "c:\Users\91921\Desktop\C\" ; if ($?) { gcc sparse.c -o sparse
} ; if ($?) { .\sparse }
Enter the number of rows:3
Enter the number of columns:3
enter the elements
1 2 0 6 0 0 0 3 0
Matrix is
1 2 0
6 0 0
0 3 0
row column value
0   0      1
0   1      2
1   0      6
2   1      3
matrix is sparse matrix
```

PROGRAM-3

AIM

Program to implement stack using an array.

THEORY

Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO (Last in First Out) or FILO (First in Last Out). There are many real-life examples of a stack. Consider an example of plates stacked over one another in the canteen.

CODE

```
#include<stdio.h>
int top=-1;
int n;
void display(int arr[]){
    if(top==-1){
        printf("Empty stack\n");
        return;
    }
    int i=top;
    printf("Stack is\n");
    while(i>=0){
        printf("%d\n",arr[i]);
        i=i-1;
    }
}
void push(int arr[]){
    if(top>=n-1){
        printf("Overflow\n");
        return;
    }
    int x;
    printf("Enter the value:");
    scanf("%d",&x);
    top=top+1;
    arr[top]=x;
    display(arr);
}
void pop(int arr[]){
    if(top==-1){
        printf("Underflow\n");
        return;
    }
    printf("Item pop:%d\n",arr[top]);
    top=top-1;
    display(arr);

}
int main(){
    printf("Enter the size of stack:");
    scanf("%d",&n);
    int arr[n];
```

```
    int a=1;
    int ans;
    while(a==1){
        printf("\n");
        printf("1.Push\n");
        printf("2.Pop\n");
        printf("3.Display\n");
        printf("4.Exit\n");
        printf("Enter your choice:");
        scanf("%d",&ans);
        if(ans==1){
            push(arr);
        }
        else if(ans==2){
            pop(arr);
        }
        else if(ans==3){
            display(arr);
        }
        else if(ans==4){
            break;
        }
        else{
            printf("Invalid syntex");
        }
    }
}
```

OUTPUT

```
1.Push
2.Pop
3.Display
4.Exit
Enter your choice:2
Underflow

1.Push
2.Pop
3.Display
4.Exit
Enter your choice:1
Enter the value:3
Empty stack

1.Push
2.Pop
3.Display
4.Exit
Enter your choice:1
Enter the value:3
Stack is
3
```

```
1.Push
2.Pop
3.Display
4.Exit
Enter your choice:1
Enter the value:4
Stack is
4
3

1.Push
2.Pop
3.Display
4.Exit
Enter your choice:1
Enter the value:7
Stack is
7
4
3

1.Push
2.Pop
3.Display
4.Exit
Enter your choice:1
Overflow

1.Push
2.Pop
3.Display
4.Exit
Enter your choice:4
PS D:\C\stack>
```

PROGRAM-4

AIM

Create a Stack and perform Pop, push, and Traverse operations on the stack using a linear Linked List.

THEORY

Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO (Last in First Out) or FILO (First in Last Out). There are many real-life examples of a stack. Consider an example of plates stacked over one another in the canteen.

A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers

CODE

```c
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node *next;
};
typedef struct Node Node;
Node* top = NULL;
void push(int value) {
    struct Node *newNode;
    newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    if (top == NULL) {
        newNode->next = NULL;
    } else {
        newNode->next = top;
    }
    top = newNode;
    printf("Node is Inserted\n\n");
}

int pop() {
    if (top == NULL) {
        printf("\nStack Underflow\n");
    } else {
        struct Node *temp = top;
        int temp_data = top->data;
        top = top->next;
        free(temp);
        printf("Popped element is :%d\n\n",temp_data);
    }
}

void display() {
    if (top == NULL) {
        printf("\nStack is empty\n");
    } else {
        printf("The stack is \n");
        struct Node *temp = top;
```

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

int main() {
    int choice, value;
    printf("\nImplementation of Stack using Linked List\n");
    while (1) {
        printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");
        printf("\nEnter your choice : ");
        scanf("%d", &choice);
        switch (choice) {
        case 1:
            printf("\nEnter the value to insert: ");
            scanf("%d", &value);
            push(value);
            break;
        case 2:
            pop();
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
            break;
        default:
            printf("\nWrong Choice\n");
        }
    }
}
```

OUTPUT

```
PS D:\C\linked list> cd "d:\C\linked list
\" ; if ($?) { gcc stack_linked_list.c -o
 stack_linked_list } ; if ($?) { .\stack_
linked_list }

Implementation of Stack using Linked List
1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 3

Stack is empty
1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 2

Stack Underflow
1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 1

Enter the value to insert: 6
Node is Inserted

1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 1

Enter the value to insert: 4
Node is Inserted

1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 3
The stack is
4
6

1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 2
Popped element is :4

1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 3
The stack is
6

1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 4
PS D:\C\linked list>
```

