Advanced DSA lab

Name : Deepankar Sharma
Course : MCA
Student ID - 23351203

Experiment 01

→ C program to find min/max in an array.

Algorithm

1. Start
2. min ← arr[0], max ← arr[0]
3. Traverse from first to last element of the array.
4. For each item compare it with min & max
   - if arr[i] < min, min ← arr[i]
   - if arr[i] > max, max ← arr[i]

code

```
#include <stdio.h>
int main() {
    int n, i; scanf("%d", &n);
    int arr[n];
    for (int i=0; i<n; i++)
        scanf("%d", &arr[i]);
    int min = arr[0]; int max = arr[0];
    for (int i=1; i<n; i++) {
        if (arr[i]<min) min = arr[i];
        if (arr[i]>max) max = arr[i];
    printf("min = %d, max = %d", min, max);
    return 0;
}
```

output

```
5
1 2 3 4 5
min = 1, max = 5
```

Assignment-2

Stack using array.

Algorithm

1. Start
2. Initialize top to -1
3. Implement function : push, pop, display
   - push : if stack is full, overflow
     else add element, top += 1
   - pop : if stack is empty, remove element
     decrement top
   - display : print all element from top to bottom
4. stop

Code

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 10

int stack[MAX];
int top = -1;

void push (int val){
      if (top == MAX-1) printf("stack overflow");
      else { top+=1; stack [top] = val; }
}
void pop () { if (top == -1) prinf("stack underflow");
   else { int n = stack [top]; top--;
      printf ("%d ", n);
   }
}
void display () { if (top==-1) printf ("Empty");
      for ( int i = top; i>=0; i--)
            printf ("%d \t ", stack[i]);
}
```

```
int main() {
    push(1);
    push(2);
    push(3);
    pop();
    display(); return 0;
}
```

Sample output

2 1

Experiment 03

demonstrate queue operation using arrays

Algorithm

1. start
2. Rear ← -1 , front = -1
3. Implement:
   - Enqueue: if queue not full, rear+=1, add element
   - Dequeue : if queue not empty, front +=1
   - display : display element from front to rear

Code

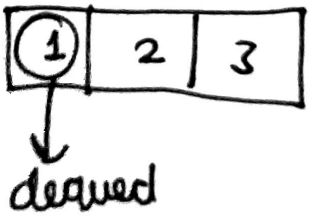
```
#include <stdio.h>
#define MAX 10
int queue[MAX];
int front = -1, rear = -1;
void enqueue(int val) {
        if (rear == MAX-1) print("Queue Overflow");
        else { if(front == -1) front++; rear++;
            queue[rear] = val; } }
```

```
void dequeue(){
    if(front==-1 || front>rear)
        printf("queue underflow");
    else{
        printf("dequeued %d", queue[front]);
        front++;
    }
}
void display(){
    if(front==-1){ printf("queue is empty"); }
    else{
        for(int i=front; i<=rear; i++){
            printf("%d, \t", &queue[i]);
        }
    }
}
int main(){
    enqueue(1); enqueue(2); enqueue(3);
    dequeue();
    display(); return 0;
}
```

Sample Output

dequed 1

2, 3



Experiment 04

C program to demonstrate following operations on a linked list.

→ creation of list
→ adding an element at beginning of the list
→ adding an element at the end of the list
→ delete first element
→ delete last element

Algorithm

① Start
② Implement functions for Linked List operations.
③ Perform the linked list operations
④ Stop

Code

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

struct Node {
    int data;
    struct Node * next;
};      struct Node * head = NULL;

void addAtBeginning (int data) {
    struct Node * newNode = (struct Node *) malloc ( sizeof (
                                                struct Node));
    if (newNode == NULL) return;
    newNode -> data = data;
    newNode -> next = head; head = newNode;
}
```

```
void addAtEnd(int data) {
    struct Node *newNode = (struct Node*)malloc(
                           sizeof(struct Node*));
    struct Node* temp = head;
    if (newNode == NULL) return;
    newNode->data = data; newNode->next = NULL;
    if (head == NULL) {
        head = newNode; return;
    }
    while (temp->next != NULL) temp = temp->next;
    temp->next = newNode;
}

void deleteFirst() {
    struct Node *temp = head;
    if (head == NULL) return;
    head = head->next;
    free(temp);
}

void deleteLast() {
    struct Node* temp = head;
    struct Node* prevNode;
    if (head == NULL) return;
    if (head->next == NULL) { free(head); head = NULL;
        return;
    }
    while (temp->next != NULL) { prevNode = temp;
        temp = temp->next;
    } prevNode->next = NULL; free(temp);
}
```

```
    void displayList() {
        struct Node* temp = head;
        if (head == NULL) return;
        while (temp != NULL) { printf("%d ", temp->data);
            temp = temp -> next;
        } printf("\n");
    }
int main() {
    addAtBeginning(1);
    addAtBeginning(2);
    addAtEnd(4);
    addAtEnd(5);
    displayList();
    deleteFist();
    displayList();
    deleteLast();
    displayList();

    return 0;
}
```

