LOKMANYA TILAK COLLEGE OF ENGINEERING

Sector No. 4., Vikas Nagar, Koparkhairane, Navi Mumbai -400 709.

Computer Science and Engineering

(Artificial Intelligence & Machine Learning)

AY 2021-22 (Odd)

(SEM. III)

Practical file

for

Data Structure Lab

(CSL-301)

Name of Student: **SINGH SUDHAM DHARMENDRA**

Roll no. : **AIMLD50**

Faculty Incharge: Prof. Susmita Dutta

INDEX

SR.NO	LIST OF EXPERIMENT
1	Implement Stack using array.
2	Implement Queue using array.
3	Convert Infix expression to Postfix expression.
4	To evaluate Postfix expression using Stack ADT.
5	Implement Singly Linked List.
6	Implement Stack using linked list.
7	Implement Circular queue using Arrays.
8	Implement Queue using linked list.
9	Implement Circular linked list.
10	Implement Binary Search Tree using Linked list.
11	ASSIGNMENT NO.1

NAME: SINGH SUDHAM DHARMENDRA

ROLL NO:AIMLD50

BRANCH: CSE(AIML)

SUBJECT: DATA STRUCTURE

TOPIC: EXPERIMENT NO 1

DATE: 19-09-2021

EXPERIMENT NO. 1

AIM : Write a Program to Implement Stack using Arrays

THEORY : **STACK**

Stack is LIFO (Last-In, First-Out) linear data structure in which elements are inserted and removed only from one end.

Operations on Stack

A stack supports three basic operations: push, pop, and peek.

1. Push Operation

The push operation is used to insert an element into the stack. The new element is added at the topmost position of the stack. Algorithm of Push operation is as follows,

ALGORITHM : PUSH

```
Step 1: IF TOP = MAX-1
           PRINT "OVERFLOW"
        [END OF IF]
Step 2: SET TOP = TOP + 1
Step 3: SET STACK[TOP] = VALUE
Step 4: END
```

In Step 1, we first check for the OVERFLOW condition. In Step 2, TOP is incremented so that it points to the next location in the array. In Step 3, the value is stored in the stack at the location pointed by TOP.

2. Pop Operation

The pop operation is used to delete the topmost element from the stack. Algorithm of Pop operation is as follows,

ALGORITHM : POP

```
Step 1: IF TOP = NULL
           PRINT "UNDERFLOW"
        [END OF IF]
Step 2: SET VAL = STACK[TOP]
Step 3: SET TOP = TOP - 1
Step 4: END
```

In Step 1, we first check for the UNDERFLOW condition. In Step 2, the value of the location in the stack pointed by TOP is stored in VAL. In Step 3, TOP is

decremented.

3. Peek Operation

b. Peek is an operation that returns the value of the topmost element of the stack without deleting it from the stack. Algorithm of Peek operation is as follows,

ALGORITHM : PEEK

```
Step 1: IF TOP = NULL
            PRINT "STACK IS EMPTY"
Step 2: RETURN STACK[TOP]
Step 3: END
```

Peek operation first checks if the stack is empty, i.e., if TOP = NULL, then an appropriate message is printed, else the value is returned.

Assignment:

1)Write algorithm of stack implementation using array
2) PDF format of programming.
3)Answers all the following answers

1.The data structure required to check whether an expression contains balanced parenthesis is?

a) Stack
b) Queue
c) Array
d) Tree

. 2.The process of accessing data stored in a serial access memory is similar to manipulating data on a ------?

a) Heap
b) Binary Tree
c) Array
d) Stack

3.What is the result of the following operation
Top (Push (S, X))

a) X
b) Null
c) S
d) None

4. Which of the following statement(s) about stack data structure is/are NOT correct?

a) Stack data structure can be implemented using linked list

b) New node can only be added at the top of the stack

c) Stack is the FIFO data structure

d) The last node at the bottom of the stack has a NULL link

5.Consider the following operation performed on a stack of size 5.

Push(1);
Pop();
Push(2);
Push(3);
Pop();
Push(4);
Pop();
Pop();
Push(5);

After the completion of all operation, the no of element present on stack are

a) 1

b) 2

c) 3

d) 4

6.Which of the following operation take worst case linear time in the array implementation of stack?

a) Push

b) Pop

c) IsEmpty

d) None

7. The type of expression in which operator succeeds its operands is?

a) Infix Expression

b) pre fix Expression

c) postfix Expression

d) None

8.Which of the following application generally use a stack?

a) Parenthesis balancing program

b) Syntax analyzer in compiler

c) Keeping track of local variables at run time

d) All of the above

9.Consider the following array implementation of stack:

```
#define MAX 10
Struct STACK
{
Int arr [MAX];
```

Int top = -1;
}

If the array index starts with 0, the maximum value of top which does not cause stack overflow is?

a) 8
b) 9
c) 10
d) 11

10.Consider the usual implementation of parentheses balancing program using stack. What is the maximum number of parentheses that will appear on stack at any instance of time during the analysis of (( ) (( )) (( )))?

a) 1
b) 2
c) 3
d) 4

Q.1. Write algorithm of stack implementation using array.

1.**Push**

Step0: START

Step1: Check if the stack is full (top==size-1)

Step2: If it is full, then display "Stack overflow" and terminate the function.

Step3: If it is not full, then increment top value by one (top++) and set stack[top] to value i.e., to be inserted.

Start4: END

2.**Pop**

Step0: START

Step1: Check if the stack is empty (top==-1)

Step2: If the stack is empty, then display "Stack is empty" and terminate the function.

Step3: If the stack is not empty, then delete stack[top] and decrement top value by one (top--).

Step4: END

3.**Peek**

Step0: START

Step1: Check if the stack is empty (top==-1)

Step2: If stack is empty, then display “Stack is empty” and terminate the program.

Step3: If the stack is not empty, then print stack [top].

Step4: END

Program:

C DSexpt.1.c > main()

```c
#include <stdio.h>
int stack[10];
int top = -1, num;
int push(int n)
{
    if (top == 9)
    {
        printf("overflow! Stack is overflow");
    }
    else
    {

        top++;
        stack[top] = n;
    }
}
int pop()
{
    if (top == -1)
    {
        printf("underflow!!!");
    }
    else
    {
        int item = stack[top];
        top--;
        printf("%d is popped out \n", item);
    }
}
int peek()
{
    if (top == -1)
    {
        printf("underflow");
    }
    else
    {
        printf("%d is top of stack \n", stack[top]);
```

DSexpt.1.c > main()

```c
int peek()
{
    if (top == -1)
    {
        printf("underflow");
    }
    else
    {
        printf("%d is top of stock \n", stack[top]);
    }
}

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

   do
    {
        printf("\nEnter your choice\n");
     printf("1:push \n2:pop \n3:peek \n4:display \n0:exit\n");
     scanf("%d", &ch);

     if (ch == 1)
```

```c
    }
}
int main()
{
    int ch;

   do
   {
        printf("\nEnter your choice\n");
    printf("1:push \n2:pop \n3:peek \n4:display \n0:exit\n");
    scanf("%d", &ch);

    if (ch == 1)
    {
        printf("Enter the number:");
        scanf("%d", &num);

        push(num);
    }
    else if (ch == 2)
    {
       pop();
    }
    else if (ch == 3)
    {
        peek();
    }

        else if (ch == 4)
    {
        display();
    }
   }
   while (ch!=0);

    return 0;
}
```

Output:

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

PS C:\Users\duste\.vscode> cd "c:\Users\duste\.vscode\" ; if ($?) { gcc madan.c -o madan } ; if ($?) { .\madan }

Enter your choice
1:push
2:pop
3:peek
4:display
0:exit
1
Enter the number:15

Enter your choice
1:push
2:pop
3:peek
4:display
0:exit
1
Enter the number:30

Enter your choice
1:push
2:pop
3:peek
4:display
0:exit
4
stack is :
30
15

Enter your choice
1:push
2:pop
3:peek
4:display
0:exit
3
30 is top of stock

Enter your choice
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
3:peek
4:display
0:exit
2
30 is popped out

Enter your choice
1:push
2:pop
3:peek
4:display
0:exit
0
PS C:\Users\duste\.vscode>
```

NAME: SINGH SUDHAM DHARMENDRA

ROLL NO.: AIMLD50

BRANCH: CSE (AIML)

SUBJECT: DATA STRUCTURE

TOPIC: EXPERIMENT NO. 2

DATE OF SUBMISSION: 18/10/2021

EXPERIMENT NO. 2

AIM : Write a Program to Implement Queue using Arrays

THEORY : **QUEUE**

A queue is a FIFO (First-In, First-Out) data structure in which the element that is inserted first is the first one to be taken out. The elements in a queue are added at one end called the REAR and removed from the other end called the FRONT.

Operations on Queue

1. Insertion of an Element

Insertion operation is used to add an element in the Queue. Algorithm for inserting an element is as follows,

ALGORITHM : INSERTION

