Program 1:

Matrix Multiplication

```
#include<stdio.h>
 #include<stdlib.h>
 int main(void)
{
 int a[10][10],b[10][10],c[10][10],n=0,m=0,i=0,j=0,p=0,q=0,k=0;
        int *pt,*pt1,*pt2;
 printf("Enter size of 1st 2d array : ");
 scanf("%d %d",&n,&m);
        for(i=0;i<n;i++)
 {
                for(j=0;j<m;j++)
                        {
                        printf("Enter element no. %d %d :",i,j);
 scanf("%d",&a[i][j]);
 }
 }
 printf("Enter size of 2nd 2d array : ");
        scanf("%d %d",&p,&q);
 for(i=0;i<p;i++)
 {
 for(j=0;j<q;j++)
                        {
 printf("Enter element no. %d %d :",i,j);
 scanf("%d",&b[i][j]);
                }
 }
 if(m!=p)
                {
 printf("Multiplication cannot be done\n");
 exit (0);
 }
 pt=&a[0][0];
 pt1=&b[0][0];
        pt2=&c[0][0];
 for(i=0;i<n;i++)
                {
                for(k=0;k<q;k++)
                        {
                        *(pt2+(i*10+k))=0;
                        for(j=0;j<m;j++)
 {
 *(pt2+(i*10+k))+=*(pt+(i*10+j))**(pt1+(j*10+k));
                        }
 }
 }
 for(i=0;i<n;i++)
                {
 for(j=0;j<q;j++)
```

```
                        {
                        printf("%d ",c[i][j]);
                }
 printf("\n");
 }
 return 0;
 }
```

Matrix addition using pointers:

```
#include<stdio.h>
void main()
{
int a[5][5],b[5][5],c[5][5],i,j,m,n;
printf("Enter m & n:");
scanf("%d %d",&m,&n);
printf("Enter 1 matrix:");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
scanf("%d",(*(a+i)+j));
}
}
printf("\nEnter 2 matrix:");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
scanf("%d",(*(b+i)+j));
}
}
printf("\nAddition Matrix:\n");
for(i=0;i<m;i++)
{ for(j=0;j<n;j++)
{
*(*(c+i)+j)=*(*(a+i)+j)+ *(*(b+i)+j);
printf("%d ",*(*(c+i)+j));
}
printf("\n");
}
}
```

Matrix Subtraction using Pointers

```
#include<stdio.h>

void main()

{
```

```
int a[5][5],b[5][5],c[5][5],i,j,m,n;
printf("Enter m & n:");
scanf("%d %d",&m,&n);
printf("Enter 1 matrix:");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
scanf("%d",(*(a+i)+j));
}
}
printf("\nEnter 2 matrix:");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
scanf("%d",(*(b+i)+j));
}
}
printf("\nSubtraction Matrix:\n");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
*(*(c+i)+j)=*(*(a+i)+j)- *(*(b+i)+j);
printf("%d ",*(*(c+i)+j));
}
printf("\n");
}
}
```

Program 2 :

Stack implementation using Linked List

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

// Structure to create a node with data and the next pointer
struct Node {
   int data;
   struct Node *next;
};
Node* top = NULL;

// Push() operation on a  stack
void push(int value) {
   struct Node *newNode;
   newNode = (struct Node *)malloc(sizeof(struct Node));
   newNode->data = value; // assign value to the node
   if (top == NULL) {
      newNode->next = NULL;
   } else {
      newNode->next = top; // Make the node as top
   }
   top = newNode; // top always points to the newly created node
   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);
      return temp_data;
   }
}

void display() {
   // Display the elements of the stack
   if (top == NULL) {
      printf("\nStack Underflow\n");
   } else {
      printf("The stack is \n");
      struct Node *temp = top;
      while (temp->next != NULL) {
         printf("%d--->", temp->data);
         temp = temp->next;
      }
      printf("%d--->NULL\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:
         printf("Popped element is :%d\n", pop());
         break;
      case 3:
         display();
         break;
      case 4:
         exit(0);
         break;
      default:
         printf("\nWrong Choice\n");
      }
   }
}
```

Stack implementation using Arrays:

```
#include<stdio.h>
int stack[100],choice,n,top,x,i;
void push(void);
void pop(void);
void display(void);
int main()
{
   //clrscr();
   top=-1;
   printf("\n Enter the size of STACK[MAX=100]:");
   scanf("%d",&n);
   printf("\n\t STACK OPERATIONS USING ARRAY");
   printf("\n\t--------------------------------");
   printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");
   do
   {
      printf("\n Enter the Choice:");
      scanf("%d",&choice);
      switch(choice)
```

```
        {
            case 1:
            {
                push();
                break;
            }
            case 2:
            {
                pop();
                break;
            }
            case 3:
            {
                display();
                break;
            }
            case 4:
            {
                printf("\n\t EXIT POINT ");
                break;
            }
            default:
            {
                printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");
            }

        }
    }
    while(choice!=4);
    return 0;
}
void push()
{
    if(top>=n-1)
    {
        printf("\n\tSTACK is over flow");

