DSA Assignment.

Q2) write a c progoam to convert infix expression to postfix expression. pass a one-dimensional character array char infix[] as input and print char postfix[]. Test your program for following input

input : (A-(B/C)*D+E) *F%GL

RUTVIK AVINASH BARBHAI
CSE-CORE-A
22580522
DSA ASSIGNMENT

Code

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

struct stack {
    int top;
    unsigned capacity;
    char *array;
};

struct stack* createStack(unsigned capacity) {
    struct stack* stack = (struct stack*)malloc(sizeof(struct stack));
    stack->capacity = capacity;
    stack->top = -1;
    stack->array = (char *)malloc(stack->capacity * sizeof(char));
    return stack;
}

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

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

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

```
}
return 0;
}

char peek (struct stack* stack) {
    if (!isEmpty (stack)) {
    return stack -> array [stack -> top];
    }
return 0;
}

int isOperator (char c)
    return (c == '+' || c == '-' || c == '*' || c == '/' || c == '%' ||
        c == '^');

int precedence (char c) {
    if (c == '^') return 3;
    if (c == '*' || c == '\' || c == '%') return 2;
    if (c == '+' || c == '-') return 1;
      return 0;
}
//convert infix to postfix expression
void infixToPostfix (char infix [], char postfix []) {
struct Stack* stack = createStack (strlen (infix));
    int i, j;
  i = j = 0;
 while (infix [i] != '\0') {
   char c = infix [i];
      if (isalnum (c)) {
       postfix [j++] = c;
   } else if (c == '(') {
        push (stack, c);
   } else if (c == ')') {
   while (!isEmpty (stack) && peek (stack) != '(') {
            printf ("Invalid expression \n");
              return;
```

```
} else {
    pop(stack);
    }
} else {
while (!isEmpty(stack) && precedence(c) <= precedence(peek(stack))
    {
     postfix[j++] = pop(stack);
        }
     push(stack, c);
    }
    i++;
   }
 while (!isEmpty(stack)) {
       postfix[j++] = pop(stack);
   }
postfix[j] = '\0';
  }
int main(){
  char infix[] = "(A-(B/C)*D+E)*F%G";
  char postfix[100];
printf("Infix Expression : %s\n", infix);
  infix To Postfix (infix, postfix);
  printf("Postfix Expression: %s\n", postfix);
   return (0);
   }
```

Input/output (sample)

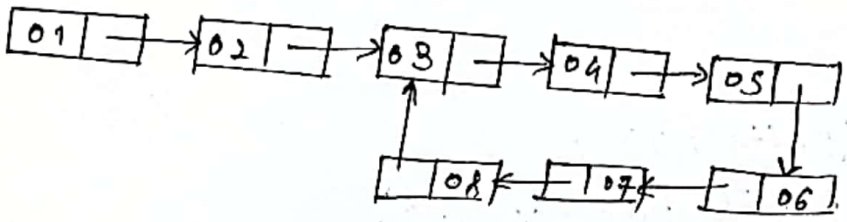
```
Enter the infix expression : (A-(B/C)* D+E )*F%G
    postfix expression : ABC/D*-E+F*G%
    Result of expression: 22.
```

Q5) Write a program to perform the following operations on Doubly linked-List:

a) Insertion: At the ~~bef~~ beginning, at the end, and at the given location in ~~the~~ sorted list

b) Deletion: first node, last node, and given item of node of given ptem from sorted list

c) Given a linked IPst with a loop/no-loop, write a C function to check whether the loop exists in the linked list or not