```
Step 1: IF REAR = MAX-1
            Write "OVERFLOW"
            Goto Step 4
        [END OF IF]
Step 2: IF FRONT = -1 and REAR = -1
            SET FRONT = REAR = 0
        ELSE
            SET REAR = REAR + 1
        [END OF IF]
Step 3: SET QUEUE[REAR] = NUM
Step 4: EXIT
```

In Step 1, we first check for the overflow condition. In Step 2, we check if the

queue is empty. In case the queue is empty, then both FRONT and REAR are set tozero, so that the new value can be stored at the 0 th location. Otherwise, if the queue already has some values, then REAR is incremented so that it points to the next location in the array. In Step 3, the value is stored in the queue at the location pointed by REAR .

2. Deletion of an Element

Deletion operation is used to remove an element from the Queue.

Algorithm for deleting an element is as follows,

ALGORITHM : DELETION

```
Step 1: IF FRONT = -1 OR FRONT > REAR
            Write "UNDERFLOW"
        ELSE
            SET VAL = QUEUE[FRONT]
            SET FRONT = FRONT + 1
        [END OF IF]
Step 2 : EXIT
```

In Step 1, we check for underflow condition. An underflow occurs if FRONT = -1 or FRONT > REAR. However, if queue has some values, then FRONT is incremented so that it now points to the next value

Assignment:

1)Write algorithm of stack implementation using array
2)PDF format of programming. INCLUDING OUTPUT
3)Answers all the following answers

1. Which one of the following is an application of Queue Data Structure? **(A)** When a resource is shared among multiple consumers.
(B) When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes
(C) Load Balancing
(D) All of the above

2. How many stacks are needed to implement a queue. Consider the situation where no other data structure like arrays, linked list is available to you.

(A) 1
(B) 2
(C) 3
(D) 4

3. How many queues are needed to implement a stack. Consider the situation where no other data structure like arrays, linked list is available to you.
1. 1
2. 2
3.3
4.4

4. Suppose a circular queue of capacity (n – 1) elements is implemented with an array of n elements. Assume that the insertion and deletion operation are carried out using REAR and FRONT as array index variables, respectively. Initially, REAR = FRONT = 0. The conditions to detect queue full and queue empty are
(A) Full: (REAR+1) mod n == FRONT, empty: REAR == FRONT
(B) Full: (REAR+1) mod n == FRONT, empty: (FRONT+1) mod n == REAR
(C) Full: REAR == FRONT, empty: (REAR+1) mod n == FRONT
(D) Full: (FRONT+1) mod n == REAR, empty: REAR == FRONT

5. Suppose implementation supports an instruction REVERSE, which reverses the order of elements on the stack, in addition to the PUSH and POP instructions. Which one of the following statements is TRUE with respect to this modified stack?

(A) A queue cannot be implemented using this stack.
(B) A queue can be implemented where ENQUEUE takes a single instruction and DEQUEUE takes a sequence of two instructions.
(C) A queue can be implemented where ENQUEUE takes a sequence of three instructions and DEQUEUE takes a single instruction.
(D) A queue can be implemented where both ENQUEUE and DEQUEUE take a single instruction each.

Write algorithm of implementation of queue using array.

ENQUEUE:-

STEP 1:- START

STEP 2:- Check whether queue is FULL.

(i.e rear = = SIZE-1)

STEP 3:- If it is FULL, then print "QUEUE is FULL" and terminate the function.

STEP 4:- If it is NOT FULL, then increment rear value by one(rear++) and set queue[rear] = value.

STEP 5:- STOP

C statements as follows:- void

```
enqueue(int value) {      if(rear
= = SIZE-1)
printf("\n QUEUE IS FULL! Insertion not possible.");
else{ if(front = =-1) front = 0; rear ++;
queue[rear]=value;
printf("\n Insertion success !");
}
}
```

DEQUEUE:-

STEP 1:- START

STEP 2:- Check whether queue is EMPTY.

STEP 3:- If it is EMPTY, then print "QUEUE is EMPTY" and terminate the function.

STEP 4:- If it is NOT EMPTY, then increment the front value by one (front ++). Then display queue (front) as deleted element. Then check whether both front and rear are equal (front = = rear), if TRUE, then set both front and rear to '-1' (front = rear =-1).

STEP 5:- STOP

C statements as follows:-

```
void dequeue(){ if(front =
=  rear)
printf("\n QUEUE is EMPTY! Deletion is not possible"); else{
printf("Deleted:%d", queue[front]);
front++; if(front = = rear) front =
rear = -1;
}
}
```

DISPLAY:

STEP 1:- START

STEP 2:- Check whether queue is EMPTY.

i.e (front = = rear)

STEP 3:- If it is EMPTY, then display "QUEUE is EMPTY" and terminate the function.

STEP 4:- If it is NOT EMPTY, then define an integer variable: and set ' i = format +1'

STEP 5 :- Display 'queue[i] value and increment 'I' value by one

(i++). Repeat the same until(i) value reaches to rear (i<= rear)

C statements are as follows:- void display( ){ if (rear = =-1) printf("\n Queue is Empty"); else{ printf("Queue elements are : \n'); for(I = front; i<=rear; i++) printf("%d", queue[i]);

}

}

Queue implementation using array C program.

PROGRAM:-

DS3.c

```c
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define SIZE 10
void enQueue(int);
void deQueue();
void display();
int queue[SIZE], front = -1, rear = -1;
void main()
{
 int value, choice;

 while(1){
 printf("\n\n***** MENU *****\n");
 printf("1. Insertion\n2. Deletion\n3. Display\n4. Exit");
 printf("\nEnter your choice: ");
 scanf("%d",&choice);
 switch(choice){
 case 1: printf("Enter the value to be insert: ");
 scanf("%d",&value);
 enQueue(value);
 break;
 case 2: deQueue();
 break;
 case 3: display();
 break;
 case 4: exit(0);
 default: printf("\nWrong selection!!! Try again!!!");
```

DS3.c

```c
    }
    }
}
void enQueue(int value)
{
    if(rear == SIZE-1)
    printf("\nQueue is Full!!! Insertion is not possible!!!");
    else
    {
    if(front == -1)
    front = 0;
    rear++;
    queue[rear] = value;
    printf("\nInsertion success!!!");
    }
}
void deQueue()
{
    if(front == rear)
    printf("\nQueue is Empty!!! Deletion is not possible!!!");
    else
    {
    printf("\nDeleted : %d", queue[front]);
    front++;
    if(front == rear)
    front = rear = -1;
    }
}
```

DS3.c

```c
}
void display()
{
    if(rear == -1)
    printf("\nQueue is Empty!!!");
    else
    {
    int i;
    printf("\nQueue elements are:\n");
    for(i=front; i<=rear; i++)
    printf("%d\t",queue[i]);
    }
}
```

OUTPUT:-

```
***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 30

Insertion success!!!

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 31

Insertion success!!!

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 40

Insertion success!!!

***** MENU *****
1. Insertion
```

```
Insertion success!!!

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 3

Queue elements are:
30      31      40

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 2

Deleted : 30

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 2

Deleted : 31

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 2

Queue is Empty!!! Deletion is not possible!!!
```

NAME: SINGH SUDHAM DHARMENDRA

ROLL NO.: AIMLD50

BRANCH: CSE (AIML)

SUBJECT: DATA STRUCUTRE

TOPIC: EXPERIMENT NO. 3

DATE OF SUBMISSION: 25/10/2021

EXPERIMENT NO. 3

AIM : Write a Program to Convert Infix Expression to Postfix Form

THEORY : Let '*I*' be an algebraic expression written in infix notation. I may contain parentheses, operands, and operators. For simplicity of the algorithm we will use only + , – , * , / , % operators. The precedence of these operators can be given as follows:

Higher priority * , / , %

Lower priority + , –

No doubt, the order of evaluation of these operators can be changed by making use of parentheses. For example, if we have an expression A + B * C , then first B * C will be done and the result will be added to A . But the same expression if written as, (A + B) * C , will evaluate A + B first and then the result will be multiplied with C.

The algorithm given below transforms an infix expression into postfix expression. The algorithm accepts an infix expression that may contain operators, operands, and parentheses. For simplicity, we assume that the infix operation contains only modulus (%), multiplication (*), division (/), addition (+), and subtraction (—) operators and that operators with same precedence are performed from left-to-right.

The algorithm uses a stack to temporarily hold operators. The postfix expression is obtained from left-to-right using the operands from the infix expression and the operators which are removed from the stack. The first step in this algorithm is to push a left parenthesis on the stack and to add a corresponding right parenthesis at the end of the infix expression. The algorithm is repeated until the stack is empty.

ALGORITHM : INFIX TO POSTFIX CONVERSION