    }
    else
    {
        printf(" Enter a value to be pushed:");
        scanf("%d",&x);
        top++;
        stack[top]=x;
    }
}
void pop()
{
    if(top<=-1)
    {
        printf("\n\t Stack is under flow");
```

```
    }
    else
    {
        printf("\n\t The popped elements is %d",stack[top]);
        top--;
    }
}
void display()
{
    if(top>=0)
    {
        printf("\n The elements in STACK \n");
        for(i=top; i>=0; i--)
            printf("\n%d",stack[i]);
        printf("\n Press Next Choice");
    }
    else
    {
        printf("\n The STACK is empty");
    }

}
```

Program 3:

Develop a program to demonstrate concept of recursion (Factorial / Binary Search / Fibonacci /Towers of Hanoi)

Factorial :

```
#include<stdio.h>
 int fact(int);
int main()
{
    int x,n;
    printf(" Enter the Number to Find Factorial :");
    scanf("%d",&n);

    x=fact(n);
    printf(" Factorial of %d is %d",n,x);

    return 0;
}
int fact(int n)
{
    if(n==0)
        return(1);
    return(n*fact(n-1));
}
```

Binary Search Program in C using Recursive and Non-Recursive Methods:

```
#include <stdio.h>
#define MAX_LEN 10
 /* Non-Recursive function*/
void b_search_nonrecursive(int l[],int num,int ele)
{
   int l1,i,j, flag = 0;
   l1 = 0;
   i = num-1;
   while(l1 <= i)
   {
      j = (l1+i)/2;
      if( l[j] == ele)
      {
    printf("\nThe element %d is present at position %d in list\n",ele,j);
         flag =1;
         break;
      }
      else
        if(l[j] < ele)
           l1 = j+1;
        else
           i = j-1;
   }
   if( flag == 0)
   printf("\nThe element %d is not present in the list\n",ele);
}

/* Recursive function*/
int b_search_recursive(int l[],int arrayStart,int arrayEnd,int a)
{
  int m,pos;
  if (arrayStart<=arrayEnd)
  {
    m=(arrayStart+arrayEnd)/2;
    if (l[m]==a)
      return m;
    else if (a<l[m])
      return b_search_recursive(l,arrayStart,m-1,a);
    else
      return b_search_recursive(l,m+1,arrayEnd,a);
   }
   return -1;
}

void read_list(int l[],int n)
{
   int i;
   printf("\nEnter the elements:\n");
   for(i=0;i<n;i++)
```

```
      scanf("%d",&l[i]);
}

void print_list(int l[],int n)
{
   int i;
  for(i=0;i<n;i++)
      printf("%d\t",l[i]);
}

/*main function*/
main()
{
  int l[MAX_LEN], num, ele,f,l1,a;
  int ch,pos;

  //clrscr();

  printf("=====================================================");
  printf("\n\t\t\tMENU");
  printf("\n=====================================================");
  printf("\n[1] Binary Search using Recursion method");
  printf("\n[2] Binary Search using Non-Recursion method");
  printf("\n\nEnter your Choice:");
  scanf("%d",&ch);

  if(ch<=2 & ch>0)
  {
    printf("\nEnter the number of elements : ");
    scanf("%d",&num);
    read_list(l,num);
    printf("\nElements present in the list are:\n\n");
    print_list(l,num);
    printf("\n\nEnter the element you want to search:\n\n");
    scanf("%d",&ele);


  switch(ch)
  {
   case 1:printf("\nRecursive method:\n");
      pos=b_search_recursive(l,0,num,ele);
      if(pos==-1)
      {
      printf("Element is not found");
      }
      else
      {
      printf("Element is found at %d position",pos);
      }
      //getch();
      break;
```

```
    case 2:printf("\nNon-Recursive method:\n");
      b_search_nonrecursive(l,num,ele);
      //getch();
      break;
    }
  }
//getch();
}
```

Fibonacci:

Fibonacci Without Recursion:

```
#include<stdio.h>
int main()
{
 int n1=0,n2=1,n3,i,number;
 printf("Enter the number of elements:");
 scanf("%d",&number);
 printf("\n%d %d",n1,n2);//printing 0 and 1
 for(i=2;i<number;++i)//loop starts from 2 because 0 and 1 are already printed
 {
  n3=n1+n2;
  printf(" %d",n3);
  n1=n2;
  n2=n3;
 }
  return 0;
 }
```

Fibonacci With Recursion:

```
#include<stdio.h>
void printFibonacci(int n){
   static int n1=0,n2=1,n3;
   if(n>0){
        n3 = n1 + n2;
        n1 = n2;
        n2 = n3;
        printf("%d ",n3);
        printFibonacci(n-1);
   }
}
int main(){
   int n;
   printf("Enter the number of elements: ");
   scanf("%d",&n);
   printf("Fibonacci Series: ");
   printf("%d %d ",0,1);
   printFibonacci(n-2);//n-2 because 2 numbers are already printed
```

```
 return 0;
}
```

Towers of Honoi using recursion:

```
#include <stdio.h>
 void towers(int, char, char, char);
 int main()
{
    int num;
     printf("Enter the number of disks : ");
    scanf("%d", &num);
    printf("The sequence of moves involved in the Tower of Hanoi are :\n");
    towers(num, 'A', 'C', 'B');
    return 0;
}
void towers(int num, char frompeg, char topeg, char auxpeg)
{
    // Base Condition if no of disks are
    if (num == 1)
    {
        printf("\n Move disk 1 from peg %c to peg %c", frompeg, topeg);
        return;
    }

    // Recursively calling function twice
    towers(num - 1, frompeg, auxpeg, topeg);
    printf("\n Move disk %d from peg %c to peg %c", num, frompeg, topeg);
    towers(num - 1, auxpeg, topeg, frompeg);
}
```

Program 4 : Design and implement an algorithm for conversion of an expression from infix to postfix and infix to prefix form. Demonstrate its working with suitable inputs

Infix to Prefix Program using arrays :

```
#include<stdio.h>
#include<string.h>
#include<limits.h>
#include<stdlib.h>
# define MAX 100
int top = -1;
char stack[MAX];

// checking if stack is full
int isFull() {
    return top == MAX - 1;
}

// checking is stack is empty
int isEmpty() {
    return top == -1;
```

```
}

// Push function here, inserts value in stack and increments stack top by 1
void push(char item) {
   if (isFull())
      return;
         top++;
         stack[top] = item;
}

// Function to remove an item from stack.  It decreases top by 1
int pop() {
   if (isEmpty())
      return INT_MIN;

   // decrements top and returns what has been popped
   return stack[top--];
}

// Function to return the top from stack without removing it
int peek(){
   if (isEmpty())
      return INT_MIN;
   return stack[top];
}

// A utility function to check if the given character is operand
int checkIfOperand(char ch) {
   return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z');
}

// Fucntion to compare precedence
// If we return larger value means higher precedence
int precedence(char ch)
{
   switch (ch)
   {
   case '+':
   case '-':
      return 1;

   case '*':
   case '/':
      return 2;

   case '^':
      return 3;
   }
   return -1;
}

// The driver function for infix to postfix conversion
int getPostfix(char* expression)
{
   int i, j;
```

```
    for (i = 0, j = -1; expression[i]; ++i)
    {
        // Here we are checking is the character we scanned is operand or not
        // and this adding to to output.
        if (checkIfOperand(expression[i]))
            expression[++j] = expression[i];

        // Here, if we scan character '(', we need push it to the stack.
        else if (expression[i] == '(')
            push(expression[i]);

        // Here, if we scan character is an ')', we need to pop and print from the stack
        // do this until an '(' is encountered in the stack.
        else if (expression[i] == ')')
        {
            while (!isEmpty(stack) && peek(stack) != '(')
                expression[++j] = pop(stack);
            if (!isEmpty(stack) && peek(stack) != '(')
                return -1; // invalid expression
            else
                pop(stack);
        }
        else // if an opertor
        {
            while (!isEmpty(stack) && precedence(expression[i]) <= precedence(peek(stack)))
                expression[++j] = pop(stack);
            push(expression[i]);
        }

    }

    // Once all inital expression characters are traversed
    // adding all left elements from stack to exp
    while (!isEmpty(stack))
        expression[++j] = pop(stack);

    expression[++j] = '\0';

}

void reverse(char *exp){

    int size = strlen(exp);
    int j = size, i=0;
    char temp[size];

    temp[j--]='\0';
    while(exp[i]!='\0')
    {
        temp[j] = exp[i];
        j--;
        i++;
    }
    strcpy(exp,temp);
```

```
}
void brackets(char* exp){
   int i = 0;
   while(exp[i]!='\0')
   {
      if(exp[i]=='(')
         exp[i]=')';
      else if(exp[i]==')')
         exp[i]='(';
      i++;
   }
}
void InfixtoPrefix(char *exp){

   int size = strlen(exp);

   // reverse string
   reverse(exp);
   //change brackets
   brackets(exp);
   //get postfix
   getPostfix(exp);
   // reverse string again
   reverse(exp);
}

int main()
{
   printf("The infix is: ");

   char expression[] = "((a/b)+c)-(d+(e*f))";
   printf("%s\n",expression);
   InfixtoPrefix(expression);

   printf("The prefix is: ");
   printf("%s\n",expression);

   return 0;
}
```

Infix to PostFix using stack

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

char stack[100];
int top = -1;

void push(char x)
{
   stack[++top] = x;
}

char pop()
```

```
{
   if(top == -1)
      return -1;
   else
      return stack[top--];
}

int priority(char x)
{
   if(x == '(')
      return 0;
   if(x == '+' || x == '-')
      return 1;
   if(x == '*' || x == '/')
      return 2;
   return 0;
}

int main()
{
   char exp[100];
   char *e, x;
   printf("Enter the expression : ");
   scanf("%s",exp);
   printf("\n");
   e = exp;

   while(*e != '\0')
   {
      if(isalnum(*e))
         printf("%c ",*e);
      else if(*e == '(')
         push(*e);
      else if(*e == ')')
      {
         while((x = pop()) != '(')
            printf("%c ", x);
      }
      else
      {
         while(priority(stack[top]) >= priority(*e))
            printf("%c ",pop());
         push(*e);
      }
      e++;
   }

   while(top != -1)
   {
      printf("%c ",pop());
   }return 0;
```

```
}
```

