Register No: Experiment No: Date:

S. No	Component	Max. Marks	Marks Secured
1	Preparedness	2	
2	Viva-Voce	2	
3	Experiment	3	
4	Analysis & Record	3	
Total		10	
Date		Signature of the Lab teacher	

Experiment - 1

AIM: write a program in c to display the 'n' terms of even natural numbers and their sum.

Program:

```
#include<stdio.h>
void main(){
    int number, evennumber, sum=0;
    printf("enter a number");
    Scanf("%d", &number);
    printf("the even numbers are");
    for(int i=1; i<=number; i++){
        evennumber = i*2;
        printf("%d\n", evennumber);
        sum = sum + evennumber;
    }
```

ister No : Experiment No : Date:

```
printf("the sum of the above even numbers is=
%d", sum);
}
```

Output:

```
Enter a number
10
the even numbers are :
2
4
6
8
10
12
14
16
18
20
the sum of the above even numbers is = 110
```

Register No : Experiment No : Date:

Aim: Write a program in c to display the reverse of a given positive number.

program:

```
#include<stdio.h>
void main(){
    int number, reversenumber=0, remainder;
    printf("enter a number");
    scanf("%d", &number);
    printf("the original number is:%d\n",
                                    number);
    while(number!=0 && number>0){
        remainder = number%10;
        reversenumber = reversenumber*10
                                + remainder;
        number = number/10;
    }
    printf("the reverse number is:%d",
                                reversenumber);
}
```

Output: enter a number

1234567

the original number is: 1234567

the reverse number is: 0

enter a number

0

the original number is: 0

the reverse number is: 0

Register No : Experiment No : Date:

Aim: Write a c program to check whether a given number is an armstrong number (or) not.

program:

```
#include<stdio.h>
void main(){
    int number, originalNumber, count=0,
    result=0, remainder;
    printf("enter a number");
    scanf("%d", &number);
    originalNumber = number;
    while (originalNumber != 0){
        remainder = originalNumber %10;
        result = result + pow(remainder,
                              count);
        originalNumber = originalNumber/10;
    }
    if(result == number){
        printf("the number is an Armst-
                rong number");
    }
    else{
        printf("the number is not an
                Armstrong number");
    }
```

Output:

enter a number

153

the number is an armstrong number

enter a number

123

the number is not an armstrong number

Register No : Experiment No : Date:

Aim: Write a C program to calculate factorial of a given number.

program:

```
#include<stdio.h>
void main(){
    int number, factorialOfNumber=1;
    printf("enter a number");
    scanf("%d", &number);
    if (number > 0){
        for (int i=1; i<=number; i++){
            factorialOfNumber = factorialOfNumber
                                *i;
        }
    printf(" the factorial of a given
            number %d is %d", number,
            factorialOfNumber);
    }
}
```

Output:

enter a number 10

the factorial of a given number 10 is 3628800.

Register No:

Experiment No:

Date:

S. No	Component	Max. Marks	Marks Secured
1	Preparedness	2	
2	Viva-Voce	2	
3	Experiment	3	
4	Analysis & Record	3	
	Total	10	
Date		Signature of the Lab teacher	

Experiment - II

AIM: write a c program to print multiplication of 2 square matrix.

Program:

```
#include<stdio.h>
void main(){
    int arr[2][2], brr[2][2], c[2][2], i,j,
        k;
    printf("enter the elements of matrix1 \n:");
for (i=0; i<2; i++){
    for (j=0; j<2; j++){
        scanf("%d", &arr[i][j]);
    }
}
printf("enter the element of matrix2
                                \n:");
for (i=0; i<2; i++){
```

gister No : Experiment No : Date:

```
for(j=0; j<2; j++){
    scanf("%d", &brr[i][j]);
  }
}

for(i=0; i<2; i++){
    for(j=0; j<2; j++){
        c[i][j]=0;
        for(k=0; k<2; k++){
            c[i][j] = c[i][j] + arr[i][k] *
                                brr[k][j];
        }
    }
}
printf("the product of two matrix is:\n");
for(i=0; i<2; i++){
    for(j=0; j<2; j++){
        printf("%d\t", c[i][j]);
    }
    printf("\n");
}
}
```

Output:

```
enter the elements of matrix 1:
1
2
3
4
enter the elements of matrix 2:
1
2
3
4
the product of two matrix is:
7       10
15      22
```