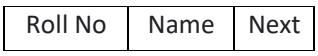
PROGRAM-5

AIM

Create a Linked List with nodes having information about students insert node at the specified position

THEORY

A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers



CODE

```c
#include <stdio.h>
#include <stdlib.h>
#include<string.h>
struct Node {
    int rollno;
    char name[50];
    struct Node *next;
};
typedef struct Node node;
node* top = NULL;
void enter(int value,char* str) {
    node* newNode;
    newNode = (node*)malloc(sizeof(node));
    newNode->rollno = value;
    strcpy(newNode->name,str);
    if (top == NULL) {
        newNode->next = NULL;
    } else {
        newNode->next = top;
    }
    top = newNode;
    printf("Student details is Inserted\n\n");
}

void display(int roll) {
    if (top == NULL) {
        printf("\nno student data found\n");
    } else {
        node *temp = top;
        while (temp->next != NULL && temp->rollno != roll) {
            temp = temp->next;
        }
        if(temp->rollno == roll){
            printf("name:%s\n\n",temp->name);
        }
        else{
            printf("roll number not found\n\n");
        }

    }
}
```

```c
int main() {
    int choice, value,r;
    printf("\nImplementation of Stack using Linked List\n");
    while (1) {
        printf("1. enter data\n2. find student name\n3.Exit\n");
        printf("\nEnter your choice : ");
        scanf("%d", &choice);
        switch (choice) {
        case 1:
            printf("\nEnter the rollno to insert: ");
            scanf("%d", &value);
            char str[50];
            printf("Enter the name of student:");
            scanf("%s",str);
            enter(value,str);
            break;
        case 2:
            printf("\nEnter the rollno which you want to find:");
            scanf("%d",&r);
            display(r);
            break;
        case 3:
            exit(0);
            break;
        default:
            printf("\nWrong Choice\n");
        }
    }
}

}
```

OUTPUT

```
Implementation of Stack using Linked List
1. enter data
2. find student name
3.Exit

Enter your choice : 2

Enter the rollno which you want to find:1

no student data found
1. enter data
2. find student name
3.Exit

Enter your choice : 1

Enter the rollno to insert: 1
Enter the name of student:Yugansh
Student details is Inserted

1. enter data
2. find student name
3.Exit

Enter your choice : 1

Enter the rollno to insert: 2
Enter the name of student:Abc
Student details is Inserted

1. enter data
2. find student name
3.Exit

Enter your choice : 2

Enter the rollno which you want to find:1
name:Yugansh

1. enter data
2. find student name
3.Exit

Enter your choice : 3
PS D:\C\student_linkedlist>
```

PROGRAM-6

AIM

Write a code to implement the multiplication of two matrix

THEORY

We can multiply two matrices in java using binary * operator and executing another loop. A matrix is also known as array of arrays. We can add, subtract and multiply matrices.

In case of matrix multiplication, one row element of first matrix is multiplied by all columns of second matrix.

CODE

```
#include<stdio.h>
void main(){
    int m1,n1,m2,n2;
    printf("First Matrix\n");
    printf("Rows:");
    scanf("%d",&m1);
    printf("column:");
    scanf("%d",&n1);
    printf("Second Matrix\n");
    printf("Rows:");
    scanf("%d",&m2);
    printf("column:");
    scanf("%d",&n2);
    if(n1!=m2){
        printf("matrix cannot be multiplied");
        return;
    }
    int a[m1][n1],b[m2][n2],c[m1][n2];
    printf("Elements of first matrix\n");
    for(int i=0;i<m1;i++){
        for(int j=0;j<n1;j++){
            scanf("%d",&a[i][j]);
        }
    }
    printf("Elements of second matrix\n");
    for(int i=0;i<m2;i++){
        for(int j=0;j<n2;j++){
            scanf("%d",&b[i][j]);
        }
    }
    for (int i = 0; i < m1; i++) {
        for (int j = 0; j < n2; j++) {
            c[i][j]=0;
        }
    }
    for (int i = 0; i < m1; i++) {
        for (int j = 0; j < n2; j++) {
            for (int k = 0; k < n1; k++) {
                c[i][j] += a[i][k] * b[k][j];
            }
```

```
        }
    }
    printf("\nOutput Matrix:\n");
    for (int i = 0; i < m1; i++) {
        for (int j = 0; j < n2; j++) {
            printf("%d  ", c[i][j]);
        }
        printf("\n");
    }
}
```

OUTPUT

```
PS D:\C\multiplication> cd "d:\C\multiplicatio
n\" ; if ($?) { gcc m.c -o m } ; if ($?) { .\m
 }
First Matrix
Rows:2
column:3
Second Matrix
Rows:3
column:3
Elements of first matrix
1 2 3
5 6 7
Elements of second matrix
1 6 8
2 5 9
1 6 3

Output Matrix:
8  34  35
24  102  115
PS D:\C\multiplication>
```

PROGRAM-7

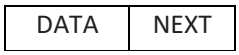
AIM

Create a Linear Queue using Linked List and implement different operations such as insert, delete, and display the queue elements.

