1.Array implementation of List

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

void create();
void insert();
void deletion();
void search();
void display();
int a,b[20], n, p, e, f, i, pos;

void main()
{
//clrscr();
int ch;
char g='y';

do
{
printf("\n main Menu");
printf("\n 1.Create \n 2.Delete \n 3.Search \n 4.Insert \n 5.Display\n 6.Exit \n");
printf("\n Enter your Choice");
scanf("%d", &ch);

switch(ch)
{
case 1:
create();
break;

case 2:
deletion();
break;

case 3:
search();
break;

case 4:
insert();
break;

case 5:
display();
break;

case 6:
```

```
exit();
break;

default:
printf("\n Enter the correct choice:");
}
printf("\n Do u want to continue:::");
scanf("\n%c", &g);
}
while(g=='y'||g=='Y');
getch();
}


void create()
{
printf("\n Enter the number of nodes");
scanf("%d", &n);
for(i=0;i<n;i++)
{
printf("\n Enter the  Element:",i+1);
scanf("%d", &b[i]);
}

}

void deletion()
{
printf("\n Enter the position u want to delete::");
scanf("%d", &pos);
if(pos>=n)
{
printf("\n Invalid Location::");
}
else
{
for(i=pos+1;i<n;i++)
{
b[i-1]=b[i];
}
n--;
}
printf("\n The Elements after deletion");
for(i=0;i<n;i++)
{
printf("\t%d", b[i]);
}
}
```

```
void search()
{
printf("\n Enter the Element to be searched:");
scanf("%d", &e);

for(i=0;i<n;i++)
{
if(b[i]==e)
{
printf("Value is in the %d Position", i);
}
else
{
printf("Value %d is not in the list::", e);
continue;
}
}
}


void insert()
{
printf("\n Enter the position u need to insert::");
scanf("%d", &pos);

if(pos>=n)
 {
 printf("\n invalid Location::");
 }
 else
 {
 for(i=MAX-1;i>=pos-1;i--)
 {
 b[i+1]=b[i];
 }
 printf("\n Enter the element to insert::\n");
 scanf("%d",&p);
 b[pos]=p;
 n++;
 }
 printf("\n The list after insertion::\n");

display();
}
```

```
void display()
{
printf("\n The Elements of The list ADT are:");
for(i=0;i<n;i++)
{
printf("\n\n%d", b[i]);
}
}
```

2. C PROGRAM TO IMPLEMENT STACK OPERATION
1.PUSH 2. POP 3.DISPLAY USING ARRAY */

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#define size 5
int item;
int s[10];
int top;
void display()
{
int i;
if(top==-1)
{
printf("\nstack is empty");
return;
}
printf("\nContent of stack is:\n");
for(i=0;i<=top;i++)
printf("%d\t",s[i]);
}
void push()
{
if(top==size-1)
{
printf("\nStack is full");
return;
}
printf("\nEnter item:\n");
scanf("%d",&item);
s[++top]=item;
}
void pop()
{
if(top==-1)
{
printf("\nstack is empty");
return;
```

```
}
printf("\nDeleted item is: %d",s[top]);
top--;
}

void main()
{
int ch;
top=-1;
clrscr();

printf("\n1.push\t\t2.pop\n3.display\t4.exit\n");
do{
printf("\nEnter your choice:\n");
scanf("%d",&ch);
switch(ch)
{
case 1:// printf("Enter item:\n");
//scanf("%d",&item);
push();
break;
case 2: pop();
break;
case 3: display();
break;
case 4: exit(0);
default: printf("\nWrong entry ! try again");
}}while(ch<=4);


getch();
}
```

