Aim :- write a Program to Perform Following operation on array

i) Find out largest number and find it's location

ii) a) Insert an element in an array
b) Delete an element from array

iii) Find out sum and average of data element from an array

Bhagwati
Page No.
Date 18/1/24

Practical No. 1

Aim :- write a program to Perform Following operation on array.

i) Find out largest number and find it's location.

```
//C Program to find maximum in arr[] of size n

# include <stdio.h>

// function to find maximum in arr[] of size n int largest
(int arr[]; int n)
{
    int i;

// Initialize maximum element
int max = arr[0];
// Transverse array elements from second and
// compare every element with current max.
```

```
for(i=1; i<n; i++)
    if (arr[i] > max]
            max = arr[i]
return max;
}
    int main()
{
  int arr[] = {10, 324, 45, 90, 9808};
int n = size of (arr) / size of
                        (arr[0]);

    // Function call
  Printf ("Largest in given array
        is %d" largest (arr, n));

  return 0;
}
```

output :-
largest in given array is 9808

Bhagwati
Page No.
Date: / /

2) a) Insert an element in an array
b) delete an element from array

```
# include <stdio.h>
int insert(int arr[], int n, int key,
           int capacity)
{
   if (n >= capacity)
      return n;

 int i;
 for (i = n-1; (i>0 && arr[i] > key;
                  i--)
   arr[i+1] = arr[i];

   arr[i+1] = key;
   return (n+1);
}
   int arr[20] = {12,16,20,40,50,70};
 int capacity = sizeof(arr)/sizeof
                  (arr[0]);
 int n=6;
 int i, key = 26;
```

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

n = insert (arr, n, key, capacity);

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

return 0;
}
```

output :-

Before Insertion : 12 16 20 40 50 70

After Insertion : 12 16 20 26 40 50 70

Bhagwati
Page No.
Date: / /

3) Find the sum and average of data elements from an array.

```
#include <stdio.h>
int main()
{
   int size;
printf("Enter size of the array:");
scanf("%d", &size);

int arr[size];
Printf("Enter array elements \n");

From (int i=0; i<size; i++)
scanf("%d", &arr[i]);
int sum = 0;

for (int i=0; i<size; j++)
sum+ = arr[i];

Printf("Sum of the array is:
                %d", sum)
return 0;
}
```

output:-

```
Enter size of the array : 3
Enter array element
1
2
3
Sum of the array is : 6

Average of the array is : 2
```

output :-

Array : 2 -1 5 6 0 -3

Conclusion :- Thus, we have write program of array to find largest no. and it's location, insertion and deletion of element and to find sum and average of date element in array

Bhagwati
Page No.
Date: / /

• Traverse of array ;

```
#include <stdio.h>
void Print Array (int* arr, int n)
{
   int i ;
   Printf ("Array :") ;
   for (i = 0 ; i<n ; i++)
   {
      Printf ("%d", arr [i]) ;
   }
      Printf ("\n") ;
}
   int main ()
{
   int arr[] = {2, -1, 5, 6, 0, -3} ;
int n = size of (arr) / size of (arr[0]);

   Print Array (arr, n) ;
   return 0 ;
}
```

Conclusion :- Thus, we have write Program of array to find largest no. and it's location, insertion and deletion of element and to find sum and average of date element in array

Practical No. 2

Aim :- WAP

a) To study and execute the linear search method.

b) To study & execute the Binary search method.

Practical No. 2.

Bhagwati
Page No.
Date: / /

Aim :- WAP

a) To study and execute the linear search method.
b) To study and execute the Binary search method.

Program Code :-

a)

```
#include<stdio.h>
int linear search (int a[], int n, int val)
{
    for (int i=0; i<n; i++)
    {
        if (a[i] = = val)
        return i+1;
    }
    return -1;
}
```

```
int main()
int a[] = {70,40,30,11,57,41,25,14,52};
          //given array.
int val = 41; //value to be searched
int n sizeof(a)/sizeof(a[0]);//size of array;
int res = linearSearch(a,n,val); // store reswt
Printf("The elements of the array are-");
for (int i=0; i<n ; i++)

Printf("\n Element to be searched is
                              %d" val);

if(res==-1)
Printf ("\n Element is not present in
                        the array");
else
     Printf("\n Element is not
                  present in the array");

     return ();
}
```

O/P :-
The elements of the array - 70, 40, 30, 11, 57, 41, 25, 14, 52
Element to be searched is 41
Element is present at 6 position of array

o/p :-

The element of the array are :
70 40 30 11 57 41 25 14 52
Element to be searched is -41
Element is present at 6 position of array.

b).