```
Step 1: Add ")" to the end of the Infix Expression
Step 2: Push "(" on to the stack
Step 3: Repeat until each character in the infix notation is scanned
        IF a "(" is encountered, push it on the stack
        IF an Operand (whether a digit or a character) is encountered, add it to the
        postfix expression.
        IF a ")" is encountered, then
           a. Repeatedly pop from stack and add it to the postfix expression until a "("
```

```
            is encountered.
         b. Discard the "(". That is, remove the "(" from stack and do not add it to
            the postfix expression
      IF an Operator is encountered, then
         a. Repeatedly pop from stack and add each operator (popped from the stack) to
            the postfix expression which has the same precedence or a higher precedence
            than 0
         b. Push the operator to the stack
      [END OF IF]
Step 4: Repeatedly pop from the stack and add it to the postfix expression until the
      stack is empty
Step 5: EXIT
```

PROGRAM:

C infinix-postfix.c > main()

```c
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<string.h>

#define Size 100

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

void push(char item)
{
    if(top >= Size-1)
    {
        printf("\nStack Overflow.");
    }
    else
    {
        top = top+1;
        stack[top] = item;
    }
}

char pop()
{
    char item ;

    if(top <0)
    {
        printf("stack under flow");
        getchar();

        exit(1);
    }
    else
    {
      item = stack[top];
      top = top-1;
```

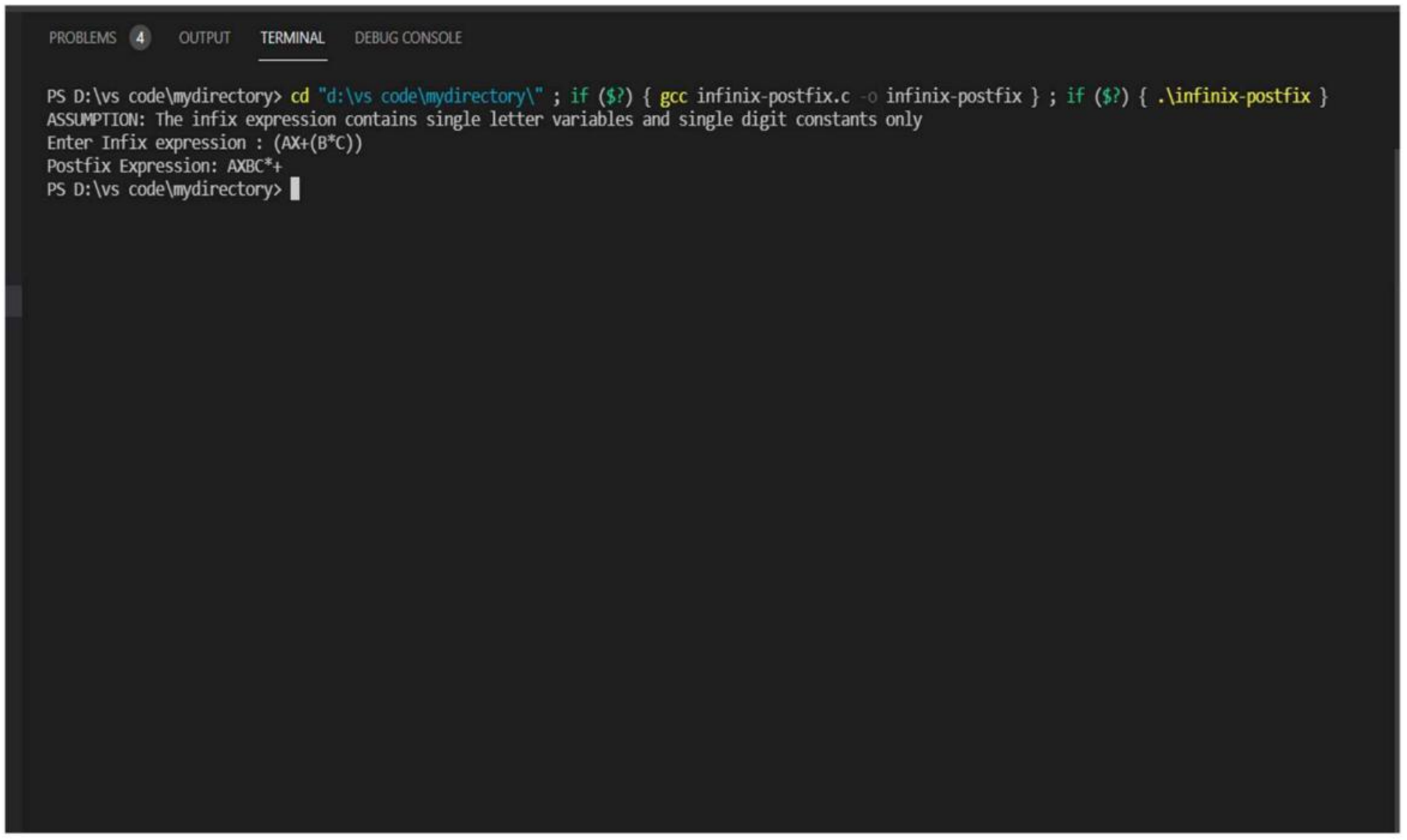
```
    }
    else
    {
      item = stack[top];
      top = top-1;
      return(item);
    }
}

int isOperator(char sym)
{
    if(sym == '^' || sym == '*' || sym == '/' || sym == '+' || sym == '-')
 {
    return 1;
 }
 else
 {
    return 0;
 }
}

int precedence(char sym)
{
    if(sym == '^')
    {
        return(3);
    }
    else if(sym == '*' || sym == '/')
    {
        return(2);
    }
    else if(sym == '+' || sym == '-')
    {
        return(1);
    }
else
{
    return(0);
```

```
    {
        return(1);
    }
else
{
    return(0);
}
}

void InfixToPostfix(char infixExp[], char postfixExp[])
{
  int s = 0;
  int m = 0;
  char item;
  char j;
  push('(');
  strcat(infixExp,")");
  item = infixExp[s];
  while(item != '\0')
  {
      if(item == '(')
      {
          push(item);
      }
      else if( isdigit(item) || isalpha(item))
      {
          postfixExp[m] = item;
          m++;
      }
      else if(isOperator(item) == 1)
      {
          j = pop();
          while(isOperator(j) == 1 && precedence(j)>= precedence(item))
          {
              postfixExp[m] = j;
              m++;
              j = pop();
          }
```

```
    {
        j = pop();
        while(isOperator(j) == 1 && precedence(j)>= precedence(item))
        {
            postfixExp[m] = j;
            m++;
            j = pop();
        }
        push(j);

        push(item);
    }
    else if(item == ')')
    {
        j = pop();
        while(j != '(')
        {
            postfixExp[m] = j;
            m++;
            j = pop();
        }
    }
    else
    {
        printf("\nInvalid infix Expression.\n");
        getchar();
        exit(1);
    }
    s++;

    item = infixExp[s];
}
if(top>0)
{
    printf("\nInvalid infix Expression.\n");
    getchar();
    exit(1);
}
```

```
        item = infixExp[s];
    }
    if(top>0)
    {
        printf("\nInvalid infix Expression.\n");
        getchar();
        exit(1);
    }
    if(top>0)
    {
        printf("\nInvalid infix expression.\n");
        getchar();
        exit(1);
    }

    postfixExp[j] = '\0';

}
int main()
{
    char infix[Size], postfix[Size];

    printf("ASSUMPTION: The infix expression contains single letter variables and single digit constants only");
    printf("\nEnter Infix expression : ");
    gets(infix);

    InfixToPostfix(infix,postfix);
    printf("Postfix Expression: ");
    puts(postfix);

    return 0;
}
```

OUTPUT:



NAME:-SINGH SUDHAM DHARMENDRA
ROLL NO:-AIMLD50
BRANCH:-CSE(AIML)
SUBJECT:-DATA STRUCTURE
EXPERIMENT NO:-4(Postfix Evaluation)
DATE OF SUBMISSION:08/12/2021

➢ CODE:

```c
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#define MAX 100
float st[MAX];
int top = -1;
void push(float st[], float val);
float pop(float st[]);
float evaluatePostfixExp(char exp[]);
int main()
{
    float val;
    char exp[100];
    printf("\n Enter any postfix expression : ");
    gets(exp);
    val = evaluatePostfixExp(exp);
    printf("\n Value of the postfix expression = %.2f", val);
    return 0;
}
float evaluatePostfixExp(char exp[])
{
    int i = 0;
    float op1, op2, value;
    while (exp[i] != '\0')
    {
        if (isdigit(exp[i]))
            push(st, (float)(exp[i] - '0'));
        else
        {
            op2 = pop(st);
            op1 = pop(st);
            switch (exp[i])
            {
            case '+':
                value = op1 + op2;
                break;
            case '-':
                value = op1 - op2;
                break;
```