THEORY

A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers

A queue in C is basically a linear data structure to store and manipulate the data elements. It follows the order of **First In First Out (FIFO)**.



CODE

```c
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node * next;
};
struct node * front = NULL;
struct node * rear = NULL;
void enqueue(int value) {
    struct node * ptr;
    ptr = (struct node * ) malloc(sizeof(struct node));
    ptr -> data = value;
    ptr -> next = NULL;
    if ((front == NULL) && (rear == NULL)) {
        front = rear = ptr;
    } else {
        rear -> next = ptr;
        rear = ptr;
    }
    printf("Node is Inserted\n\n");
}
void dequeue() {
    if (front == NULL) {
        printf("Underflow\n\n");
        return;
    }
    else {
        struct node * temp = front;
        int temp_data = front -> data;
        front = front -> next;
        free(temp);
        printf("Popped element is :%d\n\n",temp_data);
    }
}
void display() {
    struct node * temp;
    if ((front == NULL) && (rear == NULL)) {
        printf("Queue is Empty\n\n");
```

```
    } else {
        printf("The queue is \n");
        temp = front;
        while (temp->next) {
            printf("%d--->", temp -> data);
            temp = temp -> next;
        }
        printf("%d\n\n",temp->data);
    }
}

void main() {
    int choice, value;
    printf("\nImplementation of Queue using Linked List\n");
    while (1) {
        printf("1.Enqueue\n");
        printf("2.Dequeue\n");
        printf("3.Display\n");
        printf("4.Exit\n");
        printf("Enter your choice:");
        scanf("%d",&choice);
        if(choice==1){
            printf("Enter the value to insert: ");
            scanf("%d",&value);
            enqueue(value);
        }
        else if(choice==2){
            dequeue();

        }
        else if(choice==3){
            display();
        }
        else if(choice==4){
            break;
        }
        else{
            printf("Wrong Choice\n\n");
        }
    }
}
```

OUTPUT

```
PS D:\C\linked list> cd "d:\C\linked list\" ;
if ($?) { gcc queue.c -o queue } ; if ($?) { .
\queue }

Implementation of Queue using Linked List
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice:2
Underflow

1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice:3
Queue is Empty

1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice:1
Enter the value to insert: 6
Node is Inserted

1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice:1
Enter the value to insert: 4
Node is Inserted

1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice:1
Enter the value to insert: 3
Node is Inserted

1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice:3
The queue is
6--->4--->3

1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice:2
Popped element is :6

1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice:4
PS D:\C\linked list>
```

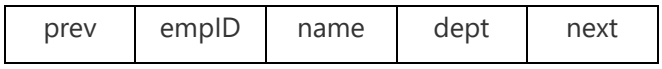
PROGRAM-8

AIM

Create a doubly linked list with nodes having information about an employee and perform Insertion at front of doubly linked list and perform deletion at end of that doubly linked list.

THEORY

Doubly linked list is a complex type of linked list in which a node contains a pointer to the previous as well as the next node in the sequence. Therefore, in a doubly linked list, a node consists of three parts: node data, pointer to the next node in sequence (next pointer), pointer to the previous node (previous pointer).



CODE

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

struct Employee{
    struct Employee* prev;
    int empID;
    char name[20];
    char dept[20];
    struct Employee* next;
};
typedef struct Employee Node;

void new_ele(Node** st,int empId,char* name,char* dept){
    Node* temp = (Node*)malloc(sizeof(Node));
    temp->empID = empId;
    strcpy(temp->name,name);
    strcpy(temp->dept,dept);
    temp->next = *st;
    temp->prev = NULL;
    (*st)->prev = temp;
    (*st) = temp;
}

void del_ele(Node** st){
    Node* temp = *st;
    if((*st)==NULL){
        printf("No Data Found\n");
        return;
    }
    while(temp->next != NULL){
        temp = temp->next;
    }
    Node* last = temp;
    if (last == (*st))
    {
```