Sample Output

```
2 1 4 5
1 4 5
1 4
```

Experiment 05

Program to create a binary tree & perform tree traversal

Algorithm

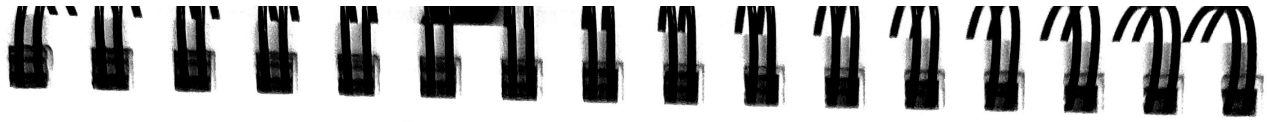
1. Start
2. Implement functions for binary tree operations
   - → create tree
   - → Pre order traversal
   - → In order traversal
   - → Post order traversal
3. Stop

Code

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

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

struct Node * createNode (int data) {
    struct Node * new_node = (struct Node *) malloc (
                sizeof (struct Node));
    if (! new_node) { return NULL; }
    new_node → data = data;
    new_node → left = new_node → right = NULL;
    return new_node; }
```



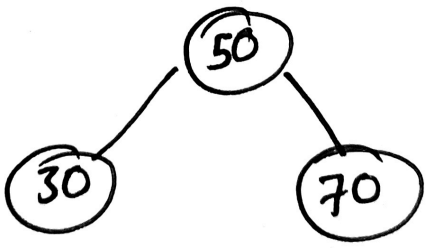
```
struct Node * insertNode (struct Node* root, int data)
{   if (root == NULL){
        root = createNode(data);
        return root;
    }
    if (data < root->data){
        root->left = insertNode(root->left, data);
     } else {
        root->right = insertNode(root->right,
                   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

void preorder(struct Node * root){
      if (root==NULL) return;
      printf("%d  ", root->data);
      preorder(root->left); preorder(root->right);
 }
void inorder(struct Node * root) {
      if (root == NULL) return;
      inorder(root->left);
      printf("%d  ", root->data);
      inorder( root->right;
}
void postorder ( struct Node * root) {
      if (root == NULL) return;
      postorder(root->left); postorder(root->right);
      printf("%d  ", root->data); }
```

```
int main() {
        struct Node * root = NULL;
root = insertNode(50);
root = insertNode(30);
root = insertNode(70);
        inorder(root);
        preorder(root);
        postorder(root);
        return 0;
}
```

Sample Output

```
30  50  70   # inorder
50  30  70   # preorder
30  70  50   # post order
```



Experiment 06

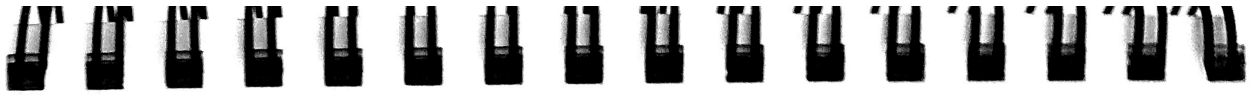
Program to implement the Quick sort in C.

Algorithm

① start
② Implement functions for quick sort
→ partition of the array
→ recursively sort the subarrays
③ Call quicksort() & sort array
④ stop

Code

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