Register No : | Experiment No : | Date:

Aim: write a c program to print transpose of a given matrix

program:

```
#include<stdio.h>
void main(){
  int a[2][3], i, j;
  printf("enter the elements into an
                    array");
  for(i=0; i<2; i++){
    for(j=0; j<3; j++){
      scanf("%d", &a[i][j]);
    }
  }
  printf("the original entered matrix is:");
  for(i=0; i<2; i++){
    for(j=0; j<3; j++){
      printf("%d\n", a[i][j]);
    }
  }
  printf("after the transposing matrix
                    is:");
  for(i=0; i<3; i++){
    for(j=0; j<2; j++){
      printf("%d\n", a[j][i]);
    }
  }
```

Output:

```
enter the elements into an array
10
20
30
40
50
60
the original entered matrix is: 10
20
30
40
50
60
after the transposing matrix is:
10
40
20
50
30
60
```

Register No: Experiment No: Date:

S. No	Component	Max. Marks	Marks Secured
1	Preparedness	2	
2	Viva-Voce	2	
3	Experiment	3	
4	Analysis & Record	3	
Total		10	
Date		Signature of the Lab teacher	

Experiment- III

a) **AIM:** Write a program in c to check whether a given number is prime number or not using the function.

Program:

```
#include<stdio.h>
int isPrime(int num){
        if (num <=1){
            return 0;
        }
        for(int i=2; i<=num/2; i++){
            if(num%i==0){
                return 0;
            }
        }
        return 1;
}
```

Register No : Experiment No : Date:

```
int main(){
    int number;
    printf("Enter a number:");
    scanf("%d", &number);
  if (isPrime(number)){
        printf("%d is a prime number.\n",
                                    number);
    }
  else {
      printf("%d is not a prime number.\n",
                                    number);
      }
    return 0;
  }
```

Register No : Experiment No : Date:

b) Aim: Write a recursive program which computes the nth fibonacci number, for appropriate values of n.

Program:

```
#include<stdio.h>
int fibonacci(int n){
    if (n<=1){
        return n;
    }
    else{
        return fibonacci(n-1) + fibonacci(n-2);
    }
}
int main(){
    int n;
    printf("Enter the value of n: ");
    scanf("%d", &n);
    if (n<0){
        printf("please enter a non-negative
                integer.\n");
    } else {
        printf("The %dth fibonacci
                number is: %d\n", n,
                fibonacci(n-1));
```

Register No : Experiment No : Date:

```
}
return 0;
}
```

Register No : | Experiment No : | Date:

c) Aim: Write a program in c to add numbers using call by reference.

Program:

```
#include<stdio.h>
void addNumbers(int *a, int *b, int *result){
    *result = *a + *b;
}
int main(){
    int num1, num2, sum;
    printf("Enter the first number:");
    scanf("%d", &num1);
    printf("Enter the second number:");
    scanf("%d", &num2);
    addNumbers(&num1, &num2, &sum);
    printf("Sum of %d and %d is: %d
    \n", num1, num2, sum);
    return 0;
}
```

Register No: Experiment No: Date:

S. No	Component	Max. Marks	Marks Secured
1	Preparedness	2	
2	Viva-Voce	2	
3	Experiment	3	
4	Analysis & Record	3	
Total		10	
Date		Signature of the Lab teacher	

EXPERIMENT-4

9a **AIM:** Write recursive programs for the following

a) Write recursive and non recursive C program for calculation of Factorial of an integer.

Program:

Recursive:

```
#include<stdio.h>
int recursiveFactorial(int n){
        if(n==0 || n==1){
                return 1;
        }
        else{
                return n*recursiveFactorial(n-1);
        }
}
int main(){
        int num;
        printf("Enter a non-negative integer:");
```

Register No : Experiment No : Date:

```
    scanf("%d", &num);
    if (num < 0){
        printf("Factorial is not defined for ne-
                gative numbers.\n");
    }
    else{
        printf("Recursive factorial of %d is :
                %d\n", num, recursiveFact-
                orial(num));
    }
    return 0;
}
```

write only non-Recursive

Non-Recursive Approach :

```
#include <stdio.h>
int nonRecursiveFactorial(int n){
    int factorial = 1;
    for(int i = 1; i <= n; i++){
        factorial *= i;
    }
    return factorial;
}
```