```
#include <stdio.h>
int binary search (int a[], int beg,
                  int end, int val)
int mid;
if (end >= beg)
{
   mid = (beg + end) / 2;
/* if the item to be searched is
   present at middle */
if (a [mid] == val)
   {
      return mid+1;
   }
/* if the item to be searched is
greater than middle, then it can
only be in rights array */
else
      {
         return binary search (a, beg,
                               mid-1, val)
      }
   }
      return -1;
}
```

```
int main() {
int a[] = {11,14,25,30,40,41,52,57,70};
  // given array
int val = 40; //value to be searched
int n = sizeof(a)/sizeof(a[0]); // size of array
int res = binarysearch(a,0,n-1,val); // store result
Printf("The Elements of the array are -");

for (int i=0; i<n ; i++)
Printf ("%d", a[i]);
Printf ("\n Element to be searched is
          %d", val);
if (ress == -1)
Printf ("\n Element is not present in
          the array");
else
    Printf("\n Element is present at
          %d position of array", res);

    return 0;

}
```

o/p :-
The element of the array are - 11, 14, 25, 30, 40, 41, 52, 57, 70.

Element to be searched is 41
Element is present at 6th position of array.

Conclusion :-

Thus we have write program of array and execute a program of linear search method & binary search method.

Conclusion :-

Thus we have write program of array and execute a program of linear search method & binary search method.

Practical No 3.

Aim:- WAP to study and execude Bubble Sort. method.

Practical No. 3

Aim:- WAP to study and execute Bubble Sort method.

Theory :-

Program code :

```
#include<stdio.h>
void main()
{
  int a[10]={54,36,83,46,94,47,58,91,
                                11,44};
  int i,j,temp;
  for(i=0;i<9;i++)
    {
      for(j=0;j<(9-1);j++)
        {
          if(a[j]<a[j+1]
          {
            temp = a[j];
            a[j] = a[j+1];
            a[j+1] = temp
          }
        }
    }
```

output :-

Sorted Element of the array are :

Element 0 is : 94
Element 1 is : 91
Element 2 is : 83
Element 3 is : 58
Element 4 is : 54
Element 5 is : 47
Element 6 is : 46
Element 7 is : 44
Element 8 is : 36
Element 9 is : 11

- Conclusion :-

  Thus we studied and execute the Bubble sort method.

Bhagwati
Page No.
Date: / /

```
Printf("Sorted element of the array
                are: \n");

for (i=0; i<10; i++)
Printf("Element %d is: %d \n",
                i, a[i]);
}
```

- conclusion :-

  Thus we studied and execute the Bubble sort method.

Practical No. 4

Aim:- To study and implement various operation on single linked list

(a) Traversing link list
(b) Insert a node at front of linked list
(c) Searching a link list.

Practical No. 4

Bhagwati
Page No.
Date: / /

Aim: To study and implement various operation on singly linked list.

(a) Traversing link list
(b) Insert a node at front of linked list
(c) Searching a link list.

(A) Traversing link list.

Program:-

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

// Define the structure for a node in the linked list

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

```
// function to traverse the linked list
void traverseList (struct Node* head ) {
    struct Node* temp = head;
    while (temp != Null ){
        printf(" %d ", temp->data);
        temp = temp->next;
    }
}
// Function to insert a node at the
front of the linked list
void insertAtFront(struct Node** head,
    int newData) {
    // Allocate memory for the
    new node
    struct Node* newNode = (struct Node*)
    malloc(sizeof(struct Node));

    // Insert the data and set the next
    pointer to the current head

    newNode->data = newData;
    newNode->next = *head;
```

```
// update the head to point to the
new node
    *head = newNode;
}
// Function to search for a node
in the linked list
struct Node* searchList(struct Node*
head, int key){
    struct Node* current = head;
// Traverse the list to find the key
    while (current != NULL){
        if (current->data == key){
            return current; // key found
        }
        current = current->next;
    }
    return NULL; // key not found.
}
```

Bhagwati
Page No.
Date: / /

```
// main function to test the operations
int main(){
    struct Node* head = NULL;
    struct Node* result;

    // Insert some nodes at the
       front
    insertAtFront(&head, 40);
    insertAtFront(&head, 20);
    insertAtFront(&head, 30);