```
            case '/':
                value = op1 / op2;
                break;
            case '*':
                value = op1 * op2;
                break;
            case '%':
                value = (int)op1 % (int)op2;
                break;
            }
            push(st, value);
        }
        i++;
    }
    return (pop(st));
}
void push(float st[], float val)
{
    if (top == MAX - 1)
        printf("\n STACK OVERFLOW");
    else
    {
        top++;
        st[top] = val;
    }
}
float pop(float st[])
{
    float val = -1;
    if (top == -1)
        printf("\n STACK UNDERFLOW");
    else
    {
        val = st[top];
        top--;
    }
    return val;
}
```

➢ OUTPUT:

```
 Enter any postfix expression : 24*4+

 Value of the postfix expression = 12.00
```

NAME:-SINGH SUDHAM DHARMENDRA
ROLL NO:-AIMLD50
BRANCH:-CSE(AIML)
SUBJECT:-DATA STRUCTURE
EXPERIMENT NO:-5
DATE OF SUBMISSION:08/12/2021

CODE:

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

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

struct Node *head = NULL;
void printList(struct Node *head)
{

    while (head)
    {
        printf("%d ", head->data);
        head = head->next;
    }
    printf("\n");
}

struct Node *pushFront(struct Node *head, int x)
{
    struct Node *new_node = (struct Node *)malloc(sizeof(struct Node));
    new_node->data = x;
    new_node->next = head;
    printf("\nNew Node Inserted at the beginning with value %d", x);
    return new_node;
}

struct Node *popFront(struct Node *head)
{
    if (head == NULL)
    {
        printf("\nNode is empty");
        return head;
    }
```

```
    struct Node *temp = head->next;
    free(head);
    printf("\nNode deleted from the beginning");

    return temp;
}

struct Node *pushKth(struct Node *head, int x, int n)
{
    struct Node *new_Node = (struct Node *)malloc(sizeof(struct Node));
    new_Node->data = x;
    if (n == 1)
    {
        new_Node->next = head;
        head = new_Node;
        return head;
    }

    struct Node *curr = head;
    for (int i = 0; (i < n - 2) && curr; i++)
    {
        curr = curr->next;
    }
    if (curr == NULL)
    {
        printf("\nThe given position beyond the limits\n");
        return head;
    }

    new_Node->next = curr->next;
    curr->next = new_Node;
    printf("\nNew Node inserted at given postion");
    return head;
}

struct Node *popKth(struct Node *head, int n)
{
    struct Node *curr = head;
    if (head == NULL)
    {
```

```
        return head;
    }
    if (n == 1)
    {
        head = curr->next;
        free(curr);
        return head;
    }
    for (int i = 0; i < n - 2; i++)
    {
        curr = curr->next;
    }
    if ((curr && curr->next) == 0)
    {
        printf("\nThe given position beyond the limits\n");
        return head;
    }
    struct Node *temp = curr->next;
    curr->next = temp->next;
    free(temp);
    printf("\nNode removed from the given postion");
    return head;
}
struct Node* pushTail(struct Node *head, int x)
{
    struct Node *temp = (struct Node *)malloc(sizeof(struct Node)); //
Creting a new Node
    temp->data = x;
    if (head == NULL)        // if head is NULL we will change the head
with temp
    {
        return temp;
    }
    else
    {
        struct Node *curr = head; // curr to traverse at the end of Linked
List
        while (curr->next)  // until curr is NULL, loop will run
        {
            curr = curr->next; // Update curr next to curr
```

```
        }
        curr->next = temp; // curr is now the end of Linked List so we
will update it with temp
        return head;
    }
}
struct Node * popTail(struct Node *head)
{
    if (head == NULL)
    {
        return NULL;
    }
    if (head->next == NULL)
    {
        free (head);
        head = NULL;
        return NULL;
    }

    struct Node *curr = head;
    while (curr->next->next)
    {
        curr = curr->next;
    }
    free (curr->next);
    curr->next = NULL;
    return head;
}

int main()
{
    int option, x, k;
    printf("\nLINKED LIST CREATED");

    do
    {
        printf("\n\n**LINKED LIST**\n");
        printf("\nSelect any one the following option : ");
        printf("\n1] Display a Linked List");
        printf("\n2] Insert a new node at the beggining");
```

```
        printf("\n3] Insert a new node at a given a postion");
        printf("\n4] Insert a new node at the end");
        printf("\n5] Delete a node at the beginning");
        printf("\n6] Delete a node at a given position");
        printf("\n7] Delete a node at the end");
        printf("\n8] EXIT\n");
        scanf("%d", &option);

        switch (option)
        {
        case 1:
            printf("\nLinked List: ");
            printList(head);
            break;

        case 2:
            printf("Enter the value: ");
            scanf("%d", &x);
            head = pushFront(head, x);
            break;
        case 3:
            printf("\nEnter the value: ");
            scanf("%d", &x);
            printf("\nEnter the position: ");
            scanf("%d", &k);
            head = pushKth(head, x, k);
            break;

        case 4:
            printf("Enter the value: ");
            scanf("%d", &x);
            head = pushTail(head, x);
            break;

        case 5:
            head = popFront(head);
            break;

        case 6:
            printf("\nEnter the position: ");
```

```
            scanf("%d", &k);
            head = popKth(head,k);
            break;
        case 7:
            head = popTail(head);
            break;

        case 8:
            break;
        default:
            printf("\nInvalid choice.");
        }
    } while (option != 8);

    return 0;
}
```

OUTPUT:

```
LINKED LIST CREATED

**LINKED LIST**

Select any one the following option :
1] Display a Linked List
2] Insert a new node at the beggining
3] Insert a new node at a given a postion
4] Insert a new node at the end
5] Delete a node at the beginning
6] Delete a node at a given position
7] Delete a node at the end
8] EXIT
1

Linked List:


**LINKED LIST**

Select any one the following option :
1] Display a Linked List
2] Insert a new node at the beggining
3] Insert a new node at a given a postion
4] Insert a new node at the end
5] Delete a node at the beginning
6] Delete a node at a given position
7] Delete a node at the end
8] EXIT
2
Enter the value: 2

New Node Inserted at the beginning with value 2

**LINKED LIST**

Select any one the following option :
1] Display a Linked List
2] Insert a new node at the beggining
3] Insert a new node at a given a postion
4] Insert a new node at the end
5] Delete a node at the beginning
6] Delete a node at a given position
7] Delete a node at the end
8] EXIT
2
Enter the value: 3

New Node Inserted at the beginning with value 3

**LINKED LIST**
```

```
Select any one the following option :
1] Display a Linked List
2] Insert a new node at the beggining
3] Insert a new node at a given a postion
4] Insert a new node at the end
5] Delete a node at the beginning
6] Delete a node at a given position
7] Delete a node at the end
8] EXIT
2
Enter the value: 4

New Node Inserted at the beginning with value 4

**LINKED LIST**

Select any one the following option :
1] Display a Linked List
2] Insert a new node at the beggining
3] Insert a new node at a given a postion
4] Insert a new node at the end
5] Delete a node at the beginning
6] Delete a node at a given position
7] Delete a node at the end
8] EXIT
4
Enter the value: 22


**LINKED LIST**

Select any one the following option :
1] Display a Linked List
2] Insert a new node at the beggining
3] Insert a new node at a given a postion
4] Insert a new node at the end
5] Delete a node at the beginning
6] Delete a node at a given position
7] Delete a node at the end
8] EXIT
1

Linked List: 4 3 2 22


**LINKED LIST**

Select any one the following option :
1] Display a Linked List
2] Insert a new node at the beggining
3] Insert a new node at a given a postion
4] Insert a new node at the end
5] Delete a node at the beginning
```

```
6] Delete a node at a given position
7] Delete a node at the end
8] EXIT
5

Node deleted from the beginning

**LINKED LIST**

Select any one the following option :
1] Display a Linked List
2] Insert a new node at the beggining
3] Insert a new node at a given a postion
4] Insert a new node at the end
5] Delete a node at the beginning
6] Delete a node at a given position
7] Delete a node at the end
8] EXIT
1

Linked List: 3 2 22


**LINKED LIST**

Select any one the following option :
1] Display a Linked List
2] Insert a new node at the beggining
3] Insert a new node at a given a postion
4] Insert a new node at the end
5] Delete a node at the beginning
6] Delete a node at a given position
7] Delete a node at the end
8] EXIT
6

Enter the position: 2

Node removed from the given postion

**LINKED LIST**

Select any one the following option :
1] Display a Linked List
2] Insert a new node at the beggining
3] Insert a new node at a given a postion
4] Insert a new node at the end
5] Delete a node at the beginning
6] Delete a node at a given position
7] Delete a node at the end
8] EXIT
1
```

```
Linked List: 3 22


**LINKED LIST**

Select any one the following option :
1] Display a Linked List
2] Insert a new node at the beggining
3] Insert a new node at a given a postion
4] Insert a new node at the end
5] Delete a node at the beginning
6] Delete a node at a given position
7] Delete a node at the end
8] EXIT
7


**LINKED LIST**

Select any one the following option :
1] Display a Linked List
2] Insert a new node at the beggining
3] Insert a new node at a given a postion
4] Insert a new node at the end
5] Delete a node at the beginning
6] Delete a node at a given position
7] Delete a node at the end
8] EXIT
1

Linked List: 3


**LINKED LIST**

Select any one the following option :
1] Display a Linked List
2] Insert a new node at the beggining
3] Insert a new node at a given a postion
4] Insert a new node at the end
5] Delete a node at the beginning
6] Delete a node at a given position
7] Delete a node at the end
8] EXIT
8
PS E:\Programming\College\Data Structures>
```

