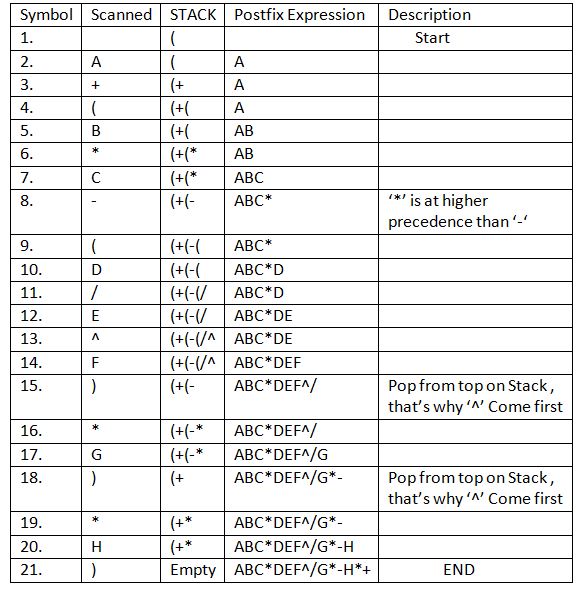
**Experiment 1**

**Aim: Implementation of Infix to postfix expression-Transformation and its evaluation program.**

**Algorithm:**

1. Let, X is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression Y.
2. Push “(“onto Stack, and add “)” to the end of X.
3. Scan X from left to right and repeat Step 3 to 6 for each element of X until the Stack is empty.
4. If an operand is encountered, add it to Y.
5. If a left parenthesis is encountered, push it onto Stack.
6. If an operator is encountered ,then:
7. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) which has the same precedence as or higher precedence than operator.
8. Add operator to Stack.  
   [End of If]
9. If a right parenthesis is encountered ,then:
10. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) until a left parenthesis is encountered.
11. Remove the left Parenthesis.  
    [End of If]  
    [End of If]
12. END.

**Example:**

**Code(infix to postfix):**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

// program to convert infix to postfix

int top = -1;

char stack[100];

void push(char a)

{

top++;

stack[top] = a;

}

char pop()

{

if (top == -1)

return -1;

else

return stack[top--];

}

int prior(char a)

{

if (a == '(')

return 0;

if (a == '+' || a == '-')

return 1;

if (a == '\*' || a == '/')

return 2;

if (a == '^')

return 3;

return 0;

}

int main()

{

char expr[100];

char x;

int i;

clrscr();

printf("Enter your exprression: ");

scanf("%s", expr);

for (i = 0; i < strlen(expr); i++)

{

if (isalnum(expr[i]))

{

printf("%c ", expr[i]);

}

else

{

if (expr[i] == '(')

push(expr[i]);

else if (expr[i] == ')')

{

while ((x = pop()) != '(')

{

printf("%c ", x);

}

}

else

{

while (prior(expr[i]) <= prior(stack[top]))

printf("%c ", pop());

push(expr[i]);

}

}

}

while (top != -1)

{

printf("%c ", pop());

}

getch();

return 0;

}

**Output:**

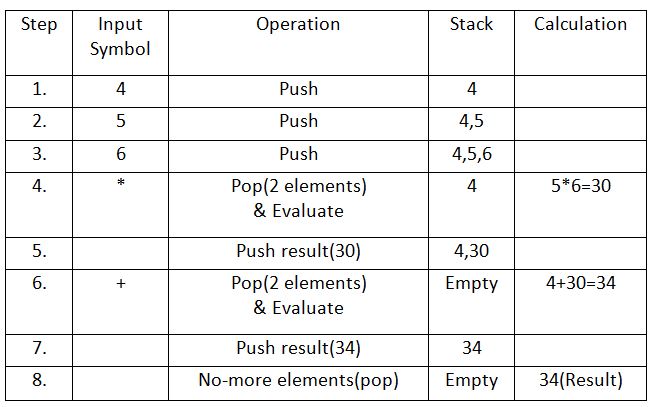
****

**Algorithm:**

1. Create a stack that holds integer type data to store the operands of the given postfix expression. Let it be st.
2. Iterate over the string from left to right and do the following -

* If the current element is an operand, push it into the stack.
* Otherwise, if the current element is an operator (say /)do the following -
* Pop an element from st, let it be op2.
* Pop another element from st, let it be op1.
* Computer the result of op1 / op2, and push it into the stack. Note the order i.e.i.e. op1 / op2 should not be changed otherwise it will affect the final result in some cases.