C Program of Queue using Array..

```
#include<stdio.h>
#include<conio.h>
#define SIZE 5            /* Size of Queue */
int Q[SIZE],f=0,r=-1;       /* Global declarations */

Qinsert(int elem)
{                       /* Function for Insert operation */
   if( Qfull())
printf("\n\n Overflow!!!!\n\n");
   else
   {
       ++r;
       Q[r]=elem;
   }
```

```
}

int Qdelete()
{                    /* Function for Delete operation */
   int elem;
   if(Qempty()){ printf("\n\nUnderflow!!!!\n\n");
   return(-1); }
   else
   {
      elem=Q[f];
      f=f+1;
      return(elem);
   }
}

int Qfull()
{                    /* Function to Check Queue Full */
   if(r==SIZE-1) return 1;
   return 0;
}

int Qempty()
{                   /* Function to Check Queue Empty */
   if(f > r) return 1;
   return 0;
}

display()
{                /* Function to display status of Queue */
   int i;
   if(Qempty()) printf(" \n Empty Queue\n");
   else
   {
      printf("Front->");
      for(i=f;i<=r;i++)
         printf("%d ",Q[i]);
      printf("<-Rear");
   }
}

void main()
{                      /* Main Program */
   int opn,elem;
   do
   {
      clrscr();
      printf("\n ### Queue Operations using Arrays### \n\n");
      printf("\n Press 1-Insert, 2-Delete,3-Display,4-Exit\n");
```

```
      printf("\n Your option ? ");
      scanf("%d",&opn);
      switch(opn)
      {
      case 1: printf("\n\nRead the element to be Inserted ?");
         scanf("%d",&elem);
         Qinsert(elem); break;
      case 2: elem=Qdelete();
         if( elem != -1)
            printf("\n\nDeleted Element is %d \n",elem);
         break;
      case 3: printf("\n\nStatus of Queue\n\n");
         display(); break;
      case 4: printf("\n\n Terminating \n\n"); break;
      default: printf("\n\nInvalid Option !!! Try Again !! \n\n");
         break;
      }
      printf("\n\n\n\n  Press a Key to Continue . . . ");
      getch();
   }while(opn != 4);
getch();
}
```

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

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

void push(int);
void pop();
void display();

void main()
{
```

```
   int choice, value;
   clrscr();
   printf("\n:: Stack using Linked List ::\n");
   while(1){
      printf("\n****** MENU ******\n");
      printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");
      printf("Enter your choice: ");
      scanf("%d",&choice);
      switch(choice){
         case 1: printf("Enter the value to be insert: ");
                 scanf("%d", &value);
                 push(value);
                 break;
         case 2: pop(); break;
         case 3: display(); break;
         case 4: exit(0);
         default: printf("\nWrong selection!!! Please try again!!!\n");
      }
   }
}
void push(int value)
{
   struct Node *newNode;
   newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   if(top == NULL)
      newNode->next = NULL;
   else
      newNode->next = top;
   top = newNode;
   printf("\nInsertion is Success!!!\n");
}
void pop()
{
   if(top == NULL)
```

```
      printf("\nStack is Empty!!!\n");
   else{
      struct Node *temp = top;
      printf("\nDeleted element: %d", temp->data);
      top = temp->next;
      free(temp);
   }
}
void display()
{
   if(top == NULL)
      printf("\nStack is Empty!!!\n");
   else{
      struct Node *temp = top;
      while(temp->next != NULL){
         printf("%d--->",temp->data);
         temp = temp -> next;
      }
      printf("%d--->NULL",temp->data);
   }
}
```

```
C Program to Implement Queue Data Structure using Linked List
 */
#include <stdio.h>
#include <stdlib.h>

struct node
{
    int info;
    struct node *ptr;
}*front,*rear,*temp,*front1;

int frontelement();
void enq(int data);
void deq();
void empty();
```

```
void display();
void create();
void queuesize();

int count = 0;

void main()
{
    int no, ch, e;

    printf("\n 1 - Enque");
    printf("\n 2 - Deque");
    printf("\n 3 - Front element");
    printf("\n 4 - Empty");
    printf("\n 5 - Exit");
    printf("\n 6 - Display");
    printf("\n 7 - Queue size");
    create();
    while (1)
    {
        printf("\n Enter choice : ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            printf("Enter data : ");
            scanf("%d", &no);
            enq(no);
            break;
        case 2:
            deq();
            break;
        case 3:
            e = frontelement();
            if (e != 0)
                printf("Front element : %d", e);
            else
                printf("\n No front element in Queue as queue is empty");
            break;
        case 4:
            empty();
            break;
        case 5:
            exit(0);
        case 6:
            display();
            break;
        case 7:
            queuesize();
            break;
        default:
            printf("Wrong choice, Please enter correct choice  ");
            break;
        }
    }
}

/* Create an empty queue */
```

```c
void create()
{
    front = rear = NULL;
}

/* Returns queue size */
void queuesize()
{
    printf("\n Queue size : %d", count);
}