Program 5 :

C Program for Postfix Evaluation

```
// C program to evaluate value of a postfix expression
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>

// Stack type
struct Stack
{
	int top;
	unsigned capacity;
	int* array;
};

// Stack Operations
struct Stack* createStack( unsigned capacity )
{
	struct Stack* stack = (struct Stack*) malloc(sizeof(struct Stack));

	if (!stack) return NULL;

	stack->top = -1;
	stack->capacity = capacity;
	stack->array = (int*) malloc(stack->capacity * sizeof(int));

	if (!stack->array) return NULL;

	return stack;
}

int isEmpty(struct Stack* stack)
{
	return stack->top == -1 ;
}

char peek(struct Stack* stack)
{
	return stack->array[stack->top];
}

char pop(struct Stack* stack)
{
	if (!isEmpty(stack))
		return stack->array[stack->top--] ;
```

```
	return '$';
}

void push(struct Stack* stack, char op)
{
	stack->array[++stack->top] = op;
}


// The main function that returns value of a given postfix expression
int evaluatePostfix(char* exp)
{
	// Create a stack of capacity equal to expression size
	struct Stack* stack = createStack(strlen(exp));
	int i;

	// See if stack was created successfully
	if (!stack) return -1;

	// Scan all characters one by one
	for (i = 0; exp[i]; ++i)
	{
		// If the scanned character is an operand (number here),
		// push it to the stack.
		if (isdigit(exp[i]))
			push(stack, exp[i] - '0');

		// If the scanned character is an operator, pop two
		// elements from stack apply the operator
		else
		{
			int val1 = pop(stack);
			int val2 = pop(stack);
			switch (exp[i])
			{
			case '+': push(stack, val2 + val1); break;
			case '-': push(stack, val2 - val1); break;
			case '*': push(stack, val2 * val1); break;
			case '/': push(stack, val2/val1); break;
			}
		}
	}
	return pop(stack);
}

// Driver program to test above functions
int main()
{
	char exp[] = "231*+9-";
	printf ("postfix evaluation: %d", evaluatePostfix(exp));
	return 0;
```

```
}
```

C Program for Prefix Evaluation

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

int s[50];
int top=0;

void push(int ch);
int pop();

int main()
{
        int a,b,c,i;
        char prefix[50];
        clrscr();

        printf("\nEnter the prefix string in figures(1 digit nos);");
        gets(prefix);

        //for(i=0;i<strlen(prefix);i++)
        for(i=strlen(prefix)-1;i>=0;i--)
        {
                if(prefix[i]=='+')
                {
                        c=pop()+pop();
                        push(c);
                }
                else if(prefix[i]=='-')
                {
                        a=pop();
                        b=pop();
                        c=b-a;
                        push(c);
                }
                else if(prefix[i]=='*')
                {       a=pop();
                b=pop();
                c=b*a;
                push(c);
                }
                else if(prefix[i]=='/')
                {
                        a=pop();
                        b=pop();
                        c=b/a;
                        push(c);
```

```
                }
                else
                {
                        push(prefix[i]-48);
                        //printf("\n INT=%d - CHAR=%d",prefix[i]-48,c);
                }
        }
        printf("\nFinal ans = %d",pop());

        getch();
        return 0;
}
```

Queue Implementation using Linked List

```
// A C program to demonstrate linked list based
// implementation of queue
#include <stdio.h>
#include <stdlib.h>

// A linked list (LL) node to store a queue entry
struct QNode {
        int key;
        struct QNode* next;
};

// The queue, front stores the front node of LL and rear
// stores the last node of LL
struct Queue {
        struct QNode *front, *rear;
};

// A utility function to create a new linked list node.
struct QNode* newNode(int k)
{
        struct QNode* temp
                = (struct QNode*)malloc(sizeof(struct QNode));
        temp->key = k;
        temp->next = NULL;
        return temp;
}

// A utility function to create an empty queue
struct Queue* createQueue()
{
        struct Queue* q
                = (struct Queue*)malloc(sizeof(struct Queue));
        q->front = q->rear = NULL;
        return q;
}

// The function to add a key k to q
```

```
void enQueue(struct Queue* q, int k)
{
	// Create a new LL node
	struct QNode* temp = newNode(k);

	// If queue is empty, then new node is front and rear
	// both
	if (q->rear == NULL) {
		q->front = q->rear = temp;
		return;
	}

	// Add the new node at the end of queue and change rear
	q->rear->next = temp;
	q->rear = temp;
}

// Function to remove a key from given queue q
void deQueue(struct Queue* q)
{
	// If queue is empty, return NULL.
	if (q->front == NULL)
		return;

	// Store previous front and move front one node ahead
	struct QNode* temp = q->front;

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

	// If front becomes NULL, then change rear also as NULL
	if (q->front == NULL)
		q->rear = NULL;

	free(temp);
}

// Driver Program to test above functions
int main()
{
	struct Queue* q = createQueue();
	enQueue(q, 10);
	enQueue(q, 20);
	deQueue(q);
	deQueue(q);
	enQueue(q, 30);
	enQueue(q, 40);
	enQueue(q, 50);
	deQueue(q);
	printf("Queue Front : %d \n", q->front->key);
	printf("Queue Rear : %d", q->rear->key);
	return 0;
```