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

node *createNode(int data)
  {
  node *newNode = (node*) malloc(sizeof(node));
        newNode -> data = data;
        newNode -> prev = NULL;
        newNode -> next = NULL;

  return newNode;
}

void insertAtbeginning(node **head, int data)
  {
```

```
node *newNode = createNode(data);
if (*head == NULL)
    *head = newNode;
else {
  newNode -> next = *head
    (*head) -> prev = newNode
    *head = newNode;
  }
}
void insertAtEnd (node **head, int data)
  { node *newNode = createNode(data);
      if (*head == NULL)
        *head = newNode;
 else {
    node *temp = *head;
   while (temp -> next != NULL)
        temp = temp -> next;
   temp -> next = newNode;
   newNode -> prev = temp;
   }
}
void insertInSorted order (node** head, int data)
{
  node *newNode = createNode (data);
 if (*head == NULL || data <= (*head) -> data) {
     newNode -> next = *head;
      if (*head != NULL)
        (*head) -> prev = newNode;
  }
else {
   node *current = *head
 while (current -> next != NULL && current -> next -> data <
  data)
   {
    current = current -> next;
   }
```

```
newNode -> next = current -> next;
  if (current -> next != NULL)
    current -> next -> prev = newNode;
current -> next = newNode;
newNode -> prev = current;
  }
}
void deleteFirstNode (node **head)
{
  if (*head == NULL)
printf ("List is empty. cannot delete \n");
else {
  node *temp = *head;
    *head = (*head) -> next;
    if (*head) -> prev = NULL;
    free (temp);
  }
}
void deleteLastNode (node **head)
{ if (*head == NULL)
  printf ("List is Empty, cannot delete. \n");
  else if ((*head) -> next == NULL){
    free (*head);
    *head = NULL;
  }
else {
  node *current = *head;
while (current -> next != NULL)
    current = current -> next;
current -> prev -> next = NULL;
    free (current);
  }
}
```

```
void deleteNodewithpata(node **head, int data)
  {
   node *current = *head;
while (current != NULL && current -> data != data)
          current = current -> next;
  if (current == NULL)
   printf ("Node with data %d not found.\n", data);
else{
     if (current -> prev == NULL) {
     *head = current -> next;
if (current -> next != NULL)
          current -> next -> prev = NULL;
  }
   else if (current -> next == NULL)
          current -> prev -> next = NULL;
else {
    current -> prev -> next = current -> next;
   current -> next -> prev = current -> prev;
}
free (current);
   }
}
  int hasloop (node *head)
 { if (head == NULL)
       return 0;
node * slow = head;
node * fast = head;
while (fast != NULL && fast -> next != NULL) {
         slow = slow -> next;
         fast = fast -> next -> next
          if (slow == fast)
            return 1;
    }
  return 0;
 }
```

```
int main() {
  node *head=NULL
  int choice, data;
printf ("\n----------\n");
printf ("\nDOUBLY LINKED LIST MENU\n");
printf ("\n----------\n");
printf ("1. Insert at Beginning\n 2. Insert at End\n
3: Insert in sorted order\n: 4: Delete first node\n:
5: Delete Last Node:\n : 6: Delete node by data\n:
7 : Check for loop\n 8: Exit\n");

printf ("Enter your choice: ");
 scanf ("%d", &choice);
  switch(choice) {

case 1: printf ("Enter value to insert : ");
      scanf ("%d", &data);
   insertAtBeginning (&head, data);
      break;

case 2: printf ("Enter value to insert : );
      scanf ("%d", &data);
   insertAtEnd (&head, data);
      break;

case 3: printf ("Enter value to be inserted: ");
      scanf ("%d", &data);
      Insert In Sorted order (&head, data);
      break;

case 4: deleteFirstNode (&head)
      break;

case 5: deleteLastNode (&head)
      break;

case 6: if (hasloop (head))
```

```
printf("The linked list has a loop\n");
else
printf("The linked list does not have a loop\n");
break

case 6: printf("Enter value to delete: ");
        scanf("%d", &data);
        deleteNodewithData(&head, data);
        break

case 7: printf("Exit");
default:
    printF("Invalid choice: \n");
    }
}
return 0;
}
```

Input/output (sample)

```
DOUBLY LINKED LIST
1. Insert at Beginning
2. Insert at End
3. Insert in Sorted order
4. Delete First Node
5. Delete Last Node
6. Delete a Node by data
7. Check for loop
8. Exit

Enter your choice: 1
Enter value to Insert: 10

Enter your choice: 3
Enter value to Insert: 20

Enter your choice: 2
Enter value to insert: 30

Enter your choice: 2
Enter value to insert: 40

Enter your choice: 3
Enter value to be inserted: 50

Enter your choice: 7
The linked list does not have a loop.