1. At last, st will consist of a single element i.e. the result after evaluating the postfix expression.

**Example:**

**Code (evaluation):**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

// program to evaluate postfix expression

int top = -1;

char stack[100];

void push(char a)

{

top++;

stack[top] = a;

}

char pop()

{

if (top == -1)

return -1;

else

return stack[top--];

}

int main()

{

char expr[100];

int i,a,b,c;

clrscr()

printf("Enter any postfix exprression: ");

scanf("%s", expr);

for (i = 0; i < strlen(expr); i++)

{

if (isdigit(expr[i]))

{

push(expr[i]-48);

}

else

{

a = pop();

b = pop();

switch(expr[i])

{

case '+':

{

c = a + b;

break;

}

case '-':

{

c = b - a;

break;

}

case '\*':

{

c = a \* b;

break;

}

case '/':

{

c = b / a;

break;

}

case '%':

c = b / a;

}

push(c);

}

}

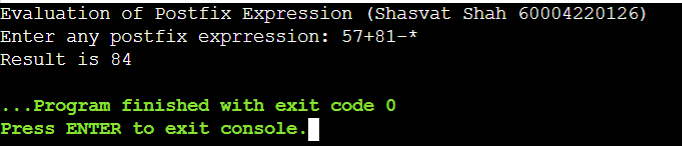
printf("Result is %d",pop());

getch();

return 0;

}

**Output:**



**Experiment 2**

**Aim: Perform Insertion Deletion, Sorting, Searching And Traversal in Linear Link List.**

**Algorithm:**

**Insertion at the beginning:**

1. Allocate memory for new node
2. Store data
3. Change next of new node to point to head
4. Change head to point to recently created node

struct node \*newNode;

newNode = malloc(sizeof(struct node));

newNode->data = 4;

newNode->next = head;

head = newNode;

**Insertion at the End:**

1. Allocate memory for new node
2. Store data
3. Traverse to last node
4. Change next of last node to recently created node

struct node \*newNode;

newNode = malloc(sizeof(struct node));

newNode->data = 4;

newNode->next = NULL;

struct node \*temp = head;

while(temp->next != NULL){

temp = temp->next;

}

temp->next = newNode;

**Insertion at the Middle:**

1. Allocate memory and store data for new node
2. Traverse to node just before the required position of new node
3. Change next pointers to include new node in between

struct node \*newNode;

newNode = malloc(sizeof(struct node));

newNode->data = 4;

struct node \*temp = head;

for(int i=2; i < position; i++) {

if(temp->next != NULL) {

temp = temp->next;

}

}

newNode->next = temp->next;

temp->next = newNode;

**Delete from beginning:**

1. IF HEAD = NULL

Write UNDERFLOW

Go to Step 5

1. [END OF IF]
2. SET PTR = HEAD
3. SET HEAD = HEAD -> NEXT
4. FREE PTR
5. EXIT

**Delete from end:**

1. IF HEAD = NULL

Write UNDERFLOW  
   Go to Step 8  
  [END OF IF]

1. SET PTR = HEAD
2. Repeat Steps 4 and 5 while PTR -> NEXT!= NULL
3. SET PREPTR = PTR
4. SET PTR = PTR -> NEXT
5. [END OF LOOP]
6. SET PREPTR -> NEXT = NULL
7. FREE PTR
8. EXIT

**Delete from middle**

1. IF HEAD = NULL

WRITE UNDERFLOW

GOTO STEP 10

END OF IF

1. SET TEMP = HEAD
2. SET I = 0
3. REPEAT STEP 5 TO 8 UNTIL I
4. TEMP1 = TEMP
5. TEMP = TEMP → NEXT
6. IF TEMP = NULL

WRITE "DESIRED NODE NOT PRESENT"