NAME: SINGH SUDHAM DHARMENDRA

ROLL NO.: AIMLD50

BRANCH: CSE (AI & ML)

SUBJECT: DATA STRUCTURE

TOPIC: EXPERIMENT NO. 6

DATE OF SUBMISSION: 9/12/2021

CODE:

```c
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int data;
    struct node* next;
};
struct node* head= NULL;
void push(int x){
    struct node* temp = (struct node*)malloc(sizeof (struct node));
    temp -> data = x;
    temp -> next= head;
    head = temp;
}
void pop(){
    int x;
    struct node* temp;
    if (head == NULL){
        printf("underflow");
    }
    else{
        temp= head;
        head = temp -> next;
        free(temp);
        printf("item is popped");
    }
}
void printlist(){
    struct node*temp1 = head;
    while (temp1 != NULL){
        printf("\n%d\n",temp1 -> data);
        temp1= temp1 -> next;
    }
}

int main(){
    int choice=0;
```

```c
}

int main(){
    int choice=0;
    while(choice != 4)
    {
        printf("\n\nChose one from the below options...\n");
        printf("\n1.Push\n2.Pop\n3.Show\n4.Exit");
        printf("\n Enter your choice \n");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
            {
                int val;
                printf("Enter the value\n");
                scanf("%d",&val);
                push(val);
                break;
            }
            case 2:
            {
                pop();
                break;
            }
            case 3:
            {
                printlist();
                break;
            }
            case 4:
            {
                printf("Exiting....");
                break;
            }
            default:
            {
```

expt6.c > printlist()

```c
                push(val);
                break;
            }
            case 2:
            {
                pop();
                break;
            }
            case 3:
            {
                printlist();
                break;
            }
            case 4:
            {
                printf("Exiting....");
                break;
            }
            default:
            {
                printf("Please Enter valid choice ");
            }
        }
    }
}
```

Ln 32, Col 30 Spaces: 4 UTF-8 CRLF C Win32

OUTPUT:

minal Help expt6.c - mydirectory - Visual Studio Code

PROBLEMS 66 OUTPUT TERMINAL DEBUG CONSOLE Code

```
Chose one from the below options...

1.Push
2.Pop
3.Show
4.Exit
 Enter your choice
1
Enter the value
34


Chose one from the below options...

1.Push
2.Pop
3.Show
4.Exit
 Enter your choice
1
Enter the value
55


Chose one from the below options...

1.Push
2.Pop
3.Show
4.Exit
 Enter your choice
3

55

34


Chose one from the below options...

1.Push
2.Pop
3.Show
4.Exit
 Enter your choice
```

Ln 12, Col 24 Spaces: 4 UTF-8 CRLF C Win32

NAME:-SINGH SUDHAM DHARMENDRA
ROLL NO:-AIMLD50
BRANCH:-CSE(AIML)
SUBJECT:-DATA STRUCTURE
EXPERIMENT NO:-7
DATE OF SUBMISSION:08/12/2021

CODE:

```c
#include <stdio.h>
#define MAX 6
int cqueue[MAX];
int front = -1;
int rear = -1;
void insert(int item);
void del();
void display();
int main()
{
    int choice, item;
    do
    {
        printf("1.Insert\n");
        printf("2.Delete\n");
        printf("3.Display\n");
        printf("4.Exit\n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("Input the element for insertion in queue : ");
            scanf("%d", &item);
            insert(item);
            break;
        case 2:
            del();
            break;
        case 3:
            display();
            break;
        case 4:
            break;
        default:
            printf("Wrong choice\n");
        }
    } while (choice != 4);
```

```
    return 0;
}
void insert(int item)
{
    if ((front == 0 && rear == MAX - 1) || (front == rear + 1))
    {
        printf("Queue Overflow \n");
        return;
    }
    if (front == -1)
    {
        front = 0;
        rear = 0;
    }
    else
    {
        if (rear == MAX - 1)
            rear = 0;
        else
            rear = rear + 1;
    }
    cqueue[rear] = item;
}
void del()
{
    if (front == -1)
    {
        printf("Queue Underflow\n");
        return;
    }
    printf("Element deleted from queue is : %d\n", cqueue[front]);
    if (front == rear)
    {
        front = -1;
        rear = -1;
    }
    else
    {
        if (front == MAX - 1)
            front = 0;
```

```
        else
            front = front + 1;
    }
}
void display()
{
    int front_pos = front, rear_pos = rear;
    if (front == -1)
    {
        printf("Queue is empty\n");
        return;
    }
printf("Queue elements :\n");
if (front_pos <= rear_pos)
    while (front_pos <= rear_pos)
    {
        printf("%d ", cqueue[front_pos]);
        front_pos++;
    }
else
{
    while (front_pos <= MAX - 1)
    {
        printf("%d ", cqueue[front_pos]);
        front_pos++;
    }
    front_pos = 0;
    while (front_pos <= rear_pos)
    {
        printf("%d ", cqueue[front_pos]);
        front_pos++;
    }
}
printf("\n");
}
```

OUTPUT:

```
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice : 1
Input the element for insertion in queue : 12
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice : 1
Input the element for insertion in queue : 23
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice : 1
Input the element for insertion in queue : 34
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice : 3
Queue elements :
12 23 34
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice : 2
Element deleted from queue is : 12
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice : 3
Queue elements :
23 34
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice : 4
```

NAME:-SINGH SUDHAM DHARMENDRA
ROLL NO:-AIMLD50
BRANCH:-CSE(AIML)
SUBJECT:-DATA STRUCTURE
TOPIC:-EXPERIMENT NO-8
DATE OF SUBMISSION:8/12/21

- **CODE:**

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

struct Node
{
    int data;
    struct Node *next;
};
struct Node *head = NULL;
struct Node *tail = NULL;

struct Node *new_node(int x)
{
    struct Node *node = (struct Node *)malloc(sizeof(struct Node));
    node->data = x;
    node->next = NULL;
    return node;
}

void printQueue()
{
    struct Node *curr = head;
    printf("Queue: ");
    while (curr != NULL)
    {
        printf("%d ", curr->data);
        curr = curr->next;
    }
    printf("\n");
}

void enqueue(int x)
{
    struct Node *newNode = new_node(x);
    if (head == NULL)
    {
        head = newNode;
        tail = newNode;
```

```
        return;
    }
    else
    {
        tail->next = newNode;
        tail = newNode;
        return;
    }
}

void deque()
{
    if (head == NULL)
    {
        printf("UnderFlow\n");
        return;
    }
    else
    {
        struct Node *temp = head;
        head = head->next;
        free(temp);
        return;
    }
}

int main()
{
    deque();
    enqueue(2);
    enqueue(3);
    printQueue();
    deque();
    printQueue();
    enqueue(5);
    printQueue();

    return 0;
}
```

- **OUTPUT:**

NAME:-SINGH SUDHAM DHARMENDRA
ROLL NO:-AIMLD50
BRANCH:-CSE(AIML)
SUBJECT:-DATA STRUCTURE
TOPIC:-EXPERIMENT NO-9
DATE OF SUBMISSION:8/12/21

- **CODE:**

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *next;
};

struct node *last = NULL;

void insert()
{
    int x;
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    printf("\nEnter data to be inserted: \n");
    scanf("%d", &x);

    if (last == NULL)
    {
        temp->data = x;
        temp->next = temp;
        last = temp;
    }

    else
    {
        temp->data = x;
        temp->next = last->next;
        last->next = temp;
    }
}

void printList()
{
    if (last == NULL)
        printf("\nList is empty\n");

    else
```

```
    {
        struct node *temp = last->next;
        do
        {
            printf("%d-->", temp->data);
            temp = temp->next;
        } while (temp != last->next);
    }
}

int main()
{
    int x, n, i;
    printf("how many nodes??\n");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        insert();
    }
    printList();
    return 0;
}
```