// Traverse and print the linked
   list

traverseList(head);

return 0;
}
```

Output :-

30 20 40.

Output :-

30 20 40.

(B) Insert a node at front of linked list.

Program:-

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

struct node {
        int data;
        struct node *next;
};

struct node *head = NULL
void insertAtFront (int data) {

    struct node *newNode = (struct node*)
        malloc (sizeof (struct node));

    newNode -> data = data;
```

Bhagwati
Page No.
Date: / /

```
    newNode->next = head;

    head = newNode;
}

void display() {
    struct Node *current = head;

 if (head == NULL) {
        printf("List is empty\n");
        return;
 }
    printf
    while (current != NULL) {
        printf("%d", current->data);
        current = current->next;
    }
}
```

Output :-

40 30 20 10

Coulusion :-

- Traversing allows us to go through each node in the list to view or process the data.
- Adding a node at the front quickly makes it the new first element in the list.

int main () {

// insert nodes at the front of the linked list

insertAtFront (10);
insertAtFront (20);
insertAtFront (30);
insertAtFront (40);

display ();
return 0;
}

Output :-

40 30 20 10

Coulusion :-

- Traversing allows us to go through each node in the list to view or process the data.
- Adding a node at the front quickly makes it the new first element in the list.

Practical No. 5

Aim:- To study & implement following operations on the doubly linked list
(a) insert a node at given position of the linked list
(b) Insert a node at given position of the linked

Practical No. 5

Bhagwati
Page No.
Date: / /

Aim:- To study and implement following operations on the doubly linked list.
(a) Insert a node at the front of the linked list.
(b) Insert a node at given position of the linked list.
(c) Delete a node of the linked list.
(d) search a node in a linked list.

Program code:-

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

struct node {
        int data;
        struct node *next;
        struct node *prev;
} *start = NULL;
```

Bhagwati
Page No.
Date: / /

```
struct node *search(int);
void create();
void disp();
void insertafter(struct node*, int);
void insertbeg(int);

int main(){
    int choice, item, ele;
    struct node * t;
while(1){ printf("\n\n***** MENU *****\n1.
 create\n2. Display\n3. search a node
\n4. Insert a node\n5. Exit\n Enter
 your choice:");
   scanf("%d", &choice);
   switch(choice){
     case 1: create(); break;
     case 2: disp(); break;
     case 3:
        printf("Enter element to search");
     scanf("%d", &item);
     t = search(item);
     if(t != NULL){
     printf("Item %d found.\n", item);
```

```
} else {
    printf("Item %od not found.\n", item);
}
  break;
case 4:
      printf("Enter element after which
              to insert:");
      scanf("%od", &ele);
      t = search(ele);
      if (t != NULL) {
    printf("Enter element to insert");
    scanf("%od", &item);
    insertafter(t, item);
  } else { printf("Element %od not found.
    \n", ele);
    }
    break;
  case 5: exit(0);
  }
}
  return 0;
}
```

```
void create(){
    struct node *ptr, *new1;
char ch;
    start = ptr = (struct node*)malloc
(sizeof(struct node));
  if (start == NULL){
    printf("memory allocation failed. \n");
  exit(1);
}
printf("Enter data for first node : ");
scanf("%d", &start->data);
start->prev = NULL;
getchar();
printf("create more nodes (y/n) :");
scanf("%c", &ch);
while(ch == 'y'){
        new1 = (struct node*)malloc
(sizeof(struct node));
    if(new1 == NULL){
        printf("memory allocation failed\n");
    exit(1);
}
```

```
ptr -> data = item;
ptr -> next = t -> next;
if (t -> next != NULL) {
        t -> next -> prev = ptr;
}
t -> next = ptr;
ptr -> prev = t;
}
void insertbeg (int item) {
    struct node *ptr = (struct node *)
malloc (sizeof (struct node));
    if (ptr == NULL) {
        printf ("Memory allocation failed.\n");
        exit (1);
    }
    ptr -> data = item;
    ptr -> next = start;
    ptr -> prev = NULL;
        start -> prev = ptr;
    }
    start = ptr;
    {
```

O/P:-

**** MENU ****
1. Create
2. Display
3. Search a node
4. Insert a node.
5. Exit
Enter your choice :- 1
Enter data for first node :- 55
create more nodes (Y/n) : Y
Enter data for next node : 27
create more nodes (Y/n) :- n

Conclusion:-

A doubly linked list allows efficient bidirectional traversal & manipulation of nodes. Operations like insertion at the front.

O/P:-

**** MENU ****

1. Create
2. Display
3. search a node
4. Insert a node
5. Exit
Enter your choice:- 1
Enter data for first node:- 55
create more nodes (Y/n) : Y
Enter data for next node: 27
create more nodes (Y/n) :- n

Conclusion:-

A doubly linked list allows efficient bidirectional traversal & manipulation of nodes. Operations like insertion at the front.

Practical No. 6

Aim:- Understand the stack structure & execute the push, pop, operation on it

ame of Practical

Practical No. 6

Page No :
Date : / /

Aim:- Understand the stack structure & execute the push, pop, operation on it.

Program:-