```
        (*st) = NULL;
        return;
    }

    Node* sec_last = last->prev;
    sec_last->next = NULL;
}

void print_ll(Node** st)
{
    Node* temp;
    temp = *st;
    while(temp!=NULL){
        printf("Employee Id :%d\n",temp->empID);
        printf("Employee Name :%s\n",temp->name);
        printf("Employee Dept :%s\n",temp->dept);
        printf("----------\n");
        temp = temp->next;
    }
}

int main(){
    Node* start =NULL;
    while(true)
    {
        int ch;
        printf("\n1.Enter employee in the list\n");
        printf("2.Delete employee from list\n");
        printf("3.Display the list\n");
        printf("4.Exit\n");
        printf("Enter your choice:");
        scanf("%d",&ch);
        if(ch==1){
            if(start == NULL)
            {
                start = (Node*)malloc(sizeof(Node));
                printf("\nEnter Employee Id for first Employee :");
                scanf("%d",&start->empID);
                printf("Enter name for first Employee :");
                scanf("%s",start->name);
                printf("Enter Employee department :");
                scanf("%s",start->dept);
                start->next = NULL;
                start->prev = NULL;
            }else{
                int empId;
                char name[20];
                char dept[20];
                printf("\nEnter new Employee Id :");
                scanf("%d",&empId);
                printf("Enter employee name :");
                scanf("%s",name);
                printf("Enter Employee Department :");
                scanf("%s",dept);
```

```
                new_ele(&start,empId,name,dept);
            }
        }
        else if(ch==2){
            del_ele(&start);
        }
        else if(ch==3){
            if(start == NULL){
                printf("No Data Found\n");
                continue;
            }
            printf("\nPrinting the Employee list\n");
            print_ll(&start);
        }
        else if(ch==4){
            break;
        }
        else{
            printf("you entered wrong choice");
        }
    }
}
```

OUTPUT

```
1.Enter employee in the list
2.Delete employee from list
3.Display the list
4.Exit
Enter your choice:2
No Data Found

1.Enter employee in the list
2.Delete employee from list
3.Display the list
4.Exit
Enter your choice:3
No Data Found

1.Enter employee in the list
2.Delete employee from list
3.Display the list
4.Exit
Enter your choice:1

Enter Employee Id for first Employee :101
Enter name for first Employee :abc
Enter Employee department :IT

1.Enter employee in the list
2.Delete employee from list
3.Display the list
4.Exit
Enter your choice:1

Enter new Employee Id :102
Enter employee name :pqr
Enter Employee Department :HR

Employee Name :pqr
Employee Dept :HR
----------
Employee Id :101
Employee Name :abc
Employee Dept :IT
----------
```

```
1.Enter employee in the list
2.Delete employee from list
3.Display the list
4.Exit
Enter your choice:2

1.Enter employee in the list
2.Delete employee from list
3.Display the list
4.Exit
Enter your choice:3

Printing the Employee list
Employee Id :102
Employee Name :pqr
Employee Dept :HR
----------

1.Enter employee in the list
2.Delete employee from list
3.Display the list
4.Exit
Enter your choice:2

1.Enter employee in the list
2.Delete employee from list
3.Display the list
4.Exit
Enter your choice:3
No Data Found

1.Enter employee in the list
2.Delete employee from list
3.Display the list
4.Exit
Enter your choice:4
PS D:\C\doubly linked list>
```

PROGRAM-9(a)

AIM

Implement the Insertion sort.

THEORY

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

CODE

```c
#include<stdio.h>
void main(){
    int n;
    printf("Enter the no of elements:");
    scanf("%d",&n);
    int a[n];
    printf("Enter numbers:");
    for(int i=0;i<n;i++){
        scanf("%d",&a[i]);
    }
    for(int i=1;i<n;i++){
        int b=a[i];
        int j=i-1;
        while(j>=0 && a[j]>b){
            a[j+1]=a[j];
            j--;
        }
        a[j+1]=b;
    }
    printf("sorted array:");
    for(int i=0;i<n;i++){
        printf("%d ",a[i]);
    }
}
```

OUTPUT

```
PS D:\C\sort> cd "d:\C\sort\" ; if ($?) { gcc
insertion_sort.c -o insertion_sort } ; if ($?)
 { .\insertion_sort }
Enter the no of elements:6
Enter numbers:11 10 6 45 78 50
sorted array:6 10 11 45 50 78
PS D:\C\sort>
```

PROGRAM-9(b)

AIM

Implement the Merge sort.

THEORY

The **Merge Sort** algorithm is a sorting algorithm that is based on the **Divide and Conquer** paradigm. In this algorithm, the array is initially divided into two equal halves and then they are combined in a sorted manner.

CODE

```
#include<stdio.h>
void merge(int arr[],int start,int end){
    int mid=(start+end)/2;
    int a[end-start+1];
    int i=0;
    int j=start;
    int k=mid+1;
    while(j<=mid && k<=end){
        if(arr[j]>arr[k]){
            a[i]=arr[k];
            k++;
        }
        else{
            a[i]=arr[j];
            j++;
        }
        i++;
    }
    while(j<=mid){
        a[i]=arr[j];
        i++;
        j++;
    }
    while(k<=end){
        a[i]=arr[k];
        i++;
        k++;
    }
    for(int b=0;b<=(end-start);b++){
        arr[b+start]=a[b];
    }
    return;