```
}
void push(int ch)
{
        top++;
        s[top]=ch;
}

int pop()
{
        int ch;
        ch=s[top];
        top=top-1;
        return(ch);
}
```

Queue Implementation using Array

```
#include <stdio.h>

#define MAX 50

void insert();
void delete();
void display();
int queue_array[MAX];
int rear = - 1;
int front = - 1;
main()
{
   int choice;
   while (1)
   {
      printf("1.Insert element to queue \n");
      printf("2.Delete element from queue \n");
      printf("3.Display all elements of queue \n");
      printf("4.Quit \n");
      printf("Enter your choice : ");
      scanf("%d", &choice);
      switch (choice)
      {
         case 1:
         insert();
         break;
         case 2:
         delete();
         break;
         case 3:
         display();
         break;
         case 4:
```

```
            exit(1);
            default:
            printf("Wrong choice \n");
        } /* End of switch */
    } /* End of while */
} /* End of main() */

void insert()
{
    int add_item;
    if (rear == MAX - 1)
    printf("Queue Overflow \n");
    else
    {
        if (front == - 1)
        /*If queue is initially empty */
        front = 0;
        printf("Inset the element in queue : ");
        scanf("%d", &add_item);
        rear = rear + 1;
        queue_array[rear] = add_item;
    }
} /* End of insert() */

void delete()
{
    if (front == - 1 || front > rear)
    {
        printf("Queue Underflow \n");
        return ;
    }
    else
    {
        printf("Element deleted from queue is : %d\n", queue_array[front]);
        front = front + 1;
    }
} /* End of delete() */

void display()
{
    int i;
    if (front == - 1)
        printf("Queue is empty \n");
    else
    {
        printf("Queue is : \n");
        for (i = front; i <= rear; i++)
            printf("%d ", queue_array[i]);
        printf("\n");
    }
}
```

Circular Queue Implementation using Array

```
#include <stdio.h>

# define max 6
int queue[max];  // array declaration
int front=-1;
int rear=-1;
// function to insert an element in a circular queue
void enqueue(int element)
{
   if(front==-1 && rear==-1)   // condition to check queue is empty
   {
      front=0;
      rear=0;
      queue[rear]=element;
   }
   else if((rear+1)%max==front)  // condition to check queue is full
   {
      printf("Queue is overflow..");
   }
   else
   {
      rear=(rear+1)%max;       // rear is incremented
      queue[rear]=element;     // assigning a value to the queue at the rear position.
   }
}

// function to delete the element from the queue
int dequeue()
{
   if((front==-1) && (rear==-1))  // condition to check queue is empty
   {
      printf("\nQueue is underflow..");
   }
 else if(front==rear)
{
   printf("\nThe dequeued element is %d", queue[front]);
   front=-1;
   rear=-1;
}
else
{
    printf("\nThe dequeued element is %d", queue[front]);
   front=(front+1)%max;
}
}
// function to display the elements of a queue
void display()
{
```

```
    int i=front;
    if(front==-1 && rear==-1)
    {
        printf("\n Queue is empty..");
    }
    else
    {
        printf("\nElements in a Queue are :");
        while(i<=rear)
        {
            printf("%d,", queue[i]);
            i=(i+1)%max;
        }
    }
}
int main()
{
    int choice=1,x;   // variables declaration

    while(choice<4 && choice!=0)   // while loop
    {
    printf("\n Press 1: Insert an element");
    printf("\nPress 2: Delete an element");
    printf("\nPress 3: Display the element");
    printf("\nEnter your choice");
    scanf("%d", &choice);

    switch(choice)
    {

        case 1:

        printf("Enter the element which is to be inserted");
        scanf("%d", &x);
        enqueue(x);
        break;
        case 2:
        dequeue();
        break;
        case 3:
        display();

    }}
    return 0;
}
```

Circular Queue Implementation using Linked List

```
#include <stdio.h>
// Declaration of struct type node
```

```
struct node
{
   int data;
   struct node *next;
};
struct node *front=-1;
struct node *rear=-1;
// function to insert the element in the Queue
void enqueue(int x)
{
   struct node *newnode;  // declaration of pointer of struct node type.
   newnode=(struct node *)malloc(sizeof(struct node));  // allocating the memory to the newnode
   newnode->data=x;
   newnode->next=0;
   if(rear==-1)  // checking whether the Queue is empty or not.
   {
      front=rear=newnode;
      rear->next=front;
   }
   else
   {
      rear->next=newnode;
      rear=newnode;
      rear->next=front;
   }
}

// function to delete the element from the queue
void dequeue()
{
   struct node *temp;   // declaration of pointer of node type
   temp=front;
   if((front==-1)&&(rear==-1))  // checking whether the queue is empty or not
   {
      printf("\nQueue is empty");
   }
   else if(front==rear)  // checking whether the single element is left in the queue
   {
      front=rear=-1;
      free(temp);
   }
   else
   {
      front=front->next;
      rear->next=front;
      free(temp);
   }
}