```
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi-1);
        quickSort(arr, pi+1, high);
    }
}
void printArray(int arr[], int size) {
    for (int i=0; i<size; i++){
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main(){
    int n = 5;
    int arr[] = {5, 2, 9, 1, 5}
    printf("Original Array: "); printArray(arr, n);
    quickSort(arr, 0, n);
    printf("Sorted Array: "); printArray(arr, n);
    return 0;
}
```

Sample Output

Original Array: 5 2 9 1 5

Sorted Array: 1 2 5 5 9

Experiment 07

C program to implement linear search.

Algorithm

1. start
2. Traverse the array, if element is found, return index
3. stop

Code

```
#include <stdio.h>
int linearSearch (int arr[], int n, int x) {
    for (int i = 0; i < n; i++) { if (arr[i] == x) return i;
    return -1;
}
int main () {
    int arr[] = {1, 2, 3, 4, 5};
    int x = 3;
    int n = 5;
    int ls = linearSearch (arr, n, x);
    if (ls == -1) {
        printf ("Element not found !");}
    else printf ("Element %d found at index %d", x, ls);
    return 0;
}
```

Sample output

Element 3 found at index 2.

Experiment 08

program to implement the binary search.

Algorithm

1. start
2. sort the array (essential for binary search)
3. Implement Binary search
4. stop

Code

```
#include <stdio.h>
int binarySearch(int arr[], int l, int r, int x) {
    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 arr[] = {1,2,3,4,5};
    int x = 7; int n=5;
    int ds = binary search(arr, 0, n-1, x);
    if (ds == -1) printf(" Element Not present!");
    else printf("%d", ds);
    return 0;
}
```

Sample Output

Element Not present!

Experiment 9

Implement DFS and BFS using a graph

Algorithm

1. Start
2. Implement function using stack/recursion for DFS & queue using BFS.
3. call functions & print the traversal
4. stop.

Code

```
#include<stdio.h>
#define MAX 4
int adj[MAX][MAX];
int visited[MAX];
int n;

void DFS(int v){
    printf("%d", v); visited[v] = 1;
    for(int i=0; i<n; i+=1){
        if(adj[v][i]==1 && !visited[i])
            DFS(i);
    }
}

void BFS(int v){
    int queue[MAX], front = -1, rear = -1
    printf("%d ", v); visited[v] = 1; queue[++rear] = v;
    while(front != rear){
        v = queue[++front];
        for(int
```

```
    for(int i=0; i<n; i++) {
        if (adj[v][i] == 1 && !visited[i]) {
            queue[++rear] = i;
            visited[i] = 1;
            printf("%d ", i);
        }}}}

int main() {
    n = 4;  int v = 0;  // starting vertex
    adj = [ [0, 1, 1, 0],
            [1, 0, 0, 1],
            [1, 0, 0, 1],
            [0, 1, 1, 0] ];

    printf("DFS Traversal:"); DFS(v);
    printf("BFS Traversal:"); BFS(v);

    return 0;
}
```

Sample Output

DFS Traversal: 0132
BFS Traversal: 0123

Experiment 10

Shortest path in the graph using Floyd Warshall Algorithm

Algorithm

1. Start
2. Implement floyd Warshall Algorithm
3. print shortest distance matrix
4. Stop

Code

```
#include <stdio.h>
#include <stdlib.h>
#define INF 99999;
#define MAX 10

void floydWarshall(int graph[MAX][MAX], int n) {
        int dist[MAX][MAX];
    for(int i=0; i<n; i++) {
        for(int j=0; j<n; j++) {
            dist[i][j] = graph[i][j];
        }
    }
    for(int k=0; k<n; k++) {
        for(int i=0; i<n; i++) {
            for(int j=0; j<n; j++) {
                if(dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }
```

```
    printf("shortest distances between every pair of
            vertices: \n");
    for (int i=0; i<n; i++){
        for (int j=0; j<n; j++){
            if (dist[i][j]==INF){
                printf("%7s", "INF");
            }else{ printf("%7d", dist[i][j]);}
        } printf("\n");
    }
}
int main(){
    int n; int graph[MAX][MAX];
    scanf("%d", &n);
    for (int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            scanf("%d", &graph[i][j]);
        }
    }
    floydWarshall(graph, n);
    return 0;
}
```

Sample Output

```
4
0  3  99999  7
8  0  2      99999
5  99999  0  1
2  99999  99999  0
```

shortest distance between every pair of vertices:

0	3	5	6
5	0	2	3
3	6	0	1
2	5	7	0