- **OUTPUT:**

```
how many nodes??
3

Enter data to be inserted:
2

Enter data to be inserted:
3

Enter data to be inserted:
4
4-->3-->2-->
```

NAME:-SINGH SUDHAM DHARMENDRA
ROLL NO:-AIMLD50
BRANCH:-CSE(AIML)
SUBJECT:-DATA STRUCTURE
TOPIC:-EXPERIMENT NO-10
DATE OF SUBMISSION:8/12/21

- **CODE:**

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    struct node *lchild;
    int info;
    struct node *rchild;
};
struct node *insert(struct node *ptr, int ikey);
void display(struct node *ptr, int level, int  side);

int main()
{
    struct node *root = NULL, *ptr;
    int choice, k, p, i;
    while (1)
    {
        printf("\n");
        printf("1.Insert\n");
        printf("2.Display\n");
        printf("3.Quit\n");
        printf("\nEnter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("\nEnter the key to be inserted : ");
            scanf("%d", &k);
            root = insert(root, k);
            break;
        case 2:
            printf("\n");
            display(root, 0, 1);
```

```
            printf("\n");
            break;
        case 3:
            exit(1);
        default:
            printf("\nWrong choice\n");
        }
    }
    return 0;
}

struct node *insert(struct node *ptr, int ikey)
{
    if (ptr == NULL)
    {
        ptr = (struct node *)malloc(sizeof(struct node));
        ptr->info = ikey;
        ptr->lchild = NULL;
        ptr->rchild = NULL;
    }
    else if (ikey < ptr->info)
        ptr->lchild = insert(ptr->lchild, ikey);
    else if (ikey > ptr->info)
        ptr->rchild = insert(ptr->rchild, ikey);
    else
        printf("\nDuplicate key\n");
    return ptr;
}

void display(struct node *ptr, int level, int side)
{
    int i, p = 0;
    if (ptr == NULL)
        return;
```

```
    level++;
    display(ptr->rchild, level, 1);
    printf("\n");
    for (i = 0; i < level; i++)
    {
        printf("\t");
    }
    if (side == 1)
        printf("/ ");
    else
        printf("\\ ");
    printf("%d", ptr->info);
    display(ptr->lchild, level, 0);
}
```

- **OUTPUT:**

```
1.Insert
2.Display
3.Quit

Enter your choice : 1

Enter the key to be inserted : 5

1.Insert
2.Display
3.Quit

Enter your choice : 1

Enter the key to be inserted : 3

1.Insert
2.Display
3.Quit

Enter your choice : 1

Enter the key to be inserted : 7

1.Insert
2.Display
3.Quit

Enter your choice : 1

Enter the key to be inserted : 1

1.Insert
2.Display
3.Quit

Enter your choice : 1

Enter the key to be inserted : 2

1.Insert
2.Display
3.Quit

Enter your choice : 1

Enter the key to be inserted : 6

1.Insert
2.Display
3.Quit
```

```
Enter your choice : 1

Enter the key to be inserted : 8

1.Insert
2.Display
3.Quit

Enter your choice : 2


                    / 8
              / 7
                    \ 6
        / 5
              \ 3
                          / 2
                    \ 1