// function to get the front of the queue
int peek()
```

```
{
   if((front==-1) &&(rear==-1))
   {
      printf("\nQueue is empty");
   }
   else
   {
      printf("\nThe front element is %d", front->data);
   }
}

// function to display all the elements of the queue
void display()
{
   struct node *temp;
   temp=front;
   printf("\n The elements in a Queue are : ");
   if((front==-1) && (rear==-1))
   {
      printf("Queue is empty");
   }

   else
   {
      while(temp->next!=front)
      {
         printf("%d,", temp->data);
         temp=temp->next;
      }
      printf("%d", temp->data);
   }
}

void main()
{
   enqueue(34);
   enqueue(10);
   enqueue(23);
   display();
   dequeue();
   peek();
}
```

Program 7 :

Design, Develop and Implement a menu driven Program in C for the following operations on Singly Linked List (SLL) of Student Data with the fields: USN, Name, Branch, Sem, PhNo
a. Create a SLL of N Students Data by using front insertion.
b. Display the status of SLL and count the number of nodes in it

c. Perform Insertion / Deletion at End of SLL
d. Perform Insertion / Deletion at Front of SLL(Demonstration of stack)
e. Exit

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

struct node
{
   char usn[25],name[25],branch[25];
   int sem;
   long int phone;
   struct node *link;
};
typedef struct node * NODE;

NODE start = NULL;
int count=0;


NODE create()
{
   NODE snode;
   snode = (NODE)malloc(sizeof(struct node));

   if(snode == NULL)
   {
      printf("\nMemory is not available");
      exit(1);
   }
   printf("\nEnter the usn,Name,Branch, sem,PhoneNo of the student:");
   scanf("%s %s %s %d %ld",snode->usn, snode->name, snode->branch, &snode->sem,
&snode->phone);
   snode->link=NULL;
   count++;
   return snode;
}

NODE insertfront()
{
   NODE temp;
   temp = create();
   if(start == NULL)
   {
         return temp;
   }

   temp->link = start;
   return temp;
}
```

```
NODE deletefront()
{
   NODE temp;
   if(start == NULL)
   {
      printf("\nLinked list is empty");
      return NULL;
   }

   if(start->link == NULL)
   {
         printf("\nThe Student node with usn:%s is deleted ",start->usn);
         count--;
         free(start);
         return NULL;
   }
   temp = start;
   start = start->link;
   printf("\nThe Student node with usn:%s is deleted",temp->usn);
   count--;
   free(temp);
   return start;
}

NODE insertend()
{
   NODE cur,temp;
   temp = create();

   if(start == NULL)
   {
    return temp;
   }
   cur = start;
   while(cur->link !=NULL)
   {
       cur = cur->link;
   }
   cur->link = temp;
   return start;
}

NODE deleteend()
{
    NODE cur,prev;
    if(start == NULL)
    {
      printf("\nLinked List is empty");
      return NULL;
    }
```

```
    if(start->link == NULL)
    {
      printf("\nThe student node with the usn:%s is deleted",start->usn);
      free(start);
      count--;
      return NULL;
    }

    prev = NULL;
    cur = start;
    while(cur->link!=NULL)
    {
       prev = cur;
       cur = cur->link;
    }

     printf("\nThe student node with the usn:%s is deleted",cur->usn);
     free(cur);
     prev->link = NULL;
     count--;
     return start;
}

void display()
{
   NODE cur;
   int num=1;


   if(start == NULL)
   {

      printf("\nNo Contents to display in SLL \n");
      return;
   }
   printf("\nThe contents of SLL: \n");
   cur = start;
   while(cur!=NULL)
   {
     printf("\n||%d|| USN:%s| Name:%s| Branch:%s| Sem:%d| Ph:%ld|",num,cur->usn,
cur->name,cur->branch, cur->sem,cur->phone);
      cur = cur->link;
      num++;
   }
   printf("\n No of student nodes is %d \n",count);
}

void stackdemo()
{
  int ch;
```

```
  while(1)
  {
   printf("\n~~~Stack Demo using SLL~~~\n");
   printf("\n1:Push operation \n2: Pop operation \n3: Display \n4:Exit \n");
   printf("\nEnter your choice for stack demo");
   scanf("%d",&ch);

   switch(ch)
   {
     case 1: start = insertfront();
             break;
     case 2: start = deletefront();
             break;
     case 3: display();
            break;
    default : return;
   }
  }
  return;
}

int main()
{
   int ch,i,n;
   while(1)
   {
      printf("\n~~~Menu~~~");
      printf("\nEnter your choice for SLL operation \n");
      printf("\n1:Create SLL of Student Nodes");
      printf("\n2:DisplayStatus");
      printf("\n3:InsertAtEnd");
      printf("\n4:DeleteAtEnd");
      printf("\n5:Stack Demo using SLL(Insertion and Deletion at Front)");
      printf("\n6:Exit \n");
      printf("\nEnter your choice:");
      scanf("%d",&ch);

      switch(ch)
      {
      case 1 : printf("\nEnter the no of students:    ");
               scanf("%d",&n);
               for(i=1;i<=n;i++)
                  start = insertfront();
               break;

      case 2: display();
              break;

      case 3: start = insertend();
              break;
```

```
        case 4: start = deleteend();
                break;

        case 5: stackdemo();
                break;

        case 6: exit(0);

        default: printf("\nPlease enter the valid choice");

        }
    }
}
```