Register No : Experiment No : Date:

```
int main(){
    int num;
    printf("Enter a non-negative integer:");
    scanf("%d", &num);
    if (num<0){
        printf("Factorial is not defined.\n");
    }
    else{
        printf("Non-recursive Factorial of %d is
                %d\n", num, nonRecursiveFact-
                orial(num));
    }
    return 0;
}
```

Register No : Experiment No : Date:

b) Aim: Write a recursive and non-recursive C program for calculation of GCD(n, m)

Program:

```
#include<stdio.h>
int recursiveGCD(int n, int m){
    if(m==0){
        return n;
    }
    else{
        return recursiveGCD(m, n%m);
    }
}
int main(){
    int num1, num2;
    printf("Enter two non-negative integers:");
    scanf("%d %d", &num1, &num2);
    if(num1<0 || num2<0){
        printf("GCD is not defined for
            negative numbers.\n");
    }
    else{
        printf("Recursive GCD of %d is
            %d\n", num1, num2, recursi-
            veGCD(num1, num2));
    }
```

egister No : Experiment No : Date:

```
    return 0;
}
```

Non-Recursive:

```
#include<stdio.h>
int nonRecursiveGCD(int n, int m){
        while(m!=0){
            int temp = m;
            m = n%m;
            n = temp;
        }
        return n;
    }
int main(){
        int num1, num2;
        printf("Enter 2 non-negative integers:");
        scanf("%d %d", &num1, &num2);
        if(num1<0 || num2<0){
            printf("GCD is not defined.\n");
        }else{
            printf("Non-Recursive GCD of %d and
              %d is: %d\n", num1, num2,
                        nonRecursiveGCD(num1,num2));
        }return 0;
    }
```

Register No: Experiment No: Date:

S. No	Component	Max. Marks	Marks Secured
1	Preparedness	2	
2	Viva-Voce	2	
3	Experiment	3	
4	Analysis & Record	3	
Total		10	
Date		Signature of the Lab teacher	

Experiment - 5

AIM: Write a c program that implement stack (it's operations) using arrays.

program:

```
#include<stdio.h>
#include<conio.h>
int stack[5];
int top=-1;
void push(){
    int x;
    printf("Enter data");
    scanf("%d",&x);
    if(top==4){
        printf("Stack Overflow");
    }
    else{
        top++;
        stack[top]=x;
    }
}
```

er No : Experiment No : Date:

```
void pop(){
    int item;
    if(top==-1){
        printf("stack is underflow");
    }
    else{
        item=stack[top];
        top--;
        printf("The poped item is %d", item);
    }
}
void peek(){
    if(top==-1){
        printf("stack is empty");
    }
    else{
        printf("%d", stack[top]);
    }
}
void display(){
    int i;
    for(i=top; i>=0; i--){
        printf("%d", stack[i]);
    }
}
```

Register No : Experiment No : Date:

```
void isfull(){
    if(top == 4){
        printf("stack overflow");
        printf("stack is full");
    }
}
void isempty(){
    if(top == -1){
        printf("stack Underflow");
        printf("Stack is empty");
    }
}
void main(){
    int ch=0;
    do{
        printf("Enter your choice 1:push 2:pop
                3:peek 4. display 5. isfull
                6 : isempty");
        scanf("%d",&ch);
        switch(ch){
            case 1: push();
                    break;
            case 2: pop();
                    break;
            case 3: peek();
                    break;
```

Register No : Experiment No : Date:

```
        case 4:display();
               break;
        case 5: isfull();
               break;
        case 6: isempty();
               break;
        default: printf("Invalid choice");
        }
    }while(ch!=0);
    getch();
    }
```

Register No : Experiment No : Date:

b) Write C program that implement stack (it's operation) using Linked Lists.

program:

```
struct node{
        int data;
        struct node *link;
    };
struct node *top=0;
void push(){
        int x;
        printf("Enter x value");
        scanf("%d", &x);
        struct node *newnode;
        newnode= (struct node*) malloc(sizeof(st-
        newnode->data = x;                    ructnode));
        newnode->link = top;
        top = newnode;
    }
void display(){
        struct node *temp;
        temp= top;
        if (top==0){
            printf("List is empty");
        }
        else {
            while (temp!=0){
                printf("%d", temp->data);
                temp =temp->link;
            }
```