GOTO STEP 12

END OF IF

1. I = I+1
2. END OF LOOP
3. TEMP1 → NEXT = TEMP → NEXT
4. FREE TEMP
5. EXIT

**Traversal:**

1. SET PTR = HEAD
2. IF PTR = NULL

WRITE "EMPTY LIST"

GOTO STEP 7

END OF IF

1. REPEAT STEP 5 AND 6 UNTIL PTR != NULL
2. PRINT PTR→ DATA
3. PTR = PTR → NEXT
4. [END OF LOOP]
5. EXIT

**Searching:**

1. SET PTR = HEAD
2. Set I = 0
3. IF PTR = NULL

WRITE "EMPTY LIST"

GOTO STEP 8

END OF IF

1. REPEAT STEP 5 TO 7 UNTIL PTR != NULL
2. if ptr → data = item

write i+1

End of IF

1. I = I + 1
2. PTR = PTR → NEXT
3. [END OF LOOP]
4. EXIT

**Sorting:**

1. Make the head as the current node and create another node index for later use.
2. If head is null, return.
3. Else, run a loop till the last node (i.e. NULL).
4. In each iteration, follow the following step 5-6.
5. Store the next node of current in index.
6. Check if the data of the current node is greater than the next node. If it is greater, swap current and index.

**Code:**

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \*next;

};

int pos;

struct node \*newnode, \*start=0, \*temp, \*prevtemp;

void create(){

newnode = (struct node\*)malloc(sizeof(struct node));

printf("\n Enter the data : ");

scanf("%d", &newnode -> data);

if(start == 0){

start = temp = newnode;

newnode -> next = 0;

}

else{

temp = start;

while(temp -> next != 0){

temp = temp -> next;

}

temp -> next = newnode;

newnode -> next = 0;

}

}

void insertatbeg(){

newnode = (struct node\*)malloc(sizeof(struct node));

printf("\n Enter the data : ");

scanf("%d", &newnode -> data);

newnode -> next = start;

start = newnode;

}

void insertatend(){

newnode = (struct node\*)malloc(sizeof(struct node));

printf("\n Enter the data : ");

scanf("%d", &newnode -> data);

temp = start;

while(temp -> next != 0){

temp = temp -> next;

}

temp -> next = newnode;

newnode -> next = 0;

}

void insertatloc(){

newnode = (struct node\*)malloc(sizeof(struct node));

printf("\n Enter the data : ");

scanf("%d", &newnode -> data);

printf("\n Enter position for insertion : ");

scanf("%d", &pos);

temp = start;

for(int i = 0; i < pos ; i++){

prevtemp = temp;

temp = temp-> next;

}

newnode -> next = prevtemp -> next;

prevtemp -> next = newnode;

}

void deleteatbeg(){

start = start -> next;

}

void deleteatend(){

temp = start;

while(temp -> next != 0){

prevtemp = temp;

temp = temp -> next;

}

prevtemp -> next = 0;

}

void deleteatloc(){

printf("\n Enter position for deletion : ");

scanf("%d", &pos);

temp = start;

for(int i = 0; i < pos; i++){

prevtemp = temp;

temp = temp ->next;

}

prevtemp -> next = temp -> next;

}

void display(){

temp = start;

int count = 0;

while (temp != 0){

printf("%d \t", temp -> data);

temp = temp -> next;

count++;

}

printf("\nCount is : %d", count);

}

int main()

{

int choice;

printf("Singly Linked List (Shasvat Shah - 60004220126)");

while(1){

printf("\n 1. Create ");

printf("\n 2. Insert at Beginning ");

printf("\n 3. Insert at End ");

printf("\n 4. Insert at Location ");

printf("\n 5. Delete from Beginning ");

printf("\n 6. Delete from End ");

printf("\n 7. Delete from Location ");

printf("\n 8. Display ");

printf("\n 9. Exit ");

printf("\n Choose from the abovce given numbers : ");

scanf("%d", &choice);

switch(choice){

case 1:

create();

break;

case 2:

insertatbeg();

break;

case 3:

insertatend();

break;

case 4:

insertatloc();

break;

case 5:

deleteatbeg();

break;

case 6:

deleteatend();

break;

case 7:

deleteatloc();

break;

case 8:

display();

break;

case 9:

exit(0);

