WEEK6:

Q1.

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

#define SIZE 10

#define UNDERFLOW\_INT -32767

typedef struct AscQueue { int \*arr;

int front, rear; int capacity;

} ASC\_QUEUE\_t;

typedef ASC\_QUEUE\_t \* ASC\_QUEUE\_p\_t;

void initQueue (ASC\_QUEUE\_p\_t queue) { queue->arr = (int \*)calloc(SIZE, sizeof(int)); queue->front = queue->rear = -1;

queue->capacity = 0;

}

int isQueueFull (ASC\_QUEUE\_t queue) { if (queue.capacity == SIZE)

return 1;

return 0;

}

int isQueueEmpty (ASC\_QUEUE\_t queue) { if (queue.capacity == 0)

return 1;

return 0;

}

void insert (ASC\_QUEUE\_p\_t queue, int item) { if (isQueueFull(\*queue)) {

printf("QUEUE FULL!\n"); return;

}

if (isQueueEmpty(\*queue)) { queue->front = 0;

queue->rear = 0;

\*(queue->arr) = item;

}

else {

int i, pos = 0;

for (i = queue->front; i <= queue->rear; ++i) if (\*(queue->arr + i) > item)

break; pos = i;

queue->rear += 1;

for (i = queue->rear; i > pos; --i)

\*(queue->arr + i) = \*(queue->arr + i - 1);

\*(queue->arr + pos) = item;

}

queue->capacity += 1;

}

int delete (ASC\_QUEUE\_p\_t queue) { if (isQueueEmpty(\*queue)) { printf("QUEUE EMPTY!\n");

return UNDERFLOW\_INT;

}

queue->capacity -= 1;

return \*(queue->arr + (queue->front)++);

}

void display (ASC\_QUEUE\_t queue) { int i;

printf("\nCURRENT QUEUE:\n");

for (i = queue.front; i <= queue.rear; ++i) printf("%d\t", \*(queue.arr + i)); printf("\n");

}

int main (int argc, const char \* argv []) {

ASC\_QUEUE\_p\_t queue = (ASC\_QUEUE\_p\_t)malloc(sizeof(ASC\_QUEUE\_t)); initQueue(queue);

char choice;

do {

printf("\n \n"); printf("0.Quit\n1.Insert\n2.Delete\n3.Display Queue\nEnter choice : \n"); scanf(" %c", &choice);

int item;

if (choice == '1') {

printf("Enter item to be inserted: \n"); scanf("%d", &item);

insert(queue, item);

}

else if (choice == '2') { item = delete(queue);

if (item != UNDERFLOW\_INT) printf("Deleted item: '%d'\n", item);