Enter your choice: 8

Thto Exit
```

RUTVIK AVINASH BARBHAI
22205222

__/__/____

D.S.A ASSIGNMENT

Q 8 A) Explain Double ended queue:

A8A) A double ended queue is a data structures capable of performing insestions and deletion at both ends. It provides flexible means of managing data as

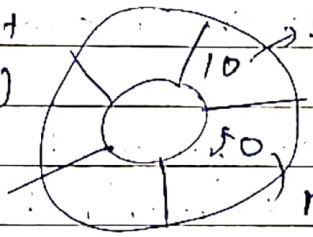
1) passing arithematic expression
2) Algorithm like breadth first seaarch
3) function as queue & stack.

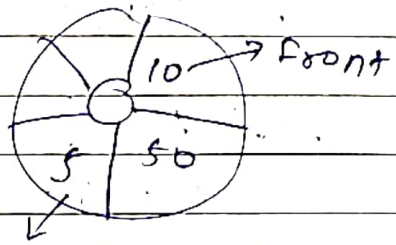
Double ended queues can also be used in form of priority queue or circular queue.

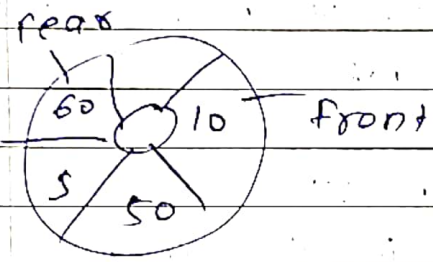
Q 8 B) Given circular queue of size = 5.

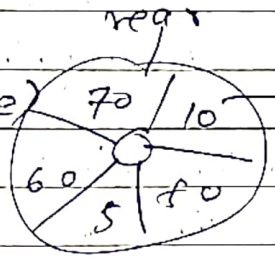
a) Enqueue (10)
b) Enqueue (50)
c) Enqueue (5)
d) Enqueue (60)
e) Enqueue (70)
f) Dequeue()
g) Dequeue()
h) Enqueue (90)
i) Dequeue()
j) Enqueue (80)
k) Dequeue()
l) Dequeue()

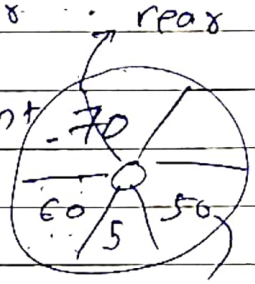
a) 

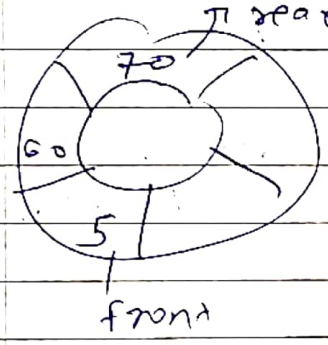
b) 

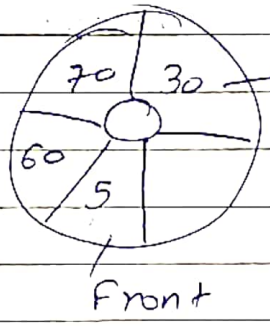
c) 

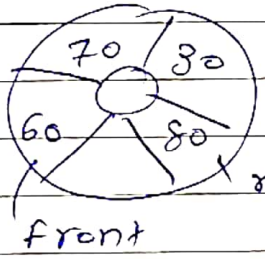
d) 

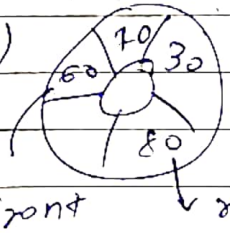
e) 

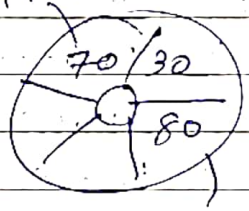
f) 

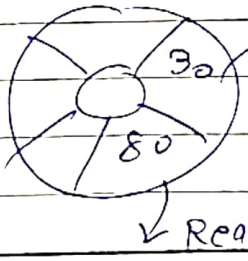
g) 

h) 

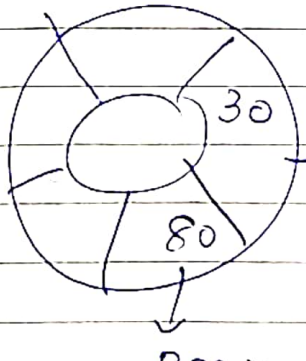
i) 

j k) 

k) 

l) 

Answer



Q 8C) Simulate pizza orders using Circular Queue

A8C)