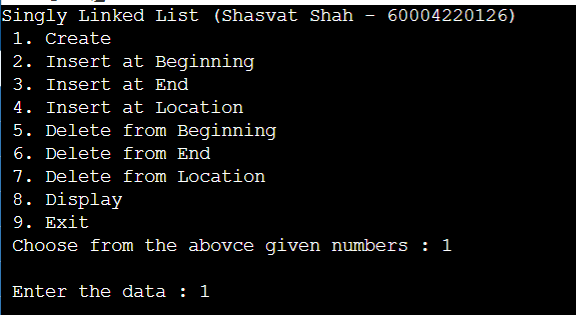
}

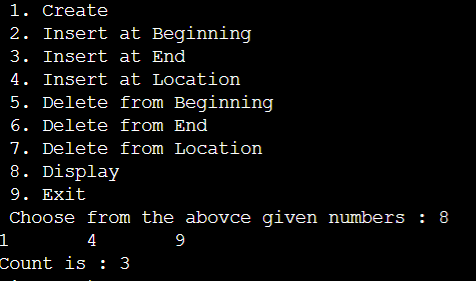
}

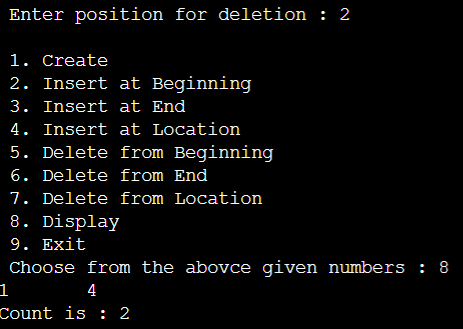
return 0;

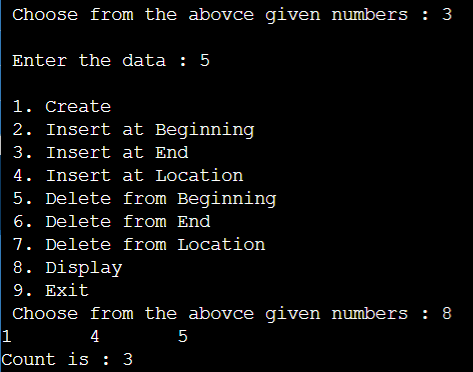
}

**Output:**









**Experiment 3**

Q1) Implementation of Stack and queue using doubly linked List.

STACK

**Algorithm:**

**Push()**

1. If the stack is empty then take a new node, add data to it and assign “null” to its previous and next pointer as it is the first node of the DLL.
2. Assign top and start as the new node. Otherwise, take a new node, add data to it and assign the “previous” pointer of the new node to the “top” node earlier and next as “null”.
3. Further, update the “top” pointer to hold the value of the new node as that will be the top element of the stack now.

**Pop()**

1. If the stack is empty, then print that stack is empty, Otherwise, assign top ->prev -> next as “null” and assign top as top->prev.

**PritnStack()**

1. If the stack is empty, then print that stack is empty. Otherwise, traverse the doubly linked list from start to end and print the data of each node.