```
#include<stdio.h>
#include<stdlib.h>
#define max_size 5
int stack[max_size], top=-1;

void push();
void pop();
void peep();
void display();

int main(){
    int choice;
    do{
        printf("\n\n---- STACK OPERATIONS-\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Peep\n");
```

Teacher's Signature

Page No :
Date : / /

Name of Practical

```
printf("4. Display\n");
printf("5. Exit\n");
printf("--------");
printf("\n Enter your choice: \t");
scanf("%d", &choice);

switch (choice){

case 1:
        push();
        break;
case 2:
        pop();
        break;
case 3:
        peep();
        break;
case4:
        display;
        break;
case5:
        exit(0);
default:
        printf("\n Invalid choice. please try again:\n")
```

Teacher's Signature

Page No :
Date : 1 1

ame of Practical

```
}
}while (choice != 5);
  return 0;
}

void push() {
      int item;
   if (top == (max_size-1)) {
        printf("In Stack Overflow");
   }else{
        printf("Enter the element to
                be inserted:\t");
      scanf("%d", &item);
      top = top + 1;
      stack[top] = item;
      printf("%d pushed into the stack.\n",
              item);
   }
}
```

Teacher's Signature

Page No :
Date : / /

ame of Practical

```
void pop(){
    if(top==-1){
        printf("\n stack Underflow");
    }else{
        int item = stack[top];
        top = top - 1;
        printf("\n The popped element is:
            %d\n", item);
    }
}

void peep(){
    if(top==-1){
        printf("\n stack is empty");
    }else{
        printf("The topmost element of
            the stack is %d\n", stack
            [top]);
    }
}
```

Teacher's Signature

Page No :
Date : 1 1

ame of Practical

```
void display(){
    if(top==-1){
        printf("\n stack is empty");
    }else{
        printf("\n The stack elements
                are:\n");
        for(int i=top; i>=0; i--){
            printf("%d\n", stack[i]);
        }
    }
}
```

O/p

Teacher's Signature

Output :-

1. push
2. pop
3. peep
4. Display
5. Exit

Enter your choice : 1
Enter the element to be inserted : 12
12 pushed into the stack.

Conclusion :-

A stack is a linear data structure that follows the Last In, First Out (LIFO) principle, where elements are added and removed from the same end. This ensures that the most recently added element is the first to be accessed or removed.

Page No :
Date : / /

ame of Practical

Teacher's Signature

Output

```
---- STACK OPERATIONS ----

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

Enter your choice : 2
The popped element is : 555

---- STACK OPERATIONS ----
1. Push
2. Pop
3. Peep
4. Display
5. Exit

Enter your choice : 3
The topmost element of the stack is 99
```

Name of Practical

- - - - STACK OPERATIONS - - -

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

Enter your choice: 4
The stack element are: 66

Conclusion:-

A queue is a linear data structure that follows the First In, First Out (FIFO) principle where elements are added at the back (rear) & removed from the front.

Page No :
Date : / /

ame of Practical

Output:-

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

Enter your choice: 1
Enter the element to be inserted: 12
12 pushed into the stack.

Conclusion:-

A stack is a linear data structure that follows the last In, first Out (LIFO) principle, where elements are added & removed from the same end. This ensures that the most recently added element is the first to be accessed or removed.

21/10/24

Teacher's Signature

Practical No. 7

Aim:- Understand the queue structure & execute the insertion, deletion.

Practical No. 7

Page No :
Date : 1 1

ame of Practical

Aim:- Understand the queue structure & execute the insertion, deletion

program :-