Register No : Experiment No : Date:

```
    }
  }
void peek(){
    if(top==0){
        printf("Stack is empty");
    }
    else{
        printf("top element is %d", top->data);
    }
}
void pop(){
    Struct node *temp;
    temp=top;
    if(top==0){
        printf("Stack is empty");
    }
    else{
        printf("The poped element is %d", top->data);
        top=top->link;
        free(temp);
    }
}
void main(){
    int ch=0;
    do{
        printf("Enter choice 1:push 2:pop
                3:peek 4:display");
        Scanf("%d",&ch);
```

Experiment No : Date:

```
switch(ch){
    case 1: push();
            break;
    case 2: pop();
            break;
    case 3: peek();
            break;
    case 4: display();
            break;
    default: printf("Invalid choice");
    }
}while(ch!=0)
getch();
}
```

Register No:

Experiment No: Date:

S. No	Component	Max. Marks	Marks Secured
1	Preparedness	2	
2	Viva-Voce	2	
3	Experiment	3	
4	Analysis & Record	3	
	Total	10	
Date		Signature of the Lab teacher	

Experiment - 6

AIM: Write a C program that uses stack operations to convert infix expression into postfix expression.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include<string.h>
#include<ctype.h>
#define MAX_Size 100
struct Stack{
        int top;
        unsigned capacity;
        char *array;
};
struct Stack *createStack(unsigned capacity){
        struct Stack * stack=(struct stack*) malloc(sizeof(struct stack));
        stack->capacity=capacity;
        stack->top=-1;
        stack->array=(char *) malloc(stack->capacity * sizeof(char));
        return stack;
}
```

Register No : | Experiment No : | Date:

```
int isFull(Struct stack* stack)
        return stack->top == stack->capacity - 1;
}
int isEmpty(Struct stack *stack) {
        return stack->top == -1;
}
void push(Struct Stack *stack, char item){
        if(isFull(Stack)) {
                printf("Stack Overflow\n");
                return;
        }
    stack->array[++stack->top] = item;
}
char pop(Struct stack *stack) {
        if (isEmpty(stack)) {
                printf("stack Underflow\n");
                return 0;
        }
        return stack->array[stack->top--];
}
char peek(Struct stack *stack) {
        if (isEmpty(stack)) {
                return 0;
        }
```

Register No : Experiment No : Date:

```
    return stack->array[stack->top];
}
int precedence(char op){
        if(op == '+' || op == '-'){
            return 1;
        }else if(op == '*' || op == '/'){
            return 2;
        }
        return 0;
}
char *infixTopostfix(char* infix){
        Struct stack *stack = createStack(MAX_
                                          SIZE);
        int i, K;
        char * postfix = (char *) malloc(strlen(in-
                         fix) * sizeof(char));
        if (!postfix){
            printf("Memory allocation failed\n");
            exit(EXIT_FAILURE);
        }
    for(i=0, K= -1 ; infix[i]; ++i){
        if (isalnum(infix[i])) {
            postfix[++K]= infix[i];
        }else if (infix[i]== '('){
            push(stack, infix[i]);
        }else if (infix[i] == ')'){
```

ister No : Experiment No : Date:

```
while(! isEmpty(stack) && peek(stack)!='('){
        postfix[++K] = pop(stack);
    }
if(! isEmpty(stack) && peek(stack)!='('){
        printf("Invalid expression\n");
        exit(EXIT_FAILURE);
 }else{
        pop(stack);
    }
}else{
      while(! isEmpty(stack) && precedence(infix[i])
                <= precedence(peek(stack))){
          postfix[++K] = pop(stack);
        }
        push(stack, infix[i]);
      }
}
 while(! isEmpty(stack)){
        postfix[++K] = pop(stack);
    }
    postfix[++K] = '\0';
    return postfix;
    }
    int main(){
        char infix[MAX_SIZE];
```

Register No : Experiment No : Date:

```
    printf("Enter an infix expression:");
    fgets(infix, sizeof(infix), stdin);
    infix[strcspn(infix, "\n")] = '\0';
    char *postfix = infixTOpostfix(infix);
    printf("Postfix expression: %s\n", postfix);
    return 0;
}
```

b) Write C program that implement Queue (it's operations using arrays.