}
void sort(int arr[],int start,int end){
    if(start>=end){
        return;
    }
    int mid=(start+end)/2;
    sort(arr,start,mid);
    sort(arr,mid+1,end);
    merge(arr,start,end);
```

```
    return;
}
void main(){
    int n;
    printf("Enter the no of elements:");
    scanf("%d",&n);
    int a[n];
    printf("Enter numbers:");
    for(int i=0;i<n;i++){
        scanf("%d",&a[i]);
    }
    sort(a,0,n-1);
    printf("sorted array:");
    for(int i=0;i<n;i++){
        printf("%d ",a[i]);
    }
}
```

OUTPUT

```
PS D:\C\sort> cd "d:\C\sort\" ; if ($?) { gcc
merge_sort.c -o merge_sort } ; if ($?) { .\mer
ge_sort }
Enter the no of elements:7
Enter numbers:4 9 1 23 98 65 2
sorted array:1 2 4 9 23 65 98
PS D:\C\sort>
```

PROGRAM-9(c)

AIM

Implement the Bubble sort.

THEORY

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.

CODE

```c
#include<stdio.h>
int main(){
    int n;
    printf("Enter the number of elements:");
    scanf("%d",&n);
    int arr[n];
    printf("enter the elements:");
    int i;
    for(i=0;i<n;i=i+1){
        int c;
        scanf("%d",&c);
        arr[i]=c;
    }
    int d,a,j;
    for (i = 0; i < n; ++i) {
        for (j = i + 1; j < n; ++j){
            if (arr[i] > arr[j]) {
                a =  arr[i];
                arr[i] = arr[j];
                arr[j] = a;
            }
        }
    }
    printf("Sorted array:");
    for(i=0;i<n;i=i+1){
        printf("%d ",arr[i]);
    }
}
```

OUTPUT

```
PS D:\C\sort> cd "d:\C\sort\" ; if ($?) { gcc
bubble_sort.c -o bubble_sort } ; if ($?) { .\b
ubble_sort }
Enter the number of elements:7
enter the elements:10 8 46 92 34 12 6
Sorted array:6 8 10 12 34 46 92
PS D:\C\sort>
```

PROGRAM-9(d)

AIM

Implement the Selection sort.

THEORY

The **selection sort algorithm** sorts an array by repeatedly finding the minimum element (considering ascending order) from the unsorted part and putting it at the beginning.

CODE

```c
#include<stdio.h>
void main(){
    int n;
    printf("Enter the no of elements:");
    scanf("%d",&n);
    int a[n];
    printf("Enter numbers:");
    for(int i=0;i<n;i++){
        scanf("%d",&a[i]);
    }
    for(int i=0;i<n;i++){
        int min=i;
        for(int j=i+1;j<n;j++){
            if(a[min]>a[j]){
                min=j;
            }
        }
        int temp=a[i];
        a[i]=a[min];
        a[min]=temp;
    }
    printf("sorted array:");
    for(int i=0;i<n;i++){
        printf("%d ",a[i]);
    }
}
```

OUTPUT

```
PS D:\C\sort> cd "d:\C\sort\" ; if ($?) { gcc s
election_sort.c -o selection_sort } ; if ($?) {
 .\selection_sort }
Enter the no of elements:7
Enter numbers:98 43 54 12 23 44 57
sorted array:12 23 43 44 54 57 98
PS D:\C\sort>
```

PROGRAM-10

AIM

Create a circular linked list having information about a college and perform Insertion at the front end and perform deletion at the end.

THEORY

The **circular linked list** is a linked list where all nodes are connected to form a circle. In a circular linked list, the first node and the last node are connected to each other which forms a circle. There is no NULL at the end.

colID	name	next

CODE

```
//circular list implementation

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

struct College{
    int colID;
    char name[20];
    struct College* next;
};
typedef struct College Node;

void print_ll(Node** st)
{
    Node* temp = *st;
    if ((*st) == NULL)
    {
        printf("Empty List\n");
        return;
    }
    printf("\nPrinting the Employee list\n");
    while(temp->next!=(*st)){
        printf("Collegeee Id :%d\n",temp->colID);
        printf("College Name :%s\n",temp->name);
        printf("----------\n");
        temp = temp->next;
    }
    printf("Collegeee Id :%d\n",temp->colID);
    printf("College Name :%s\n",temp->name);
    printf("----------\n");

}

void new_ele(Node** st,int colID,char* name){
    Node* new = (Node*)malloc(sizeof(Node));
    new->colID = colID;
    strcpy(new->name,name);
```

```
    Node* temp = *st;
    while (temp->next!=(*st))
    {
        temp= temp->next;
    }
    temp->next = new;
    new->next = (*st);
    (*st) = new;