/* Enqueing the queue */
void enq(int data)
{
    if (rear == NULL)
    {
        rear = (struct node *)malloc(1*sizeof(struct node));
        rear->ptr = NULL;
        rear->info = data;
        front = rear;
    }
    else
    {
        temp=(struct node *)malloc(1*sizeof(struct node));
        rear->ptr = temp;
        temp->info = data;
        temp->ptr = NULL;

        rear = temp;
    }
    count++;
}

/* Displaying the queue elements */
void display()
{
    front1 = front;

    if ((front1 == NULL) && (rear == NULL))
    {
        printf("Queue is empty");
        return;
    }
    while (front1 != rear)
    {
        printf("%d ", front1->info);
        front1 = front1->ptr;
    }
    if (front1 == rear)
        printf("%d", front1->info);
}

/* Dequeing the queue */
void deq()
{
    front1 = front;

    if (front1 == NULL)
```

```
    {
        printf("\n Error: Trying to display elements from empty queue");
        return;
    }
    else
        if (front1->ptr != NULL)
        {
            front1 = front1->ptr;
            printf("\n Dequed value : %d", front->info);
            free(front);
            front = front1;
        }
        else
        {
            printf("\n Dequed value : %d", front->info);
            free(front);
            front = NULL;
            rear = NULL;
        }
        count--;
}

/* Returns the front element of queue */
int frontelement()
{
    if ((front != NULL) && (rear != NULL))
        return(front->info);
    else
        return 0;
}

/* Display if queue is empty or not */
void empty()
{
     if ((front == NULL) && (rear == NULL))
        printf("\n Queue empty");
    else
       printf("Queue not empty");
}
$ cc pgm4.c
$ a.out

1 - Enque
2 - Deque
3 - Front element
4 - Empty
5 - Exit
6 - Display
7 - Queue size
Enter choice : 1
Enter data : 14

Enter choice : 1
Enter data : 85

Enter choice : 1
Enter data : 38
```

```
Enter choice : 3
Front element : 14
Enter choice : 6
14 85 38
Enter choice : 7

Queue size : 3
Enter choice : 2

Dequed value : 14
Enter choice : 6
85 38
Enter choice : 7

Queue size : 2
Enter choice : 4
Queue not empty
Enter choice : 5
```

```c
#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");
        }

    }
```

Copy

OUTPUT:

```
Enter the size of STACK[MAX=100]:10
```

```
         STACK OPERATIONS USING ARRAY
        --------------------------------
         1.PUSH
         2.POP
         3.DISPLAY
         4.EXIT
 Enter the Choice:1
 Enter a value to be pushed:12

 Enter the Choice:1
 Enter a value to be pushed:24

 Enter the Choice:1
 Enter a value to be pushed:98

 Enter the Choice:3

 The elements in STACK

98
24
12
 Press Next Choice
 Enter the Choice:2

         The popped elements is 98
 Enter the Choice:3

 The elements in STACK

24
12
 Press Next Choice
 Enter the Choice:4

         EXIT POINT
```

```c
#include<stdio.h>
#define n 5
int main()
{
    int queue[n],ch=1,front=0,rear=0,i,j=1,x=n;
    printf("Queue using Array");
    printf("\n1.Insertion \n2.Deletion \n3.Display \n4.Exit");
    while(ch)
    {
        printf("\nEnter the Choice:");
        scanf("%d",&ch);
```

```
        switch(ch)
        {
        case 1:
            if(rear==x)
                printf("\n Queue is Full");
            else
            {
                printf("\n Enter no %d:",j++);
                scanf("%d",&queue[rear++]);
            }
            break;
        case 2:
            if(front==rear)
            {
                printf("\n Queue is empty");
            }
            else
            {
                printf("\n Deleted Element is %d",queue[front++]);
                x++;
            }
            break;
        case 3:
            printf("\nQueue Elements are:\n ");
            if(front==rear)
                printf("\n Queue is Empty");
            else
            {
                for(i=front; i<rear; i++)
                {
                    printf("%d",queue[i]);
                    printf("\n");
                }
                break;
            case 4:
                exit(0);
            default:
                printf("Wrong Choice: please see the options");
```

```
            }
        }
    }
    return 0;
}
```

Copy

OUTPUT:

```
Queue using Array
1.Insertion
2.Deletion
3.Display
4.Exit
Enter the Choice:1

 Enter no 1:10

Enter the Choice:1

 Enter no 2:54

Enter the Choice:1

 Enter no 3:98

Enter the Choice:1

 Enter no 4:234

Enter the Choice:3

 Queue Elements are:
 10
54
98
234

Enter the Choice:2

 Deleted Element is 10
Enter the Choice:3

 Queue Elements are:
 54
98
234

Enter the Choice:4
```

Application of stack infix to prefix conversion

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

// A structure to represent a stack
struct Stack {
   int top;
   int maxSize;
   // we are storing string in integer array, this will not give error
   // as values will be stored in ASCII and returned in ASCII thus, returned as string again
   int* array;
};

struct Stack* create(int max)
{
   struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
   stack->maxSize = max;
   stack->top = -1;
   stack->array = (int*)malloc(stack->maxSize * sizeof(int));
   return stack;
}

// Checking with this function is stack is full or not
// Will return true is stack is full else false
//Stack is full when top is equal to the last index
```

```
int isFull(struct Stack* stack)
{
   if(stack->top == stack->maxSize - 1){
      printf("Will not be able to push maxSize reached\n");
   }
   // Since array starts from 0, and maxSize starts from 1
   return stack->top == stack->maxSize - 1;
}

// By definition the Stack is empty when top is equal to -1
// Will return true if top is -1
int isEmpty(struct Stack* stack)
{
   return stack->top == -1;
}

// Push function here, inserts value in stack and increments stack top by 1
void push(struct Stack* stack, int item)
{
   if (isFull(stack))
      return;
   stack->array[++stack->top] = item;
}

// Function to remove an item from stack.  It decreases top by 1
int pop(struct Stack* stack)
{
```

```
    if (isEmpty(stack))
        return INT_MIN;
    return stack->array[stack->top--];
}

// Function to return the top from stack without removing it
int peek(struct Stack* stack)
{
    if (isEmpty(stack))
        return INT_MIN;
    return stack->array[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;

    // Stack size should be equal to expression size for safety
    struct Stack* stack = create(strlen(expression));
    if(!stack) // just checking is stack was created or not
        return -1 ;

    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(stack, 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(stack, 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)";
    printf("%s\n",expression);
```

```
    InfixtoPrefix(expression);

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


/*C program for different tree traversals */
#include <stdio.h>
#include <stdlib.h>

/* A binary tree node has data, pointer to left child
   and a pointer to right child */
struct node {
    int data;
    struct node* left;
    struct node* right;
};

/* Helper function that allocates a new node with the
   given data and NULL left and right pointers. */
struct node* newNode(int data)
{
    struct node* node
        = (struct node*)malloc(sizeof(struct node));
    node->data = data;
    node->left = NULL;
```

```
    node->right = NULL;

    return (node);
}

/* Given a binary tree, print its nodes according to the
  "bottom-up" postorder traversal. */
void printPostorder(struct node* node)
{
    if (node == NULL)
        return;

    // first recur on left subtree
    printPostorder(node->left);

    // then recur on right subtree
    printPostorder(node->right);

    // now deal with the node
    printf("%d ", node->data);
}

/* Given a binary tree, print its nodes in inorder*/
void printInorder(struct node* node)
{
    if (node == NULL)
        return;
```

```
    /* first recur on left child */
    printInorder(node->left);

    /* then print the data of node */
    printf("%d ", node->data);

    /* now recur on right child */
    printInorder(node->right);
}

/* Given a binary tree, print its nodes in preorder*/
void printPreorder(struct node* node)
{
    if (node == NULL)
        return;

    /* first print data of node */
    printf("%d ", node->data);

    /* then recur on left sutree */
    printPreorder(node->left);

    /* now recur on right subtree */
    printPreorder(node->right);
}
```

```
/* Driver program to test above functions*/
int main()
{
   struct node* root = newNode(1);
   root->left = newNode(2);
   root->right = newNode(3);
   root->left->left = newNode(4);
   root->left->right = newNode(5);

   printf("\nPreorder traversal of binary tree is \n");
   printPreorder(root);

   printf("\nInorder traversal of binary tree is \n");
   printInorder(root);

   printf("\nPostorder traversal of binary tree is \n");
   printPostorder(root);

   getchar();
   return 0;
}
```