1.Insert
2.Display
3.Quit

Enter your choice : 3
PS F:\Programming\College\Data Structures>
```

NAME: SINGH SUDHAM DHARMENDRA

ROLL NO.: ALMLD50

BRANCH: CSE (AI & ML)

SUBJECT: DATA STRUCTURE

TOPIC: ASSIGNMENT NO. 1

DATE OF SUBMISSION: 10/10/2021

LOKMANYA TILAK COLLEGE OF ENGINEERING

Sector No. 4., Vikas Nagar, Koparkhairane, Navi Mumbai -400 709.

DEPARTMENT OF COMPUTER ENGINEERING
(Artificial intelligent and Machine learning)

Class/Sem/Year: S.E. / III / 2021-22 (Odd)

Subject: Data Structures

Assignment No. 1

Last Date or submission: 10/10/21

Q. No.	Question	CO Covered
1	Explain different types of data structures with examples	CSC303.1
2	What is Abstract Data Type? Write ADT of stack and state applications of stack.	
3	Explain infix, postfix and prefix expressions with example	
4	Evaluate the following expression using stack ABC*DEF^/G*-H*+ where A = 6, B=1, C= 4, D = 16, E=2, F= 3, G = 2, H = 5	
5	Write a note on Priority Queue.	CSC303.2 & CSC303.3
6	Explain Circular queue and Double ended queue with example.	
7	Write a C program to implement singly linked list. Provide the following operations: i) Insert at beginning ii) Insert at location iii) Delete from beginning iv) Delete from location	
8	Write a C program to implement Queue using Single linked list.	
9	Give applications of linked list. How it can be used to perform additive operation on polynomial?	

Q.1) Explain different types of data structures with examples.

→ ① Every data structures stores the data in a special way to access and used data efficiently. Anything that can be store the data called as a Data structure.

② Based on the characteristics, Data structure are classified into Primitive & Non-primitive data structure.

③ Primitive Data Structure :-

(i) Primitive data structure are the fundamental data types which are supported by a programming language.

(ii) The term 'data type', 'basic data type' and 'primitive data type' are often interchangably.

(iii) Some basic data type are integer, Float, double, characters, pointer, bytes, etc.

④ Non-primitive Data structure :-

(i) Data structure are defined that by user, and are created using primitive data structure they are called as non-primitive data structure.

(ii) It is further classified into Linear & Non-linear data structure

- Linear Data Structure :-

In linear data structure, the elements are stored in a sequential manner. The relationship of adjacency is maintained between elements.

for eg :- Stack, Array, Queue, linked list

- Non-Linear Data structure :-

Non-linear data structure, the elements are not stored in a linear or sequential manner. The element are not adjacent to each other.

for eg :- Tree, graphs.

Q.2) What is Abstract Data Type? Write ADT of stack and state applications of stack.

→ **Abstract Data Type (ADT):-**

① Data type is set of values that the variable can take
eg:- Int, char, float, double.

② Abstract means considered apart from the detailed specifications or implementation.

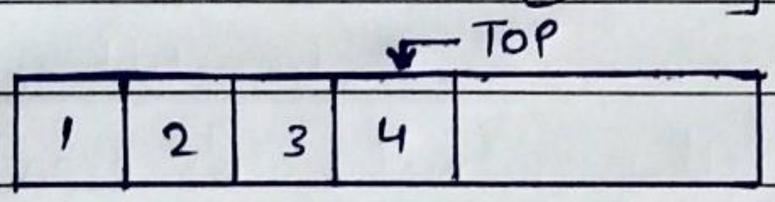
③ An abstract data type (ADT) is a set of operation irrespective of how the set of operation is implemented.
eg:- Stacks, Queues.

ADT of Stack:-

① A stack is a linear data structure with the restriction that inserts and deletes can be performed at only one end.

② This end is called as Top of the stack.

③ Since, the last element added to the stack would be the first element to be deleted Stack is also called as Last In First Out [LIFO] structure.



General implementation of stack.

The fundamental operation on a stack are:-

(i) PUSH - insert an element.

(ii) POP - delete the most recently inserted element.

(iii) PEEK - display an element at the top.

Application of Stack:-

(i) Reversal of a string.

(ii) Checking validity of an expression containing nested paranthesses

(iii) Function calls.

(iv) Conversation & evaluation of infix, postfix & prefix expressions.

Q. 3) Explain infix, postfix & prefix expression with example.

→ ① Infix notation - A+B

When we write any arithmetic expression in infix notation, operators are written in-between their operands.

For eg:- ① (A+B)

② A * (B+C) / D

② Postfix notation (Reverse Polish Notation) - AB+

When we write any arithmetic expression in postfix notation, operators are written after the operands.

For eg:- A B C + *

③ Prefix ~~expressio~~ notation (Polish Notation) - + AB

When we write any arithmetic expression in prefix notation, operators are written before their operands

For eg:- / * A+BCD

The data structure called stack can be used to convert these complex arithmetic expression into the polish strings and then can be executed into two operands and an operator form.

By comparing examples:-

Infix expression	Prefix expression	Postfix expression
A+B	+ AB	AB+
(A+B) * C	* + ABC	AB+C*
(A+B) * (C+D)	* + AB + CD	AB+CD+*

Q.4) Evaluate the following expression using stack

ABC * DEF ^ / G * - H * +

where A=6, B=1, C=4, D=16, E=2, F=3, G=2, H=5

→

Character	Stack
A	A
B	AB
C	ABC
*	AB * C
D	AB * CD
E	AB * CDE
F	AB * CDEF
^	AB * CDE ^ F
/	AB * CD / E ^ F
G	AB * CD / E ^ FG
*	AB * CD / E ^ F * G
-	AB * C - D / E ^ F * G
H	AB * C - D / E ^ F * GH
*	AB * C - D / E ^ F * G * H
+	A + B * C - D / E ^ F * G * H

The infix expression is;

A + (B * C - (D / E ^ F) * G) * H

By putting values in above expression we get,

6 + (1 * 4 - (16 / 2 ^ 3) * 2) * 5

∴ 6 + (1 * 4 - (16 / 8) * 2) * 5

∴ 6 + (1 * 4 - 2 * 2) * 5

∴ 6 + (0) * 5

∴ 6 + 0

∴ 6

(Q.5) Write a note on Priority Queue.

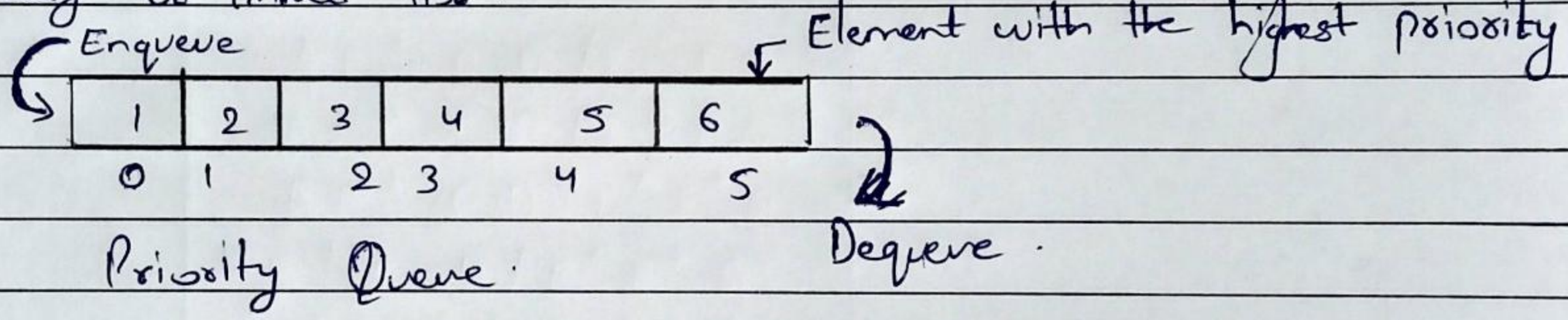
→ A priority queue is a queue in which every element is assigned a priority.

(i) Priority of the elements can be based on several reasons such as importance of one element over the other, CPU time required to execute the process, in case of processes.

(ii) An element with a higher priority will be processed before an element of lower priority.

(iii) When two elements have the same priority, these elements are processed on a first come first served basis.

(iv) Priority queues are best implemented using heap than using array or linked list.



Priority Queue. Dequeue.

Operation possible on Priority queue :-

(i) Enqueue / insert an element at the rear of the queue

(ii) Dequeue / delete and return the element at the front of queue.

Application of Priority Queue :-

(i) Priority queue used for External sorting.

(ii) All the queue applications where priority is involved, these queue are used.

(iii) Priority queue can be used in discrete event simulation

(iv) Used in CPU scheduling.

(v) Priority queue are used in graph algorithm like Prim's minimum spanning tree, etc.

Q.6) Explain Circular queue & double ended queue with example.

→ ① Circular queue is a linear data structure where the first index comes right after the last index assuming indices are attached in a circular manner.

② Implementation of circular queue is similar to the Linear queue with certain difference in enqueue and dequeue operations.

Algorithm to insert an element in circular queue (Enqueue):-

1: If front = 0 & rear = Max-1
   write overflow
   Goto 4
2: If front = -1 & rear = -1
   SET front = Rear = 0
   Else If rear = Max-1 & front != 0
   SET Rear = 0
   ELSE
      SET Rear = Rear + 1
3: SET Queue (Rear) = val
4: EXIT

Algorithm to delete an element in circular queue : (Dequeue):-

1: If front = -1
   Write underflow
   goto 4
2: SET Val = Queue (front)
3: If Front = Rear
   SET Front = Rear = -1
   Else
      If front = Max-1
         SET front = 0
      Else
         SET Front = Front + 1
4: EXIT

Double Ended Queue :-

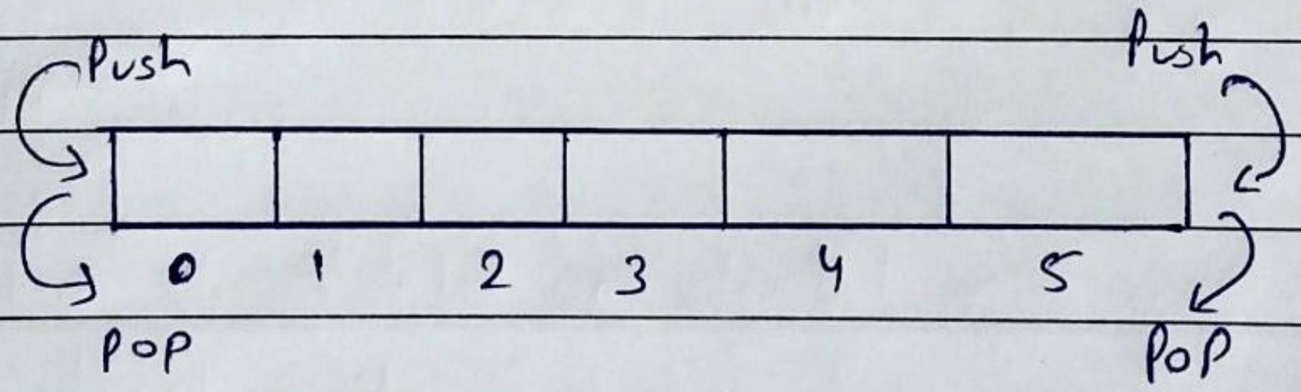
① Dable ended queue is a special type of queue in which elements can be insexted or deleted at both the ends.

② Removing the restrictions given in a linear and circular, in this queue elements can be added to or removed from either the front (head) or the ~~blac~~ back (tail) end.
Hence, it is also called as Head - Tail linked list.
But we cannot add or delete the element from the middle.

③ In dqueue, left & right pointer are maintained.
Dequeue can be implemented using circular array.
There are two types of dequeue.

(i) In-put restricted dequeue :-
In this type of dequeue, deletions can be done from both the ends and insertions can be done from one end only.

(ii) Output restricted dequeue :-
In this type of dequeue, insertion can be done from both the ends and deletions can be done from one end only.



Dequeues.

Q.7) Write a C program to implement singly linked list. Provide the following operations:

I. Insert at beginning
II. Insert at location
III. Delete from beginning
IV. Delete from location

PROGRAM: -

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

void create_list(int);
void addatbeg(int);
void addafter(int,int);
void del(int);
void display();

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

////////////// MAIN FUNCTION ////////////

void main()
{
    int ch,n,m,position,i;
    start = NULL;
    printf("\n 1. Create List\t 2. Add at beginning\t 3. Add after\n");
    printf("\n 4.Delete\t 5.Display\t 6.Quit\n");
     while(1)
     {
        printf("\nEnter your choice:\t");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:
            printf("How many node you want?:\t");
            scanf("%d",&n);
            for(i=0;i<n;i++)
```

```
        {
            printf("Enter the element %d:\t",i+1);
            scanf("%d",&m);
            create_list(m);
        }
        break;

        case 2:
        printf("Enter the element:\t");
        scanf("%d",&m);
        addatbeg(m);
        break;

        case 3:
        printf("Enter the element:\t");
        scanf("%d",&m);
        printf("Enter the position after which element:\t");
        scanf("%d",&position);
        addafter(m,position);
        break;

        case 4:
        if(start == NULL)
        {
            printf("List is empty!!\n");
            continue;
        }
        printf("Enter the element for deletion:\t");
        scanf("%d",&m);
        del(m);
        break;

        case 5:
        display();
        break;

        case 6:
        exit(0);

        default:
        printf("Wromg choice!!\n");
            }
        }
}

////////////////// CREATE LIST FUNCTION ////////////////////

void create_list(int n)
```

```
{
    struct node *q, *tmp;
    tmp= malloc(sizeof(struct node));
    tmp->data=n;
    tmp->next=NULL;

    if(start == NULL)
    start = tmp;
    else
    {
        q=start;
        while(q->next!=NULL)
        q=q->next;
        q->next=tmp;
    }
}

/////////////// ADD AT THE BEGINNING FUNCTION ////////////////

void addatbeg(int n)
{
    struct node *tmp;
    tmp = malloc(sizeof(struct node));
    tmp->data=n;
    tmp->next=start;
    start=tmp;
}

////////////// ADD AFTER FUNCTION ////////////////

void addafter(int n,int pos)
{
    struct node *tmp,*q;
    int i;
    q=start;
    for(i=0;i<pos-1;i++)
    {
        q=q->next;
        if(q==NULL)
        {
            printf("There are less than %d elements",pos);
            return;
        }
    }
    tmp = malloc(sizeof(struct node));
    tmp->next=q->next;
    tmp->data=n;
    q->next=tmp;
```

```c
}

/////////// DELETE FUNCTION /////////////////

void del(int n)
{
    struct node *tmp, *q;
    if(start->data == n)
    {
        tmp=start;
        start=start->next; /*first element delete*/
        free(tmp);
        return;
    }
    q=start;
    while(q->next->next!=NULL)
    {
        if(q->next->data==n) /* Element deleted in between*/
        {
            tmp=q->next;
            q->next=tmp->next;
            free(tmp);
            return;
        }
        q=q->next;
    }
    if(q->next->data==n) /* Last element delete */
    {
        tmp=q->next;
        free(tmp);
        q->next=NULL;
        return;
    }
    printf("Elements %d not found\n",n);
}

/////////// DISPLAY FUNCTION //////////////

void display()
{
    struct node *q;
    if(start == NULL)
    {
        printf("List is empty!!\n");
        return;
    }
    q=start;
    printf("List is:\n");
```

```
    while(q!=NULL)
    {
        printf("%d\t",q->data);
        q=q->next;
    }
    printf("\n");
}

////////////// END PROGRAM //////////////
```