```
#define N 5
int queue[N];
int front = -1;
int rear = -1;
void enqueue(int x){
        if(rear == N-1){
            printf("Queue Overflow");
        }
        else if(front == -1 && rear == -1){
            front = rear = 0;
            queue[rear] = x;
        }
        else{
            rear++;
            queue[rear] = x;
        }
}
void dequeue(){
        if(front == -1 && rear == -1){
```

ster No : Experiment No : Date:

```
    printf("Queue Underflow");
  }
else if (front == rear){
        front=rear=-1;
  }
  else {
     printf("%d", queue[front]);
     front++;
   }
 }
void display(){
      if(front == -1 && rear == -1){
           printf("Queue is empty");
       }
       else{
          for(int i=front; i<rear+1; i++){
              printf("%d", queue[i]);
          }
    }
void peek(){
      if(front == -1 && rear == -1){
          printf("Queue is empty");
       }else{
          printf("%d", Queue[front]);
       }
    }
```

Register No : | Experiment No : | Date:

9) Write C program that implements Queue (its operations) using linked list

```
struct node{
        int data;
        struct node *next;
    };
struct node *front=0;
struct node *rear=0;
void enque(int x){
        Struct node *newnode;
        newnode = (struct node *) malloc(sizeof(struct node));
        newnode -> data = x;
        newnode -> next=0;
        if(front == 0 && rear ==0){
                front = rear = newnode;
          }
        else{
          rear -> next = newnode;
          rear = newnode;
        }
    }
void main(){
        enqueue(5);
        enqueue(0);
        enqueue(-3);
        display();
        dequeue();
        peek();
    }
```

Register No: Experiment No: Date:

S. No	Component	Max. Marks	Marks Secured
1	Preparedness	2	
2	Viva-Voce	2	
3	Experiment	3	
4	Analysis & Record	3	
	Total	10	
Date		Signature of the Lab teacher	

Experiment - 7

AIM: Write a c program that uses functions to create a singly linked list and perform various operations on it.

program:

```
struct node{
    int data;
    struct node *next;
};
void creation_of_linked_list(){
    int choice=1;
    struct node *head, *newnode, *temp;
    head=0;
    while(choice!=0){
        newnode=(struct node *) malloc(sizeof
                                    (struct node));
        printf("Enter data");
```

```
scanf("%d", &newnode -> data);
newnode -> next = 0;
if (head == 0){
        head = newnode;
        temp = newnode;
    }
else{
    temp -> next = newnode;
    temp = temp -> next;
    printf("Do you want to enter another
                        node say(1|0)");
    Scanf("%d", &choice);
    }
  }
}
void insertion_at_begining(){
    Struct node *head, *newnode;
  newnode = (Struct node *) malloc(sizeof(Struct
                                        node));
  printf("Enter data for first node");
  scanf("%d", &newnode -> data);
  newnode -> next = head;
  head = newnode;
  }
```

Register No : Experiment No : Date:

```
void insertion_at_pos(){
        int pos;
        Struct node *newnode, *temp, *head;
newnode =(struct node *) malloc(sizeof(struct node));
printf("Enter position you want to insert");
scanf("%d", &pos);
if(pos > count){
        printf("Invalid position");
    }
else{
    temp = head;
    int i=1;
    while(i< pos - 1){
        temp = temp -> next;
        i++;
    }
    printf("Enter data");
    scanf("%d", &newnode -> data);
    newnode -> next = temp -> next;
    temp -> next = newnode;
}
```

Register No : Experiment No : Date:

```
void insertion_at_end(){
    struct node *head, *newnode, *temp;
    newnode = (struct node *) malloc (sizeof (struct node));
    printf("Enter data for last node");
    scanf("%d", &newnode->data);
    temp = head;
    while (temp->next != Null){
            temp = temp->next;
    }
    temp->next = newnode;
    }
void deletion_at_beg(){
        Struct node *temp;
        temp = head;
        head = head->next;
        free(temp);
        }
void deletion_at_pos(){
        int pos, i=1;
        temp = head;
        printf("Enter position you want to
                delete");
        scanf("%d", &pos);
        while(i<pos-1){
```

r No : Experiment No : Date:

```
        temp = temp -> next;
        i++;
    }
    nextnode = temp -> next;
    temp -> next = nextnode -> next;
    free(nextnode);
}

void deletion_at_end(){
    struct node *temp, *prenode;
    temp = head;
    while(temp -> next != null){
        prenode = temp;
        temp = temp -> next;
    }
    if(temp == head){
        head = 0;
        free(temp);
    }
    else{
        prevnode -> next = null;
        free(temp);
    }
}
```

Register No : Experiment No : Date:

```
void display(){
        int count=0;
        temp=head;
        while(temp!=0){
                printf("%d", temp->data);
                temp=temp->next;
                count++;
            }
         printf("%d", count);
        }
void  main(){
            creation_of_linked_list();
            insertion_at_begining();
            insertion_at_pos();
            insertion_at_end();
            deletion_at_beg();
            deletion_at_pos();
            deletion_at_end();
            display();
        }
```

Register No: Experiment No: Date:

S. No	Component	Max. Marks	Marks Secured
1	Preparedness	2	
2	Viva-Voce	2	
3	Experiment	3	
4	Analysis & Record	3	
	Total	10	
Date		Signature of the Lab teacher	

Experiment- 8

a) **AIM:** Write a C program that use both recursive and non-recursive functions to perform Linear search for a key value in a given list.

```
#include<stdio.h>
int linearSearch(int arr[], int n, int key){
    for(int i=0; i<n; i++){
        if(arr[i]==key){
            return i;
        }
    }
    return -1;
}
int recursivelinearsearch(int arr[], int key, int
                          index, int size){
    if(index>=size){
        return -1;
    }
    if(arr[index]==key){
        return index;
    }
    return recursivelinearsearch(arr,key,index+1,
                                 size);
}
```

r No : Experiment No : Date:

```
int main(){
    int arr[] = {3,5,2,8,9,1,1};
    int key = 8;
    int size = sizeof(arr)/sizeof(arr[0]);
    int result = linearSearch(arr, size, key);
    if(result != -1){
        printf("Non-recursive linear search:
            Element %d fount at index %d\n",
                key, result);
    }else{
        printf("Non-recursive linear search:
            Element %d not found\n", key);
    }
    result = recursivelinearsearch(arr, key, 0, size);
    if(result != -1){
        printf("Recursive linear search: Eleme-
            nt %d found at index %d\n", key,
            result);
    }else{
        printf("Recursive linear search: Element
            %d not found\n", key);
    }
    return 0;
}
```

Register No : Experinient No : Date:

b) Write C program that use both recursive and non-recursive function to perform binary search for a key value in given list.

```
#include<stdio.h>
int binarysearch(int arr[], int left, int right, int key){
    while(left <= right){
        int mid = left + (right-left)/2;
        if(arr[mid] == key){
            return mid;
        }
        if(arr[mid] < key){
            left = mid+1;
        }
        else{
            right = mid-1;
        }
    }
    return -1;
}
int recursivebinarysearch(int arr[], int left, int
                          right, int key){
    if(left <= right){
        int mid = left + (right-left)/2;
        if(arr[mid] == key){
            return mid;
        }
        if(arr[mid] < key){
```

Register No : Experiment No : Date:

```
    return recursivebinarysearch(arr, mid+1, right, key);
  } else {
    return recursivebinarysearch(arr, left, mid-1, key);
  }
 }
  return -1;
 }
int main(){
    int arr[] = {1,2,3,4,5,6,7,8,9};
    int key=7;
    int size = sizeof(arr)/sizeof(arr[0]);
    int result =binarysearch(arr,0, size-1,key);
    if (result! =-1){
            printf("Non-recursive binary search:
                Element %d found at index %d
                \n", key, result);
        }else{
            printf("Non-recursive binary search:
                Element %d not found\n",
                    key);
        }
    result=recursivebinarysearch(arr, 0, size-1, key);
    if (result! = -1){
        printf("Recursive binary search: Eleme-
        nt %d found at index %d \n",key, result);
    }
```

```
else{
    printf("Recursive binary search: Element %d not found\n",
                                                    key);
}
return 0;
}
```

Register No:

Experiment No: 1

Date:

S. No	Component	Max. Marks	Marks Secured
1	Preparedness	2	
2	Viva-Voce	2	
3	Experiment	3	
4	Analysis & Record	3	
	Total	10	
Date		Signature of the Lab teacher	

a) AIM: Write a c program to store a polynomial expression in memory using linked list and perform polynomial addition.