Program 8:

```
/*
8 Design, Develop and Implement a menu driven Program in C for the
following operations on Doubly Linked List (DLL) of Employee Data with
the fields: SSN, Name, Dept, Designation, Sal, PhNo
a. Create a DLL of N Employees Data by using end insertion.
b. Display the status of DLL and count the number of nodes in it
c. Perform Insertion and Deletion at End of DLL
d. Perform Insertion and Deletion at Front of DLL
e. Demonstrate how this DLL can be used as Double Ended Queue
f. Exit
*/

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

typedef struct node
{
        char ssn[20], name[20], department[20], designation[20];
        float sal;
        long int phno;
        struct node *llink, *rlink;
} NODE;

typedef struct headnode
{
        int count;
        struct node *llink, *rlink;
} HEAD;

void insfront(HEAD *head);
void insrear(HEAD *head);
void delfront(HEAD *head);
void delrear(HEAD *head);
void display(HEAD *head);
NODE *getNode();
```

```
void main()
{
	int ch;
	HEAD *head = (HEAD *) malloc(sizeof(HEAD));
	head->count = 0;
	head->llink = NULL;
	head->rlink = NULL;
	for(;;)
	{
		printf("\n\nMenu\n");
		printf("\n1. Insert Front\n2. Insert Rear\n3. Delete Front\n4. Delete Rear\n5.
Display\n6. Exit\n");
		scanf("%d", &ch);
		switch(ch)
		{
			case 1:
				insfront(head);
				break;
			case 2:
				insrear(head);
				break;
			case 3:
				if(head->rlink == NULL)
					printf("List Empty");
				else
					delfront(head);
				break;
			case 4:
				if(head->rlink == NULL)
					printf("List Empty");
				else
					delrear(head);
				break;
			case 5:
				if(head->rlink == NULL)
					printf("List Empty");
				else
					display(head);
				break;
			case 6:
				exit(0);
		}
	}
}

NODE *getNode()
{
	NODE *temp = (NODE *) malloc(sizeof(NODE));
	if(temp == NULL)
	{
```

```
                printf("No Memory\n");
                exit(0);
        }
        return temp;
}

void insfront(HEAD *head)
{
        NODE *new = getNode();
        NODE *next = head->rlink;
        printf("Enter Details such as SSN Name Department Designation Salary PhNo\n");
        scanf("%s%s%s%s%f%ld", (new->ssn), (new->name), (new->department),
(new->designation), &(new->sal), &(new->phno));
        if(next != NULL)
                next->llink = new;
        new->rlink = next;
        head->rlink = new;
        (head->count)++;
}

void insrear(HEAD *head)
{
        NODE *new = getNode();
        NODE *temp = NULL;
        printf("Enter Details such as SSN Name Department Designation Salary PhNo\n");
        scanf("%s%s%s%s%f%ld", (new->ssn), (new->name), (new->department),
(new->designation), &(new->sal), &(new->phno));
        (head->count)++;
        new->rlink = NULL;
        if(head->rlink == NULL)
        {
                head->rlink = new;
                return;
        }
        temp = head->rlink;
        while(temp->rlink != NULL)
                temp = temp->rlink;
        temp->rlink = new;
        new->llink = temp;
}

void delfront(HEAD *head)
{
        NODE *temp = head->rlink;
        printf("Deleted Record is\n");
        printf("%s\t%s\t%s\t%s\t%f\t%ld\n", (temp->ssn), (temp->name), (temp->department),
(temp->designation), (temp->sal), (temp->phno));
        head->rlink = temp->rlink;
        free(temp);
        (head->count)--;
}
```

```
void delrear(HEAD *head)
{
	NODE *previous = NULL, *present = head->rlink;
	if(present->rlink == NULL)
	{
		head->rlink = NULL;
	}
	else
	{
		while(present->rlink != NULL)
		{
			previous = present;
			present =  present->rlink;
		}
		previous->rlink = NULL;
	}
	printf("Deleted Record is\n");
	printf("%s\t%s\t%s\t%s\t%f\t%ld\n", (present->ssn), (present->name), (present->department),
(present->designation), (present->sal), (present->phno));
	(head->count)--;
	free(present);
}

void display(HEAD *head)
{
	NODE *temp = head->rlink;
	printf("Total Number of records are %d\n", head->count);
	printf("SSN\tName\tDepartment\tDesignation\tSalary\t\tPhNo\n");
	while(temp != NULL)
	{
		printf("%s\t%s\t%s\t\t%s\t\t%f\t%ld\n", (temp->ssn), (temp->name),
(temp->department), (temp->designation), (temp->sal), (temp->phno));
		temp = temp->rlink;

	}
}
```