OUTPUT: -

```
 1. Create List  2. Add at beginning     3. Add after

 4.Delete        5.Display       6.Quit

Enter your choice:      1
How many node you want?:        3
Enter the element 1:    10
Enter the element 2:    20
Enter the element 3:    30

Enter your choice:      5
List is:
10      20      30

Enter your choice:      2
Enter the element:      25

Enter your choice:      5
List is:
25      10      20      30

Enter your choice:      3
Enter the element:      75
Enter the position after which element: 2

Enter your choice:      5
List is:
25      10      75      20      30

Enter your choice:      4
Enter the element for deletion: 1
Elements 1 not found

Enter your choice:      4
Enter the element for deletion: 25

Enter your choice:      5
List is:
10      75      20      30

Enter your choice:      4
Enter the element for deletion: 20

Enter your choice:      5
List is:
10      75      30

Enter your choice:      4
Enter the element for deletion: 30

Enter your choice:      5
List is:
10      75
```

Q.8) Write a C program to implement Queue using singly linked list.

PROGRAM:

```c
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define NULL 0

struct queue
{
    int data;
    struct queue*next;
}*p,*front=NULL,*rear=NULL;

////////////// MAIN FUNCTION ////////////////

void main()
{
    int n,i,k,x;
    char ans,ch,Y,y;
    printf("\n 1.Enqueue\t 2.Dequeue\n");
    printf("\n3.Display\t 4.Exit\n");
    do
    {
      printf("\nEnter your choice:\t");
      scanf("%d",&k);
      switch(k)
      {
          ////////////// ENQUEUE ////////////////

          case 1:
          printf("\nEnter an integer:\t");
          scanf("%d",&n);
          p=(struct queue*)malloc(sizeof(struct queue));
          p->data=n;
          p->next=NULL;
          if(rear==NULL)
          front=p;
          else
          {
            rear->next=p;
          }
          rear=p;
          break;
```

```
            ////////////// DEQUEUE ///////////////

            case 2:
            if(rear==NULL)
            printf("\nQueue is empty.");
            else
            {
                p=front;
                printf("\n Element dequeue is :\t %d",front->data);
                front=front->next;
                if(front==NULL)
                rear=NULL;
                free(p);
            }
            break;

            ///////////////// DISPLAY QUEUE /////////////////

            case 3:
            if(rear==NULL)
            printf("\n Queue if Empty.");
            else
            {
                p=front;
                printf("\n Queue contains:\n");
                while(p!=NULL)
                {
                    printf("%d\t",p->data);
                    p=p->next;
                }
            }
            break;

            case 4:
            exit(0);
            break;

            default:
            printf("Wrong choice!!");
            break;
        }
        printf("\n Do you want to continue(y/n)");
        _flushall();
        ans=getch();
    } while (ans=='Y'||'y');
    printf("\n Press any key to continue.");
    getch();
```

```
}

 ////////////// END PROGRAM /////////////////
```

OUTPUT:

```
 1.Enqueue       2.Dequeue

3.Display        4.Exit

Enter your choice:      1

Enter an integer:       25

 Do you want to continue(y/n)
Enter your choice:      1

Enter an integer:       30

 Do you want to continue(y/n)
Enter your choice:      1

Enter an integer:       67

 Do you want to continue(y/n)
Enter your choice:      3

 Queue contains:
25      30      67
 Do you want to continue(y/n)
Enter your choice:      2

 Element dequeue is :    25
 Do you want to continue(y/n)
Enter your choice:      3

 Queue contains:
30      67
 Do you want to continue(y/n)
Enter your choice:      1

Enter an integer:       45

 Do you want to continue(y/n)
Enter your choice:      3

 Queue contains:
30      67      45
 Do you want to continue(y/n)
Enter your choice:      2

 Element dequeue is :    30
 Do you want to continue(y/n)
Enter your choice:      3

 Queue contains:
67      45
 Do you want to continue(y/n)
Enter your choice:      4
PS B:\VScode C programs>
```

Q.9) Give application of Linked list. How it can be used to perform additive operation on polynomial?

→ Application of Linked List:-

1. Implement of Stacks and queues.
2. Implement of graphs.
3. Dynamic memory allocation.
4. Perform arithmetic operation on long integers.
5. Maxipulation of polynomial by storing constant in the node of linked list.

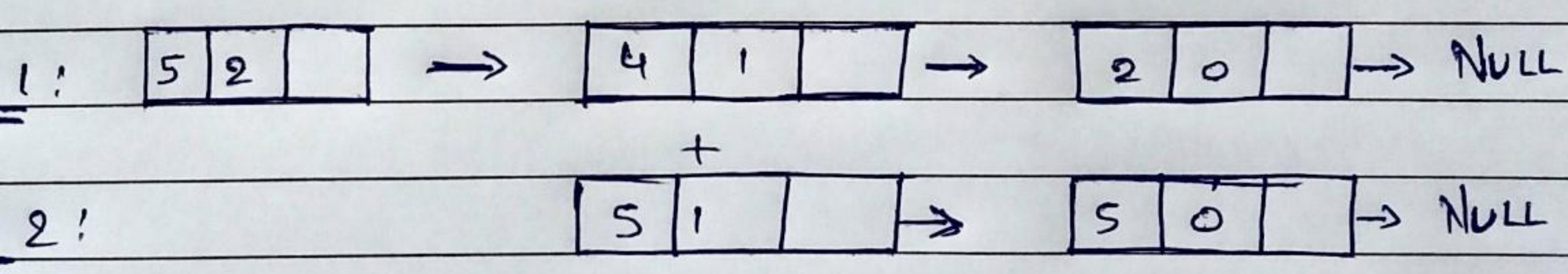
For adding two polynomials that are stored as a linked list, we need to add the coefficients of variables with the same powers. In a linked list, node contains 3 members, coefficient value, power and link to the next node.

For example :- Consider two polynomials;

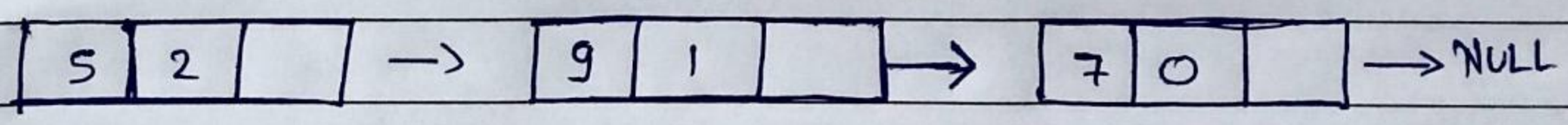
1st → $5x^2 + 4x^1 + 2x^0$

2nd → $5x^1 + 5x^0$

The addition of these two polynomials will result in:-



Resultant List



Output :- $5x^2 + 9x^1 + 7x^0$