**Code:**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct node {**

**int data;**

**struct node \*next;**

**struct node \*prev;**

**};**

**struct node \*newnode, \*top = 0, \*temp;**

**void push(){**

**newnode = (struct node\*)malloc(sizeof(struct node));**

**printf("\n Enter the data : ");**

**scanf("%d", &newnode -> data);**

**if(top == 0){**

**top = newnode;**

**newnode -> next = 0;**

**newnode -> prev = 0;**

**}**

**else{**

**newnode -> next = top;**

**top -> prev = newnode;**

**newnode -> prev = 0;**

**top = newnode;**

**}**

**}**

**void pop(){**

**if(top == 0){**

**printf("Stack is Empty");**

**}**

**else{**

**top = top -> next;**

**top -> prev = 0;**

**}**

**}**

**void peep(){**

**if(top == 0){**

**printf("Stack is Empty");**

**}**

**else{**

**printf("The topmost element of the stack is : %d",top->data);**

**}**

**}**

**void display(){**

**temp = top;**

**int count = 0;**

**while (temp != 0){**

**printf("%d \t", temp -> data);**

**temp = temp -> next;**

**count++;**

**}**

**printf("\nCount is : %d", count);**

**}**

**int main()**

**{**

**int choice;**

**printf("Stack using Doubly Linked List (Shasvat Shah - 60004220126)");**

**while(1){**

**printf("\n 1. Push ");**

**printf("\n 2. Pop ");**

**printf("\n 3. Peep ");**

**printf("\n 4. Display ");**

**printf("\n 5. Exit ");**

**printf("\n Choose from the abovce given numbers : ");**

**scanf("%d", &choice);**

**switch(choice){**

**case 1:**

**push();**

**break;**

**case 2:**

**pop();**

**break;**

**case 3:**

**peep();**

**break;**

**case 4:**

**display();**

**break;**

**case 5:**

**exit(0);**

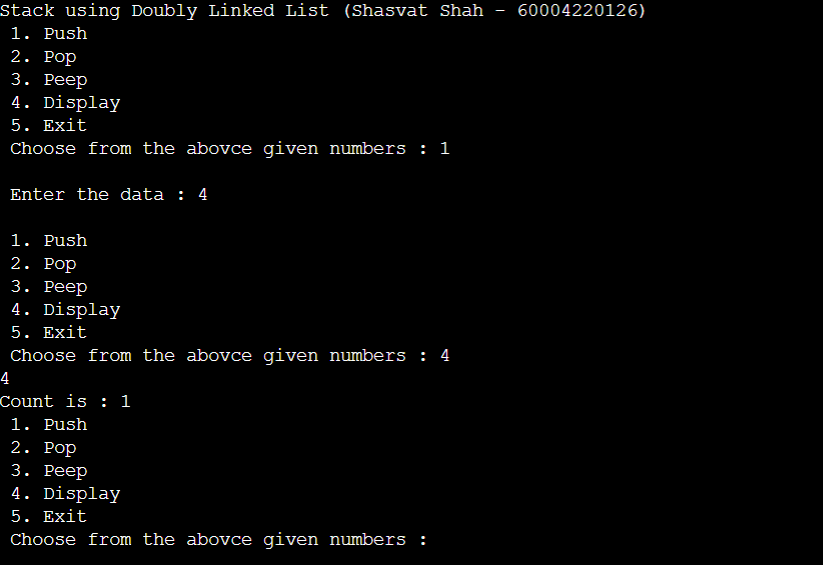
**}**

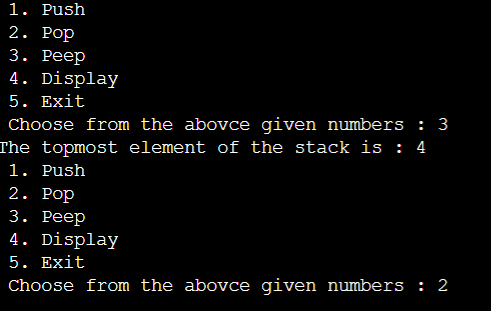
**}**

**return 0;**

**}**

**Output:**

****

****

QUEUE:

**Algortihm:**

**Insertion at front end**

1. Allocate space for a **newNode** of doubly linked list.
2. IF newNode == NULL, then
3. print "Overflow"
4. ELSE
5. IF front == NULL, then
6. rear = front = newNode
7. ELSE
8. newNode->next = front
9. front->prev = newNode
10. front = newNode

**Insertion at the end.**

1. IF newNode == NULL, then
2. print "Overflow"
3. ELSE
4. IF rear == NULL, then
5. front = rear = newNode
6. ELSE
7. newNode->prev = rear
8. rear->next = newNode
9. rear = newNode

**Deletion from front end : .**

1. IF front == NULL
2. print "Underflow"
3. ELSE
4. Initialize temp = front
5. front = front->next
6. IF front == NULL
7. rear = NULL
8. ELSE
9. front->prev = NULL
10. Deallocate space for temp