```
#include<stdio.h>
#include <stdlib.h>
struct Term{
        int coefficient;
        int exponent;
        Struct Term *next;
  };
struct Term * createTerm(int coefficient, int exponent){
     Struct Term *term = (Struct Term*) malloc (size-
                                          of(Struct Term));
     if( !term){
           printf ("Memory allocation failed\n");
           exit (EXIT_FAILURE);
         }
       term → coefficient = coefficient;
       term→ exponent= exponent;
```

Register No : Experiment No : Date:

```
    term->next = NULL;
    return term;
}
void insertTerm(struct Term* poly, int coefficient, int exponent){
    struct Term* newTerm = createTerm(coefficient, exponent);
    if (*poly == NULL) {
        *poly = newTerm;
    } else {
        struct Term* temp = *poly;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newTerm;
    }
}
void displayPolynomial(struct Term* poly){
    if (poly == NULL) {
        printf("Polynomial is empty\n");
    } else {
        while (poly != NULL) {
            printf("(%d x^%d)", poly->coefficient, poly->exponent);
            if (poly->next != NULL) {
                printf(" + ");
            }
            poly = poly->next;
        }
        printf("\n");
    }
}
struct Term* addPolynomials(struct Term* poly1,
```

Register No : Experiment No : Date:

```
struct Term* poly2) {
    Struct Term* result = NULL;
    while (poly1 != NULL && poly1y2 != NULL) {
        if (poly1 -> exponent > poly2 -> exponent) {
            insertTerm(&result, poly1 -> coeffi-
                cient, poly1 -> exponent);
            poly1 = poly1 -> next;
        } else if (poly1 -> exponent < poly2 -> expone-
                nt) {
            insertTerm(&result, poly2 -> coefficie-
                nt, poly2 -> exponent);
            poly2 = poly2 -> next;
        } else {
            int sumCoefficients = poly1 -> coefficie-
                nt + poly2 -> coefficient;
            insertTerm(& result, sumCoefficients,
                poly1 -> exponent);
            poly1 = poly1 -> next;
            poly2 = poly2 -> next;
        }
    }
    while (poly1 != NULL) {
        insertTerm(&result, poly2 -> coefficient,
            poly1 -> exponent);
        poly1 = poly1 -> next;
    }
```

Register No : Experiment No : Date:

```
while (poly2 != NULL){
        insertTerm(&result, poly2->coefficient, poly2->
                                                exponent);
        poly2 = poly2->next;
    }
    return result;
}
void freePolynomial(struct Term* poly){
        struct Term * temp;
        while (poly != NULL){
                temp = poly;
                poly = poly->next;
                free(temp);
        }
}
int main(){
        struct Term * poly1 = NULL;
        struct Term* poly2 = NULL;
        struct Term* result = NULL;
        insertTerm(&poly1, 3, 4);
        insertTerm(&poly1, -2, 3);
        insertTerm(&poly1, 5, 2);
        insertTerm(&poly2, 2, 4);
        insertTerm(&poly2, 1, 3);
        insertTerm(&poly2, -3, 1);
        printf("First polynomial:");
        displayPolynomial(poly1);
```

Register No : Experiment No : Date:

```
printf("Second Polynomial:");
displayPolynomial(poly2);
result = addPolynomials(poly1, poly2);
printf("Result of polynomial addition:");
displayPolynomial(result);
freePolynomial(poly1);
freePolynomial(poly2);
freepolynomial(result);
return 0;
}
```

b) Write a recursive C program for traversing a binary Tree in preorder, inorder and postorder.