```
#include<stdio.h>
#include<stdlib.h>
#define max_size 5
int queue[max_size], front=-1, rear=-1;

void insert();
void del();
void display();

int main(){
    int choice;
    do{
        printf("\n\n ---- Queue Operations ----\n");
        printf("1. Insert\n");
        printf("2. Deleted\n");
        printf("3. Display\n");
        printf("4. Exit\n");
```

Teacher's Signature

Page No :
Date : / /

Name of Practical

```
printf("----------");
printf("\n Enter your choice:\t");
scanf("%d", &choice);

switch (choice){
    case 1:
        insert();
        break;
    case 2:
        del();
        break;
    case 3:
        display();
        break;
    case 4:
        exit(0);
    default:
        printf("\n Invalid choice. please try
                again.\n");
}
}while(choice != 4);

return 0;
}
```

Teacher's Signature

Page No :
Date : / /

he of Practical

```
Void insert (){
    int item;
    if (rear == (max_size - 1)){
    printf("\n Queue OverFlow\n");
} else{
    printf("Enter the element to be
            inserted:\t");
    scanf("%d", &item);
    if(front == -1)
        front = 0;
    rear = rear + 1;
    queue[rear] = item;
    printf("%d inserted into the queue.\n",
            item);
}
}
```

Teacher's Signature

Page No :
Date : / /

Name of Practical

```
void del(){
    if(front == -1){
        printf("\n Queue Underflow\n");
    }else{
        printf("\n The queue elements are:\n");
        for(int i= front ; i<= rear; i++){
            printf("%d\t", queue[i]);
        }
        printf("\n");
    }
}
```

Teacher's Signature

Output:-

1. insert
2. Deleted
3. Display
4. Exit

Enter your choice: 1
Enter the element to be inserted : 33
33 inserted into the queue.
Enter your choice :- 2
The deleted element is : 33
Enter your choice: 3
Queue is empty
Enter your choice : 4
=== code Executed Successful ===

Conclusion :-

A queue is a linear data structure that follows the First In, First Out (FIFO) principle, where elements are added at the back (rear) & removed from the front.

Page No :
Date : 1 1

Name of Practical

Output :-

1. insert
2. Deleted
3. Display
4. Exit

Enter your choice :- 1
Enter the element to be inserted : 33
33 inserted into the queue.

Conclusion :-

A queue is a linear data structure that follows the First In, First Out (FIFO) principle, where elements are added at the back (rear) & removed from the front.

Teacher's Signature

Practical No. 8

Aim:- Execute the insertion, deletion operation on circular queue.

Name of Practical

Page No :
Date : / /

Practical no. 8

Aim:- Execute the insertion, deletion operation on Circular Queue.

program:-

```
#include<stdio.h>
#include<stdlib.h>
#define max_size 5
int queue[max_size], front = -1, rear = -1;

void insert();
void del();
void display();

int main(){
    int choice;
    do{
        printf("\n\n---CIRCULAR
        queue Operation----\n");
```

Teacher's Signature

Page No :
Date : / /

ame of Practical

```
printf("1. Insert\n");
printf("2. Delete\n");
printf("3. Display\n");
printf("4. Exit\n");
printf("  ------  ");
printf("\n Enter your choice:t");
scanf("%d", &choice);

switch(choice) {
        case 1:
            insert();
            break;
        case 2:
            del();
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
        default:
            printf("\n Invalid choice:\n");
    }
} while (choice != 4);
    return 0;
}
```

Teacher's Signature

Page No :
Date : / /

Name of Practical

```
void insert () {
    int item;
    if ((front == 0 && rear == max_size - 1)
 || (rear + 1) % max_size == front)
 {
    printf ("\n Queue Overflow");
 } else {
    printf ("Enter the element to be
            inserted : \t");
    scanf ("%d", &item);

    if (front == -1){
        front = 0;
        rear = 0;
    } else {
        rear = (rear + 1) % max_size;
    }
    queue [rear] = item;
    printf ("%d inserted into the queue.\n", item);
 }
}
```

Teacher's Signature

Page No :
Date : / /

ame of Practical

```
void del(){
    if(front == -1){
        printf("\nQueue UnderFlow");
    }else{
        int item = cqueue[front];
        printf("\n The deleted element is:
            %d\n", item);

    if(front == rear){
        front = -1;
        rear = -1;
    }else{
        front = (front + 1) % max_size;
    }
  }
}
```

Teacher's Signature

Page No :
Date : / /

ame of Practical