**Deletion from Rear end :**

1. IF front == NULL
2. print "Underflow"
3. ELSE
4. Initialize temp = rear
5. rear = rear->prev
6. IF rear == NULL
7. front = NULL
8. ELSE
9. rear->next = NULL
10. Deallocate space for temp

**Code:**

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \*next;

struct node \*prev;

};

struct node \*front = 0, \*rear = 0, \*newnode, \*temp;

void enqueue(){

newnode = (struct node\*)malloc(sizeof(struct node));

printf("\n Enter the data : ");

scanf("%d", &newnode -> data);

if(front == 0 && rear == 0){

front = rear = newnode;

newnode -> next = 0;

newnode -> prev = 0;

}

else{

rear -> next = newnode;

newnode -> prev = rear;

rear = newnode;

newnode -> next = 0;

}

}

void dequeue(){

if(front == 0 && rear == 0){

printf("\n Queue is empty...");

}

else{

temp = front;

front = front -> next;

front -> prev = 0;

free(temp);

}

}

void peep(){

if(front == 0 && rear == 0){

printf("\n Queue is empty...");

}

else{

printf("\n The first element of the queue is : %d", front->data);

}

}

void display(){

temp = front;

while(temp != 0){

printf("%d \t",temp -> data);

temp = temp -> next;

}

}

int main()

{

int choice;

printf("Queue using Doubly Linked List (Shasvat Shah - 60004220126)");

while(1){

printf("\n 1. Enqueue ");

printf("\n 2. Dequeue ");

printf("\n 3. Peep ");

printf("\n 4. Display ");

printf("\n 5. Exit ");

printf("\n Choose from the abovce given numbers : ");

scanf("%d", &choice);

switch(choice){

case 1:

enqueue();

break;

case 2:

dequeue();

break;

case 3:

peep();

break;

case 4:

display();

break;

case 5:

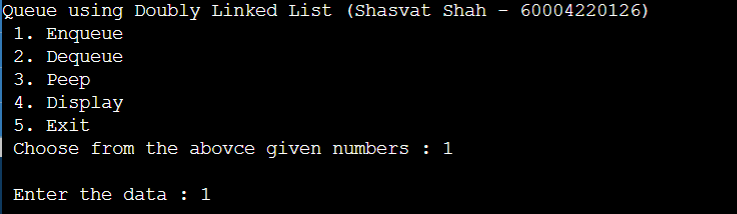
exit(0);

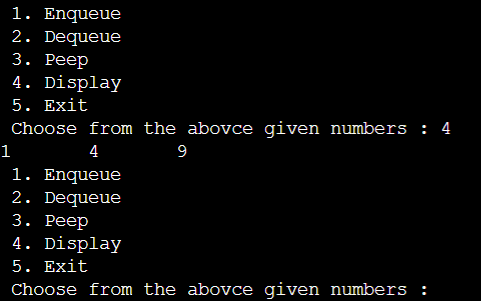
}

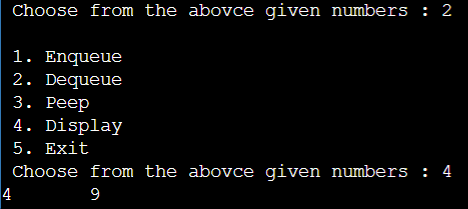
}

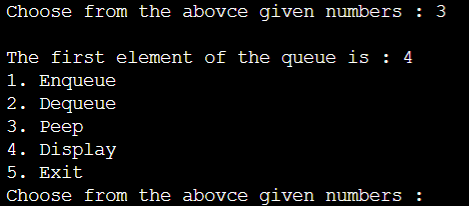
return 0;