}

void del_ele(Node** st){
    Node* temp = *st;
    if((*st)==NULL){
        printf("No Data Found\n");
        return;
    }
    if (temp->next == (*st))
    {
        (*st)=NULL;
        return;
    }
    while(temp->next->next != (*st)){
        temp = temp->next;
    }
    temp->next = (*st);
}

int main(){
    Node* start =NULL;
    while(true)
    {
        int ch;
        printf("\n1.Enter college in the list\n");
        printf("2.Delete college from list\n");
        printf("3.Display the list\n");
        printf("4.Exit\n");
        printf("Enter your choice:");
        scanf("%d",&ch);
        if(ch==1){
            if(start == NULL)
            {
                start = (Node*)malloc(sizeof(Node));
                printf("\nEnter College Id :");
                scanf("%d",&start->colID);
                printf("Enter College name :");
                scanf("%s",start->name);
                start->next = start;
            }else{
                int colId;
                char name[20];
                printf("\nEnter College Id :");
                scanf("%d",&colId);
                printf("Enter College name :");
```

```
                scanf("%s",name);
                new_ele(&start,colId,name);
            }
        }
        else if(ch==2){
            del_ele(&start);
        }
        else if(ch==3){
            print_ll(&start);
        }
        else if(ch==4){
            break;
        }
        else{
            printf("you entered wrong choice");
        }
    }
    return 0;
}
```

OUTPUT

```
1.Enter college in the list
2.Delete college from list
3.Display the list
4.Exit
Enter your choice:2
No Data Found

1.Enter college in the list
2.Delete college from list
3.Display the list
4.Exit
Enter your choice:3
Empty List

1.Enter college in the list
2.Delete college from list
3.Display the list
4.Exit
Enter your choice:1

Enter College Id :101
Enter College name :MAIT

1.Enter college in the list
2.Delete college from list
3.Display the list
4.Exit
Enter your choice:1

Enter College Id :102
Enter College name :MAIMS

1.Enter college in the list
2.Delete college from list
3.Display the list
4.Exit
Enter your choice:3

Printing the Employee list
Collegeee Id :102
College Name :MAIMS
----------
Collegeee Id :101
College Name :MAIT
----------
```

```
1.Enter college in the list
2.Delete college from list
3.Display the list
4.Exit
Enter your choice:2

1.Enter college in the list
2.Delete college from list
3.Display the list
4.Exit
Enter your choice:3

Printing the Employee list
Collegeee Id :102
College Name :MAIMS
----------

1.Enter college in the list
2.Delete college from list
3.Display the list
4.Exit
Enter your choice:2

1.Enter college in the list
2.Delete college from list
3.Display the list
4.Exit
Enter your choice:3
Empty List

1.Enter college in the list
2.Delete college from list
3.Display the list
4.Exit
Enter your choice:4
PS D:\C\circular linked list>
```

PROGRAM-11

AIM

Create a Binary Tree and perform Tree Traversals (Preorder, Postorder, Inorder) using the concept of recursion.

THEORY

In **preorder traversal**, first, root node is visited, then left sub-tree and after that right sub-tree is visited. The process of preorder traversal can be represented as –

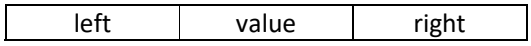
root → left → right

The **postorder traversal** is one of the traversing techniques used for visiting the node in the tree. It follows the principle

Left → right → root.

An **inorder traversal** first visits the left child (including its entire subtree), then visits the node, and finally visits the right child (including its entire subtree).

Left → Root → Right



CODE

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

struct Node{
    struct Node* left;
    int value;
    struct Node* right;
};
typedef struct Node node;
void insertion(int data,node** top){

    if((*top)==NULL){
        node* new;
        new=(node*)malloc(sizeof(node));
        new->left=NULL;
        new->right=NULL;
        new->value=data;
        (*top)=new;
        return;
    }
    if(data==(*top)->value){
        printf("same number cannot be inserted\n");
        return;
    }
    if(data>(*top)->value){
        return insertion(data,&(*top)->right);
    }
    else{
```

```
        return insertion(data,&(*top)->left);
    }
}
void print(node* root,int space){
    int COUNT=3;
    space += COUNT;
    if (root == NULL)
        return;
    print(root->right, space);
    printf("\n");
    for (int i = COUNT; i < space; i++)
        printf(" ");
    printf("%d\n", root->value);
    print(root->left, space);
}
void preorder(node* root){
    if(root!=NULL){
        printf("%d ",root->value);
        preorder(root->left);
        preorder(root->right);
    }
}
void inorder(node* root){
    if(root!=NULL){
        inorder(root->left);
        printf("%d ",root->value);
        inorder(root->right);
    }
}

void postorder(node* root){
    if(root!=NULL){
        postorder(root->left);
        postorder(root->right);
        printf("%d ",root->value);
    }
}

int main(){
    printf("First enter the node\n");
    node* root =NULL;
    char ch='y';
    while(ch=='y'){
        int n;
        printf("Enter data :");
        scanf("%d",&n);
        insertion(n,&root);
        printf("Do yo want to enter more data(y,n):");
        scanf(" %c",&ch);
    }
    print(root,0);
    while (true)
    {
        printf("\nPress 1 to print Pre-Order Traversal");
```

```
        printf("\nPress 2 to print In-order Traversal");
        printf("\nPress 3 to print Post-Order Traversal");
        printf("\nPress 4 to Exit\n");
        int c;
        printf("Enter your choice:");
        scanf("%d",&c);
        if (c==1)
        {
            preorder(root);
        }
        else if(c==2){
            inorder(root);
        }
        else if(c==3){
            postorder(root);
        }
        else if(c==4){
            break;
        }
        else{
            printf("\nyou entered a wrong choice\n");
        }
    }
    return 0;
}
```