```
#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*)ma-
                        lloc(sizeof(struct Node));
    if(!newnode){
        printf("Memory allocation failed\n");
        exit(EXIT_FAILURE);
    }
```

Register No : Experiment No : Date:

```
    newnode->data = data;
    newnode->left = null;
    newnode->right = null;
    return newnode;
}
void preordertraversal(Struct Node * root){
        if(root != NULL){
            printf("%d", root->data);
            preordertraversal(root->left);
            preordertraversal(root->right);
        }
    }
void inordertraversal(Struct Node * root){
        if(root != NULL){
            inordertraversal(root->left);
            printf("%d", root->data);
            inorderTraversal(root->right);
        }
    }
void postordertraversal(Struct Node * root){
        if(root != NULL){
            postordertraversal(root->left);
            postordertraversal(root->right);
            printf("%d", root->data);
        }
    }
```

Experiment No : Experiment No : Date:

```
int main() {
    Struct Node *root = CreateNode(1);
    root->left = CreateNode(2);
    root->right = createNode(3);
    root->left->left = createNode(4);
    root->left->right = createNode(5);
    printf(" Preorder traversal:");
    preordertraversal(root);
    printf("\n");
    printf("Inorder traversal:");
    inordertraversal(root);
    printf("\n");
    free(root->left->left);
    free(root->left->right);
    free(root->left);
    free(root->right);
    free(root);
    return 0;
    }
```

Register No:

Experiment No: 10

Date:

S. No	Component	Max. Marks	Marks Secured
1	Preparedness	2	
2	Viva-Voce	2	
3	Experiment	3	
4	Analysis & Record	3	
	Total	10	
Date		Signature of the Lab teacher	

a) AIM: Write a C program that implements bubble sort, to sort a given list of integers in ascending order.

```
#include<stdio.h>
void bubblesort(int arr[], int n){
    for(int i=0; i<n-1; i++){
        int swapped=0;
        for(int j=0; j<n-i-1; j++){
            if(arr[j]>arr[j+1]){
                int temp=arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=temp;
                swapped=1;
            }
        }
        if(swapped==0){
            break;
        }
    }
}
int main(){
    int arr[]={64,25,12,22,11};
```

Register No : Experiment No : Date:

```
int n=sizeof(arr)/sizeof(arr[0]);
printf("Original array:");
for(int i=0; i<n; i++){
        printf("%d", arr[i]);
    }
    printf("\n");
bubbleSort(arr, n);
printf("Sorted array:");
for(int i=0; i<n; i++){
        printf("%d", arr[i]);
    }
    printf("\n");
    return 0;
}
```

b) Write a C program that implements quick sort, to sort a given list of integers in ascending order.

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

Register No : Experiment No : Date:

```
for(int j= low; j<=high-1; j++){
        if(arr[j] <=pivot){
            i++;
            swap(&arr[i], &arr[j];
        }
    }
    swap(&arr[i+1], &arr[high]);
    return(i+1);
}
void quicksort(int arr[], int low, int high){
        if(low<high){
            int pi= partition(arr, low, high);
            quicksort(arr, low, pi-1);
            quicksort(arr, pi+1, high);
        }
    }
int main(){
    int arr[] = {10,7,8,9,1,5};
    int n=sizeof(arr)/sizeof(arr[0]);
    printf("Original array:");
    for(int i=0; i<n; i++){
        printf("%d", arr[i]);
    }
    printf("\n");
    quicksort(arr, 0, n-1);
    printf("Sorted array:");
```

Register No : Experiment No : Date:

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

c) Write a c program that implements merge sort, to sort a given list of integers in ascending order.

```
#include<stdio.h>
#include<stdlib.h>
void merge(int arr[], int l, int m, int r) {
        int i, j, k;
        int n1 = m-l+1;
        int n2 = r-m;
        int L[n1], R[n2];
        for(i=0; i<n1; i++){
                L[i] = arr[l+i];
        for (j=0; j<n2; j++){
                R[j] = arr[m+1+j];
        i=j=0;
        k=l;
        while (i<n1 && j<n2) {
                if(L[i] <= R[j]){
                        arr[k] = L[i];
                        i++;
```

Register No : Experiment No : Date:

```
} 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++;
  }
}
void mergesort(int arr[], int l, int r){
    if (l < r){
        int m = l+(r-l)/2;
        mergesort(arr, l, m);
        mergersort(arr, m+1, r);
        merge(arr, l, m, r);
    }
}
int main(){
    int arr[] = {12,11,13,5,6,7};
```

Experiment No : Date:

```
int n= sizeof(arr)/sizeof(arr[0]);
printf("Original array: ");
for(int i=0; i<n; i++){
        printf("%d", arr[i]);
}
printf("\n");
mergesort(arr,0,n-1);
printf("Sorted array:");
for(int i=0; i<n; i++) {
        printf("%d", arr[i]);
    }
  printf("\n");
  return 0;
}
```