}**Output:**

****

****

****

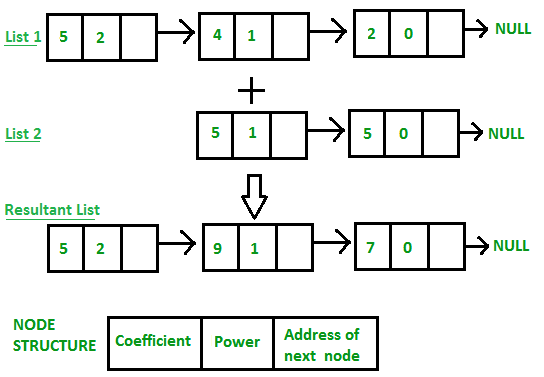
****

**Experiment 4**

**Aim: Implementation of Polynomial addition using linked list.**

**Algorithm:**

1. Create a new linked list, newHead to store the resultant list.
2. Traverse both lists until one of them is null.
3. If any list is null insert the remaining node of another list in the resultant list.
4. Otherwise compare the degree of both nodes, a (first list as node) and b (second list as node). Here three cases are possible:
   * If the degree of a and b is equal, we insert a new node in the resultant list with the coefficient equal to the sum of coefficients of a and b and the same degree.
   * If the degree of a is greater than b, we insert a new node in the resultant list with the coefficient and degree equal to that of a.
   * If the degree of b is greater than a, we insert a new node in the resultant list with the coefficient and degree equal to that of b.

**Example:**

**Code:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int coeff;

int pow;

struct Node\* next;

};

void newnode(int x, int y, struct Node\*\* temp)

{

struct Node \*r, \*z;

z = \*temp;

if (z == NULL) {

r = (struct Node\*)malloc(sizeof(struct Node));

r->coeff = x;

r->pow = y;

\*temp = r;

r->next = (struct Node\*)malloc(sizeof(struct Node));

r = r->next;

r->next = NULL;

}

else {

r->coeff = x;

r->pow = y;

r->next = (struct Node\*)malloc(sizeof(struct Node));

r = r->next;

r->next = NULL;

}

}

void polyadd(struct Node\* poly1, struct Node\* poly2,

struct Node\* poly)

{

while (poly1->next && poly2->next) {

if (poly1->pow > poly2->pow) {

poly->pow = poly1->pow;

poly->coeff = poly1->coeff;

poly1 = poly1->next;

}

else if (poly1->pow < poly2->pow) {

poly->pow = poly2->pow;

poly->coeff = poly2->coeff;

poly2 = poly2->next;

}

else {

poly->pow = poly1->pow;

poly->coeff = poly1->coeff + poly2->coeff;

poly1 = poly1->next;

poly2 = poly2->next;

}

poly->next

= (struct Node\*)malloc(sizeof(struct Node));

poly = poly->next;

poly->next = NULL;

}

while (poly1->next || poly2->next) {

if (poly1->next) {

poly->pow = poly1->pow;

poly->coeff = poly1->coeff;

poly1 = poly1->next;

}

if (poly2->next) {

poly->pow = poly2->pow;

poly->coeff = poly2->coeff;

poly2 = poly2->next;

}

poly->next

= (struct Node\*)malloc(sizeof(struct Node));

poly = poly->next;

poly->next = NULL;

}

}

void show(struct Node\* node)

{

while (node->next != NULL) {

printf("%dx^%d", node->coeff, node->pow);

node = node->next;

if (node->coeff >= 0) {

if (node->next != NULL)

printf("+");

}

}

}

int main()

{

struct Node \*poly1 = NULL, \*poly2 = NULL, \*poly = NULL;

newnode(1, 2, &poly1);

newnode(1, 1, &poly1);

newnode(1, 0, &poly1);

newnode(3, 1, &poly2);

newnode(7, 0, &poly2);

printf("1st Number: ");

show(poly1);

printf("\n2nd Number: ");

show(poly2);

poly = (struct Node\*)malloc(sizeof(struct Node));

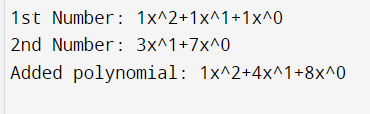
polyadd(poly1, poly2, poly);

printf("\nAdded polynomial: ");

show(poly);

return 0;