OUTPUT

```
First enter the node
Enter data :7
Do yo want to enter more data(y,n):y
Enter data :9
Do yo want to enter more data(y,n):y
Enter data :8
Do yo want to enter more data(y,n):y
Enter data :12
Do yo want to enter more data(y,n):y
Enter data :10
Do yo want to enter more data(y,n):y
Enter data :3
Do yo want to enter more data(y,n):y
Enter data :4
Do yo want to enter more data(y,n):y
Enter data :2
Do yo want to enter more data(y,n):y
Enter data :3
same number cannot be inserted
Do yo want to enter more data(y,n):n

      12

         10

   9

      8

7

      4

   3

      2
```

```
Press 1 to print Pre-Order Traversal
Press 2 to print In-order Traversal
Press 3 to print Post-Order Traversal
Press 4 to Exit
Enter your choice:1
Press 4 to Exit
Enter your choice:2
Press 4 to Exit
Press 4 to Exit
Enter your choice:2
2 3 4 7 8 9 10 12
Press 1 to print Pre-Order Traversal
Press 2 to print In-order Traversal
Press 3 to print Post-Order Traversal
Press 4 to Exit
Enter your choice:3
2 4 3 8 10 12 9 7
Press 1 to print Pre-Order Traversal
Press 2 to print In-order Traversal
Press 3 to print Post-Order Traversal
Press 4 to Exit
Enter your choice:4
PS D:\C\tree>
```

PROGRAM-12

AIM

Implement insertion, deletion, and display (Inorder, Preorder, Postorder) on binary search tree with the information in the tree about the details of an automobile (type, company, year of make).

THEORY

In **preorder traversal**, first, root node is visited, then left sub-tree and after that right sub-tree is visited. The process of preorder traversal can be represented as –

root → left → right

The **postorder traversal** is one of the traversing techniques used for visiting the node in the tree. It follows the principle

Left → right → root.

An **inorder traversal** first visits the left child (including its entire subtree), then visits the node, and finally visits the right child (including its entire subtree).

Left → Root → Right

left	type	company	year	right

CODE

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

struct node{
    struct node* left;
    char type[20];
    char company[20];
    int year;
    struct node* right;
};

typedef struct node Node;

void preorder(Node* root){
    if(root!=NULL){
        printf("%s %s %d\n",root->type,root->company,root->year);
        preorder(root->left);
        preorder(root->right);
    }
}
void inorder(Node* root){
    if(root!=NULL){
        inorder(root->left);
        printf("%s %s %d\n",root->type,root->company,root->year);
        inorder(root->right);
    }
```

```
}

void postorder(Node* root){
    if(root!=NULL){
        postorder(root->left);
        postorder(root->right);
        printf("%s %s %d\n",root->type,root->company,root->year);
    }
}
void deleteit(Node** root){
    if((*root)->right==NULL && (*root)->left==NULL){
        (*root)=NULL;
        return;
    }
    else if((*root)->right==NULL || (*root)->left==NULL){
        if((*root)->right==NULL){
            (*root)=(*root)->left;
        }
        else{
            (*root)=(*root)->right;
        }
        return;
    }
    else{
        Node* temp=(*root)->right;
        if(temp->left==NULL){
            Node* left=(*root)->left;
            (*root)=(*root)->right;
            (*root)->left=left;
            return;
        }
        while(temp->left->left!=NULL){
            temp=temp->left;
        }
        Node* right=(*root)->right;
        Node* left=(*root)->left;
        (*root)=temp->left;
        temp->left=NULL;
        (*root)->right=right;
        (*root)->left=left;

    }
}
void delete(Node** root,int y){
     if ((*root)==NULL)
    {
        printf("No data found");
        return;
    }
    else if (y < (*root)->year)
    {
        delete(&(*root)->left,y);
    }else if(y > (*root)->year){
        delete(&(*root)->right,y);
```