```
void display (){
    if (Fout == -1){
        printf ("\n Queue is empty");
    }else{
        printf ("\n The queue elements
                are:\n");
    int i=Fout;
    if (Fout <= rear){
        while (i<=rear){
            printf ("%d\t", cqueue[i]);
            i++;
        }
    }else{
        while (i < max_size){
            printf ("%d\t", cqueue[i]);
            i++;
        }
        i=0
        while (i<=rear){
            printf ("%d\t", cqueue[i]);
            i++;
        }
    }
    printf ("\n");
}
```

Teacher's Signature

Op:- 1. Insert
2. Delete
3. Display
4. Exit

Enter your choice: 1
Enter the element to be inserted: 44
44 inserted into the queue.

Enter your choice: 2
The deleted element is: 44

Enter your choice: 3
Queue is empty

Enter your choice: 4
=== code Execution Successful ===

Conclusion:-

A circular queue efficiently use memory by connecting the end back to the beginning allowing continuous insertion & deletion without shifting element.

Page No :
Date : / /

Name of Practical

Output:-

---CIRCULAR QUEUE OPERATIONS---

1. Insert
2. Delete
3. Display
4. Exit

Enter your choice: 1
Enter the element to be inserted: 44
44 Inserted into the queue.

Conclusion:-

A circular queue efficiently uses memory by connecting the end back to the beginning, allowing continuous insertion & deletion without shifting elements.

18/11/24

Teacher's Signature

Practical No. 09

Aim :- Understand the Tree structure and implement the pre-order, In-order, post-order traversing operation on it.

Page No :
Date : / /

Practical No. 09.

Name of Practical

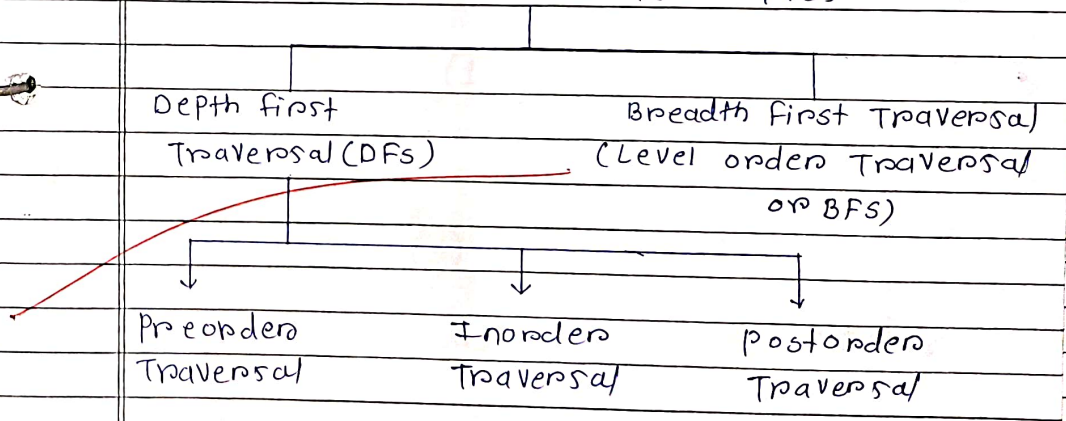
Aim :- Understand the Tree structure and implement the Pre-order, In-order, Post-order traversing operations on it.

Theory :

Tree Traversal meaning :-

Tree Traversal refers to the process of visiting or accessing each node of the tree exactly once in a certain order. There are multiple tree traversal techniques which decide the order in which the nodes of the tree are to be visited.

Tree Traversal Techniques



Teacher's Signature _____

Page No :
Date : \ \

Name of Practical

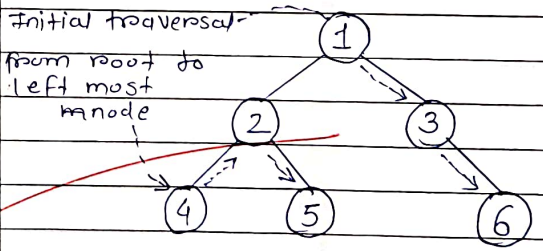
A Tree Data structure can be traversed in following ways :

- Depth First search or DFS
  - Inorder ~~Traversal~~ Traversal
  - Preorder Traversal
  - Postorder Traversal
- Level order Traversal or Breadth First Search or BFS.

Inorder Traversal :

Inorder traversal visits the node in the order : Left → Root → Right.

Inorder Traversal of Binary Tree



Inorder Traversal : 4 → 2 → 5 → 1 → 3 → 6.

Teacher's Signature _____

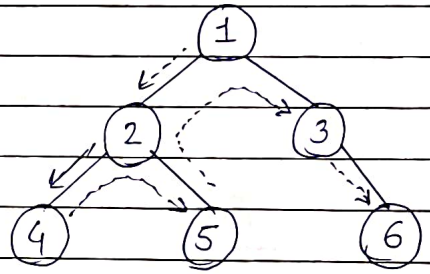
Page No :
Date : \ \

Name of Practical

Preorder Traversal :-

Preorder Traversal visits the node in the order : Root → Left → Right.

Preorder Traversal of Binary Tree



Preorder Traversal : 1 → 2 → 4 → 5 → 3 → 6.

Teacher's Signature _____

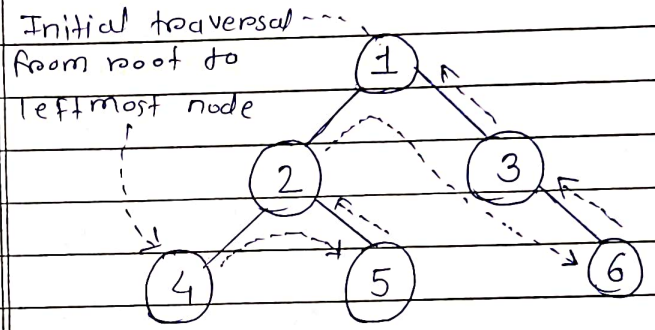
Page No :
Date : \ \

Name of Practical

Postorder Traversal :-

Post order traversal visits the node in the order : Left → Right → Root.

Postorder Traversal of Binary Tree



postorder Traversal : 4 → 5 → 2 → 6 → 3 → 1

Teacher's Signature _____

Page No :
Date : \ \

Name of Practical

Program Code :-