}

**Output:**

**Experiment 5**

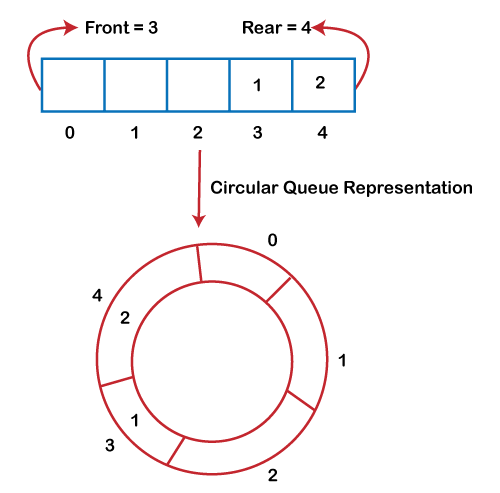
**Aim: Implementation of circular Queue.**

**Algorithm:**

**Enqueue:**

1. IF (REAR+1)%MAX = FRONT
2. Write " OVERFLOW "
3. Goto step 4
4. [End OF IF]
5. IF FRONT = -1 and REAR = -1
6. SET FRONT = REAR = 0
7. ELSE IF REAR = MAX - 1 and FRONT ! = 0
8. SET REAR = 0
9. ELSE
10. SET REAR = (REAR + 1) % MAX
11. [END OF IF]
12. SET QUEUE[REAR] = VAL
13. EXIT

**Dequeue:**

1. IF FRONT = -1
2. Write " UNDERFLOW "
3. Goto Step 4
4. [END of IF]
5. SET VAL = QUEUE[FRONT]
6. IF FRONT = REAR
7. SET FRONT = REAR = -1
8. ELSE
9. IF FRONT = MAX -1
10. SET FRONT = 0
11. ELSE
12. SET FRONT = FRONT + 1
13. [END of IF]
14. [END OF IF]
15. EXIT

**Example:**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#define MAX 5

int queue[MAX];

int front = -1;

int rear = -1;

void enqueue(void);

void dequeue(void);

void peep(void);

void display(void);

int main()

{

int choice;

printf("Queue using Arrays");

while(1){

printf("\nChoose from the options give below");

printf("\n1. Insert element into the queue");

printf("\n2. Delete element from the queue");

printf("\n3. Display the first element of the queue");

printf("\n4. Display the queue");

printf("\n5. Exit\n");

scanf("%d", &choice);

switch(choice){

case 1:

enqueue();

break;

case 2:

dequeue();

break;

case 3:

peep();

break;

case 4:

display();

break;

case 5:

exit(0);

break;

default:

printf("Choose from the options given below.. ");

break;

}

}

return 0;

}

void enqueue(){

int value;

printf("Enter the element to be inserted : ");

scanf("%d", &value);

if(front == (rear+1)%MAX){

printf("Overlfow");

}

else if(front == -1 && rear == -1){

front = rear = 0;

queue[rear] = value;

}

else{

rear = (rear +1) % MAX;

queue[rear] = value;

}

}

void dequeue(){

if(front == -1 && rear == -1){

printf("Queue is empty");

}

else if(front == rear){

front = rear = -1;

}

else{

front = (front +1)%MAX;

}

}

void peep(){

if(front == -1){

printf("Queue is empty");

}

else{

printf("%d", queue[front]);

}

}

void display(){

int i = front;

if(front == -1 || front > rear){

printf("Queue is Empty");

}

while(i != rear){

printf("%d \t", queue[i]);

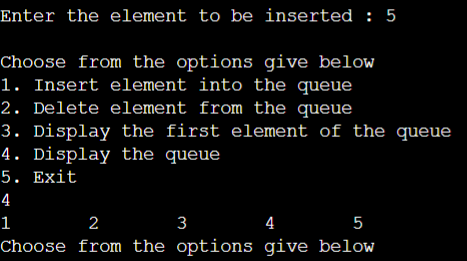
i = (i +1)%MAX;

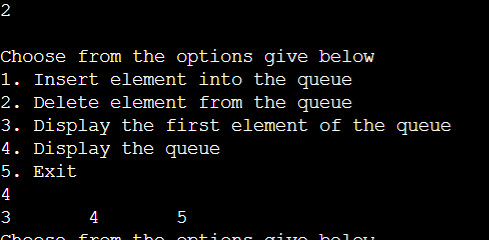
}

printf("%d", queue[i]);

}

**Output:**

****

****