```
    }else{
        deleteit(&(*root));
        return;
    }
}
void insert(Node** root,char* type,char* comp,int y){
    if ((*root)==NULL)
    {
        (*root) = (Node*)malloc(sizeof(Node));
        strcpy((*root)->type,type);
        strcpy((*root)->company,comp);
        (*root)->year = y;
        (*root)->left = NULL;
        (*root)->right = NULL;
        return;
    }
    else if (y < (*root)->year)
    {
        insert(&(*root)->left,type,comp,y);
    }else if(y > (*root)->year){
        insert(&(*root)->right,type,comp,y);
    }else{
        printf("same number cannot be added");
        return;
    }
}

int main(){
    Node* root =NULL;
    char ch = 'y';
    printf("Enter the data of automobile\n");
    while(ch=='y'){
        int y;
        char type[20];
        char company[20];
        printf("Enter type :");
        scanf("%s",type);
        printf("Enter company :");
        scanf("%s",company);
        printf("Enter year :");
        scanf("%d",&y);
        insert(&root,type,company,y);
        printf("DO you want to enter more(y/n):");
        scanf(" %c",&ch);

    }

    while (true)
    {
        printf("\nPress 1 to print Pre-Order Traversal");
        printf("\nPress 2 to print In-order Traversal");
        printf("\nPress 3 to print Post-Order Traversal");
        printf("\npress 4 to delete the node");
        printf("\nPress 5 to Exit\n");
```

```
        int c;
        scanf("%d",&c);
        if (c==1)
        {
            preorder(root);
        }
        else if(c==2){
            inorder(root);
        }
        else if(c==3){
            postorder(root);
        }
        else if(c==4){
            int y;
            printf("Enter the year which you want to delete:");
            scanf("%d",&y);
            delete(&root,y);
        }
        else if(c==5){
            break;
        }
        else{
            printf("\nyou entered a wrong choice\n");
        }

    }
    return 0;


}
```

OUTPUT

```
Enter the data of automobile
Enter type :SUV
Enter company :Toyota
Enter year :2017
DO you want to enter more(y/n):y
Enter type :Sedan
Enter company :KIA
Enter year :2019
DO you want to enter more(y/n):y
Enter type :Sports-car
Enter company :Lamborghini
Enter year :2015
DO you want to enter more(y/n):y
Enter type :Coupe
Enter company :Audi
Enter year :2021
DO you want to enter more(y/n):y
Enter type :CUV
Enter company :Ford
Enter year :2018
DO you want to enter more(y/n):y
Enter type :MPV
Enter company :Maruti
Enter year :2016
DO you want to enter more(y/n):n
```

```
Press 1 to print Pre-Order Traversal
Press 2 to print In-order Traversal
Press 3 to print Post-Order Traversal
press 4 to delete the node
Press 5 to Exit
1
SUV Toyota 2017
Sports-car Lamborghini 2015
MPV Maruti 2016
Sedan KIA 2019
CUV Ford 2018
Coupe Audi 2021

Press 1 to print Pre-Order Traversal
Press 2 to print In-order Traversal
Press 3 to print Post-Order Traversal
press 4 to delete the node
Press 5 to Exit
2
Sports-car Lamborghini 2015
MPV Maruti 2016
SUV Toyota 2017
CUV Ford 2018
Sedan KIA 2019
Coupe Audi 2021

Press 1 to print Pre-Order Traversal
Press 2 to print In-order Traversal
Press 3 to print Post-Order Traversal
press 4 to delete the node
Press 5 to Exit
3
MPV Maruti 2016
Sports-car Lamborghini 2015
CUV Ford 2018
Coupe Audi 2021
Sedan KIA 2019
SUV Toyota 2017
```

```
Press 1 to print Pre-Order Traversal
Press 2 to print In-order Traversal
Press 3 to print Post-Order Traversal
press 4 to delete the node
Press 5 to Exit
4
Enter the year which you want to delete:2018

Press 1 to print Pre-Order Traversal
Press 2 to print In-order Traversal
Press 3 to print Post-Order Traversal
press 4 to delete the node
Press 5 to Exit
1
SUV Toyota 2017
Sports-car Lamborghini 2015
MPV Maruti 2016
Sedan KIA 2019
Sports-car Lamborghini 2015
MPV Maruti 2016
Sedan KIA 2019
press 4 to delete the node
Press 5 to Exit
3
MPV Maruti 2016
Sports-car Lamborghini 2015
Coupe Audi 2021
Sedan KIA 2019
SUV Toyota 2017

Press 1 to print Pre-Order Traversal
Press 2 to print In-order Traversal
Press 3 to print Post-Order Traversal
press 4 to delete the node
Press 5 to Exit
5
PS D:\C\tree>
```