```
#include <stdio.h>
#include <stdlib.h>
struct node {
int element;
struct node* left;
struct node* right;
};
/* To create a new node*/
struct node* createNode(int val)
{
 struct node* Node = (struct node*) malloc(sizeof
                                (struct node));
Node -> element = val;
Node -> left = NULL;
Node -> right = NULL;
return (Node);
}
/* Function to traverse the nodes of
   binary tree in Preorder */

void traversePreorder(struct node* root)
{
```

Teacher's Signature

Page No :
Date : \ \

Name of Practical

```
if (root == NULL)
return;
Printf ("%d", root -> element);
traversePreorder (root -> left);
traversePreorder (root -> right);
}
/* Function to traverse the nodes to binary
    tree in Inorder */
void traverseInorder (struct node* root)
{
  if (root == NULL)
  return;
  traverseInorder (root -> left);

  Printf ("%d", root -> element);
  traversePostorder (struct node* root)
  {
    if (root == NULL)
    return;
  traversePostorder (root -> left);
  traversPostorder (root -> right);
  Printf ("%d", root -> element);
  }
```

Teacher's Signature

o/p :- The Preorder traversal of given binary tree is-

36 26 21 11 24 31 46 41 56 51 66

The Inorder traversal of given binary tree is-

11 21 24 26 31 36 41 46 51 56 66

The Postorder traversal of given binary tree is-

11 24 21 31 26 41 51 66 56 46 36

Page No :
Date : / /

Name of Practical

```
int main()
{
  struct node* root = CreateNode(36);
root->left = CreateNode(26);
root->right = CreateNode(46);
root->left->left = CreateNode(21);
root->left->right = CreateNode(31);
root->left->left->left = CreateNode(11);
root->left->left->right = CreateNode(24);
root->right->left = CreateNode(41);
root->right->right = CreateNode(56);
root->right->right->left = CreateNode(51);
root->right->right->right = CreateNode(66);

Printf("\n The Preorder traversal of
 given binary tree is-\n");
traversePreorder(root);
Printf("\n The Inorder traversal of
      given binary tree is-\n");
traverseInorder(root);
Printf("\n The Postorder traversal of
given binary tree is-\n");
 tar traversePostorder(root);
   return 0;
}
```

Teacher's Signature

conclusion :- The traversals are essential for searching, sorting, and evaluating expression.

Page No :
Date : / /

Name of Practical

conclusion :- These traversals are essential for searching, sorting, and evaluating expression.

11/8/24

Teacher's Signature