```
#include <stdio.h>
#include <stdlib.h>
#define max 5

struct queue {
    int of, n;
    int size;
    int *arr;
};
struct Que * create (int size) {
struct Que* q = (struct Que*)malloc(sizeof
(struct Que));

 q -> size = size;
 q -> f = Que -> n - 1;
 q -> arr = malloc(q -> size * int);
 return q;
}
int isFull (struct Que *q) {
 return ((q -> r+1) % size == q -> f);
 }
int isEmpty(struct Que*q) {
 return(q -> f == -1);

void enque (struct que *q, int data) {
 if (isFull(q)) {
  printf ("Queue is full \n");
}
```

__/__/____

Q 3C) Simulate pizza orders using Circular Queue

A3C)

```
#include <stdio.h>
#include <stdlib.h>
#define max 5

struct Queue {
    int f, r;
    int size;
    int *arr;
};
struct Que * create (int size) {
struct Que* q = (struct Que*)malloc(sizeof
(struct Que));

 q -> size = size;
 q -> f = que -> r - 1;
 q -> arr = malloc(q -> size * int);
 return q;
}
int isFull (struct Que *q) {
 return ((q -> r+1) % size == q -> f);
 }
int isEmpty(struct Que*q) {
 return(q -> f == -1);

void enque (struct que *q, int data) {
 IF (isFull(q)) {
  printf ("Queue is full\n");
}
```

```
if (isEmpty (q)) {
        q -> f = 0;
}
q -> r = (q -> r+1) %(q -> size);
q -> arr [q -> r] = data;
}

void deque(struct que * q) {
 if (isEmpty (q)) {
 printf ("Queue is Empty \n");
   }
if (q -> F == q -> r)

        q -> f = -1
        q -> r = -1
printf ("Queue is empty now \n");

} else {
     q -> F = (q -> F+1) %(q -> size);
 }
}
void display (struct que * q) {

if (isEmpty (q)) {

 printf(" Queue is Empty \n")
 {
int i = q -> f q -> f;
printf (" Queue is : \n")
  for (i = f ; i != (q -> r+1) %(q -> size);
(i+1) %(q -> size)) {
```

```
printf("%d", q->arr[i]);
    } printf("\n\n\n")
}

void main() {
struct Que *q = create(max);
 int ch, data;
while(1>0) {
printf("\t\t\t PIZZA ORDER\n\t1. Place order\n\t 2. Serve order\t 3:
Display order\n\t4: Exit\n\n");

printf("\n Enter choice ");
scanf("%d", &ch);

switch(ch) {
case 1: printf("\n Enter order no: ");
        scanf("%d", &data);
       enque(q, data)
       printf("\n order placed \n");
        break;
case 2:  deque(q)
        printf("\n order served \n");
        break;
case 4
case 3: printf("\n Orders Displayed \n");
        display(q)
        break;
case 4: exit(1);
   }
```