Program 9:

```
#include <stdio.h>

#include <stdlib.h>

struct BST

{

int data;

struct BST *left;
```

```
struct BST *right;
};
typedef struct BST NODE;
NODE *node;
NODE* createtree(NODE *node, int data)
{
if (node == NULL)
{
NODE *temp;
temp= (NODE*)malloc(sizeof(NODE));
temp->data = data;
temp->left = temp->right = NULL;
return temp;
}
if (data < (node->data))
{
node->left = createtree(node->left, data);
}
else if (data > node->data)
{
node -> right = createtree(node->right, data);
}
return node;
}
NODE* search(NODE *node, int data)
```

```
{
if(node == NULL)
printf("\nElement not found");
else if(data < node->data)
{
node->left=search(node->left, data);
}
else if(data > node->data)
{
node->right=search(node->right, data);
}
else
printf("\nElement found is: %d", node->data);
return node;
}
void inorder(NODE *node)
{
if(node != NULL)
{
inorder(node->left);
printf("%d\t", node->data);
inorder(node->right);
}
}
void preorder(NODE *node)
{
```

```
if(node != NULL)
{
printf("%d\t", node->data);
preorder(node->left);
preorder(node->right);
}
}
void postorder(NODE *node)
{
if(node != NULL)
{
postorder(node->left);
postorder(node->right);
printf("%d\t", node->data);
}
}
NODE* findMin(NODE *node)
{
if(node==NULL)
{
return NULL;
}
if(node->left)
return findMin(node->left);
else
```

```
return node;

}

NODE* del(NODE *node, int data)

{

NODE *temp;

if(node == NULL)

{

printf("\nElement not found");

}

else if(data < node->data)

{

node->left = del(node->left, data);

}

else if(data > node->data)

{

node->right = del(node->right, data);

}

else

{ /* Now We can delete this node and replace with either minimum element in the right sub tree or maximum element in the left subtree */

if(node->right && node->left)

{ /* Here we will replace with minimum element in the right sub tree */

temp = findMin(node->right);

node -> data = temp->data;

/* As we replaced it with some other node, we have to delete that node */

node -> right = del(node->right,temp->data);
```

```
}

else

{

/* If there is only one or zero children then we can directly remove it from the tree and connect its parent to its child */

temp = node;

if(node->left == NULL)

node = node->right;

else if(node->right == NULL)

node = node->left;

free(temp); /* temp is longer required */

}

}

return node;

}

void main()

{

int data, ch, i, n;

NODE *root=NULL;

while (1)

{

printf("\n1.Insertion in Binary Search Tree");

printf("\n2.Search Element in Binary Search Tree");

printf("\n3.Delete Element in Binary Search Tree");

printf("\n4.Inorder\n5.Preorder\n6.Postorder\n7.Exit");

printf("\nEnter your choice: ");
```

```
scanf("%d", &ch);
switch (ch)
{
case 1: printf("\nEnter N value: " );
scanf("%d", &n);
printf("\nEnter the values to create BST like(6,9,5,2,8,15,24,14,7,8,5,2)\n");
for(i=0; i<n; i++)
{
scanf("%d", &data);
root=createtree(root, data);
}
break;
case 2: printf("\nEnter the element to search: ");
scanf("%d", &data);
root=search(root, data);
break;
case 3: printf("\nEnter the element to delete: ");
scanf("%d", &data);
root=del(root, data);
break;
case 4: printf("\nInorder Traversal: \n");
inorder(root);
break;
case 5: printf("\nPreorder Traversal: \n");
preorder(root);
break;
```

```
case 6: printf("\nPostorder Traversal: \n");

postorder(root);

break;

case 7: exit(0);

default:printf("\nWrong option");

break;

}

}

}
```

Program 10:

```
// Tree traversal in C

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

struct node {
  int item;
  struct node* left;
  struct node* right;
};

// Inorder traversal
void inorderTraversal(struct node* root) {
  if (root == NULL) return;
  inorderTraversal(root->left);
  printf("%d ->", root->item);
  inorderTraversal(root->right);
}

// Preorder traversal
void preorderTraversal(struct node* root) {
  if (root == NULL) return;
  printf("%d ->", root->item);
  preorderTraversal(root->left);
  preorderTraversal(root->right);
}

// Postorder traversal
void postorderTraversal(struct node* root) {
```

```
  if (root == NULL) return;
  postorderTraversal(root->left);
  postorderTraversal(root->right);
  printf("%d ->", root->item);
}

// Create a new Node
struct node* createNode(value) {
  struct node* newNode = malloc(sizeof(struct node));
  newNode->item = value;
  newNode->left = NULL;
  newNode->right = NULL;

  return newNode;
}

// Insert on the left of the node
struct node* insertLeft(struct node* root, int value) {
  root->left = createNode(value);
  return root->left;
}

// Insert on the right of the node
struct node* insertRight(struct node* root, int value) {
  root->right = createNode(value);
  return root->right;
}

int main() {
  struct node* root = createNode(1);
  insertLeft(root, 2);
  insertRight(root, 3);
  insertLeft(root->left, 4);

  printf("Inorder traversal \n");
  inorderTraversal(root);

  printf("\nPreorder traversal \n");
  preorderTraversal(root);

  printf("\nPostorder traversal \n");
  postorderTraversal(root);
}
```