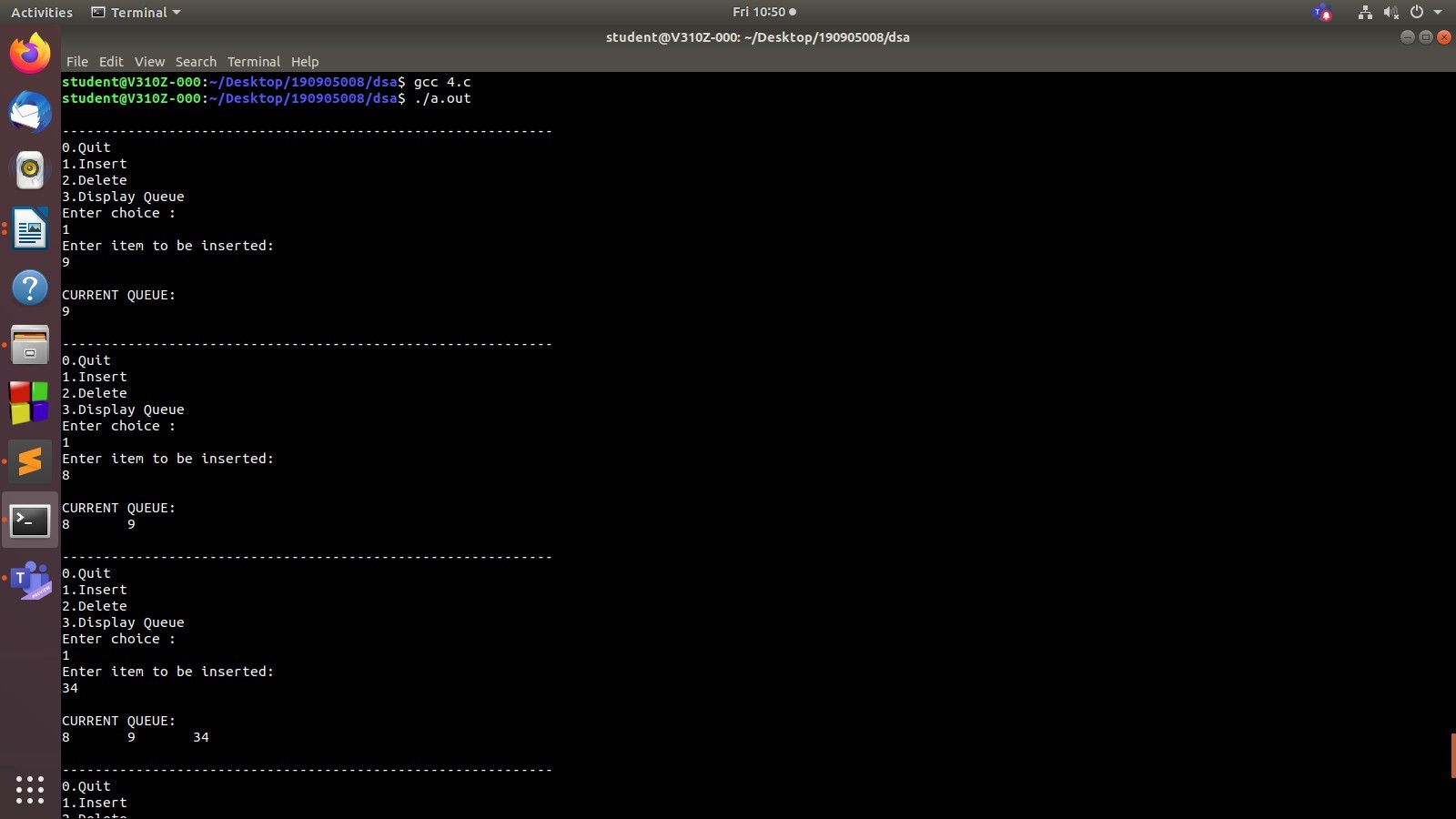
}

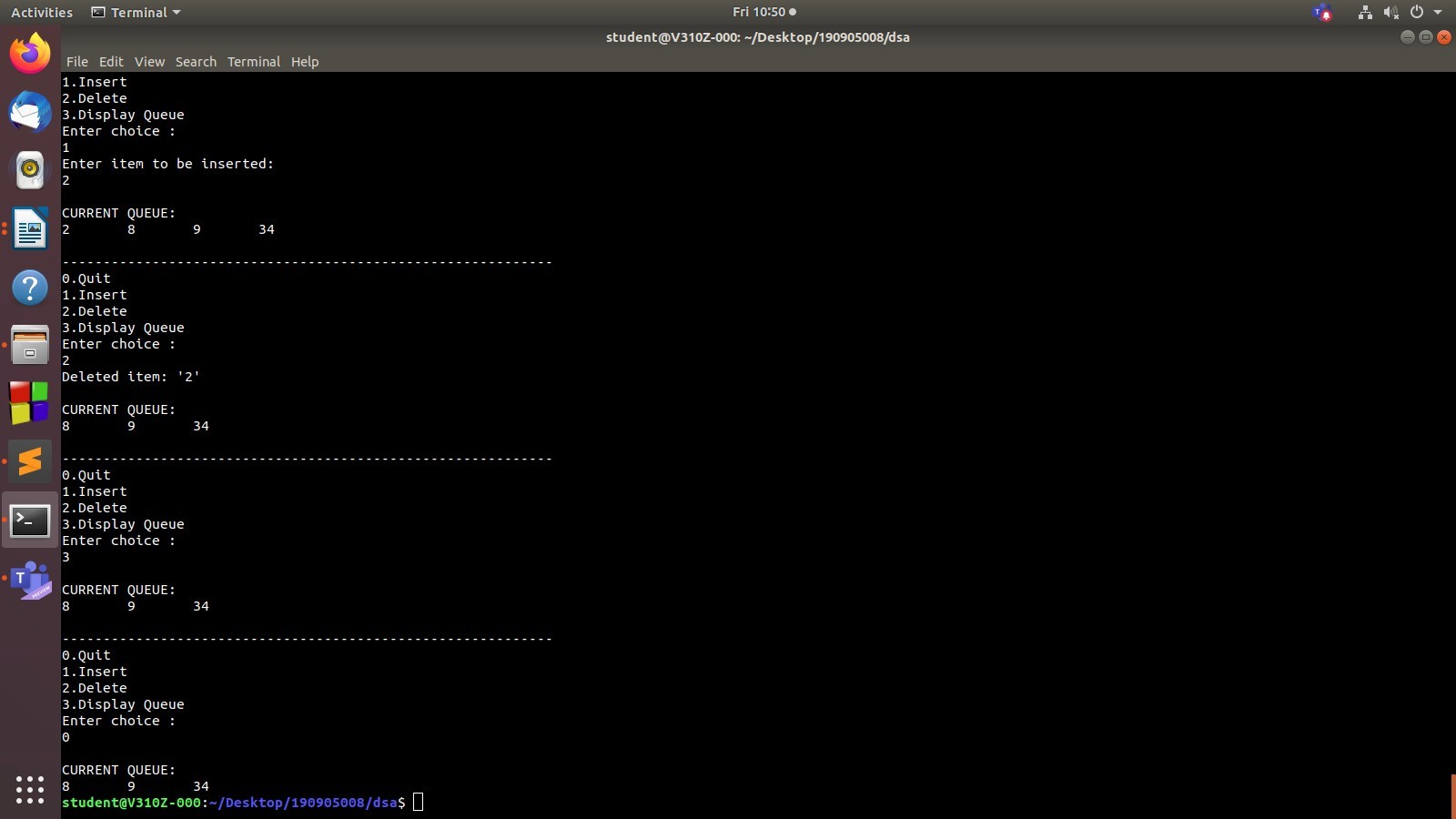
display(\*queue);

} while (choice == '1' || choice == '2' || choice == '3');

return 0;

}





Q2.

#include <stdio.h> #include <string.h> # define MAX 20 typedef struct

{

char x[MAX][MAX]; int front;

int rear;

} Q;

Q q;

void enqueueRight(char []); void display();

void dequeueLeft();

void enqueueLeft(char []);

int main()

{

q.front = -1;

q.rear = -1;

int ch, flag = 1; char x[MAX]; while (flag)

{

printf("1. Enqueue Left\n2. Enqueue Right\n3. Dequeue\n4. Display\n5. Exit\n");

scanf("%d", &ch); switch (ch)

{

case 1:

printf("Enter element\n"); scanf("%s", x); enqueueLeft(x);

break;

case 2:

printf("Enter element\n"); scanf("%s", x); enqueueRight(x);

break; case 3:

dequeueLeft(); break;

case 4: display(); break;

default : flag = 0; break;

}

}

return 0;

}

void enqueueRight(char x[])

{

if (q.rear == MAX) printf("Full from right\n");

else

{

strcpy(q.x[++q.rear], x); if (q.front == -1)

q.front = 0;

}

}

void enqueueLeft(char x[])

{

if (q.rear == -1 && q.front == -1) enqueueRight(x);

else if (q.front == 0) printf("Full from left\n");

else

strcpy(q.x[--q.front], x);

}

void dequeueLeft ()

{

if (q.rear == -1 && q.front == -1)

{

printf("Empty dequeue\n"); return ;

}

else if (q.front == q.rear)

{

printf("Deleted element is %s\n", q.x[q.front]);

q.front = q.rear = -1;

}

else

printf("Deleted element is %s\n", q.x[q.front++]);

}

void display()

{

int i;

if (q.front == -1 && q.rear == -1) printf("Empty dequeue\n");

else

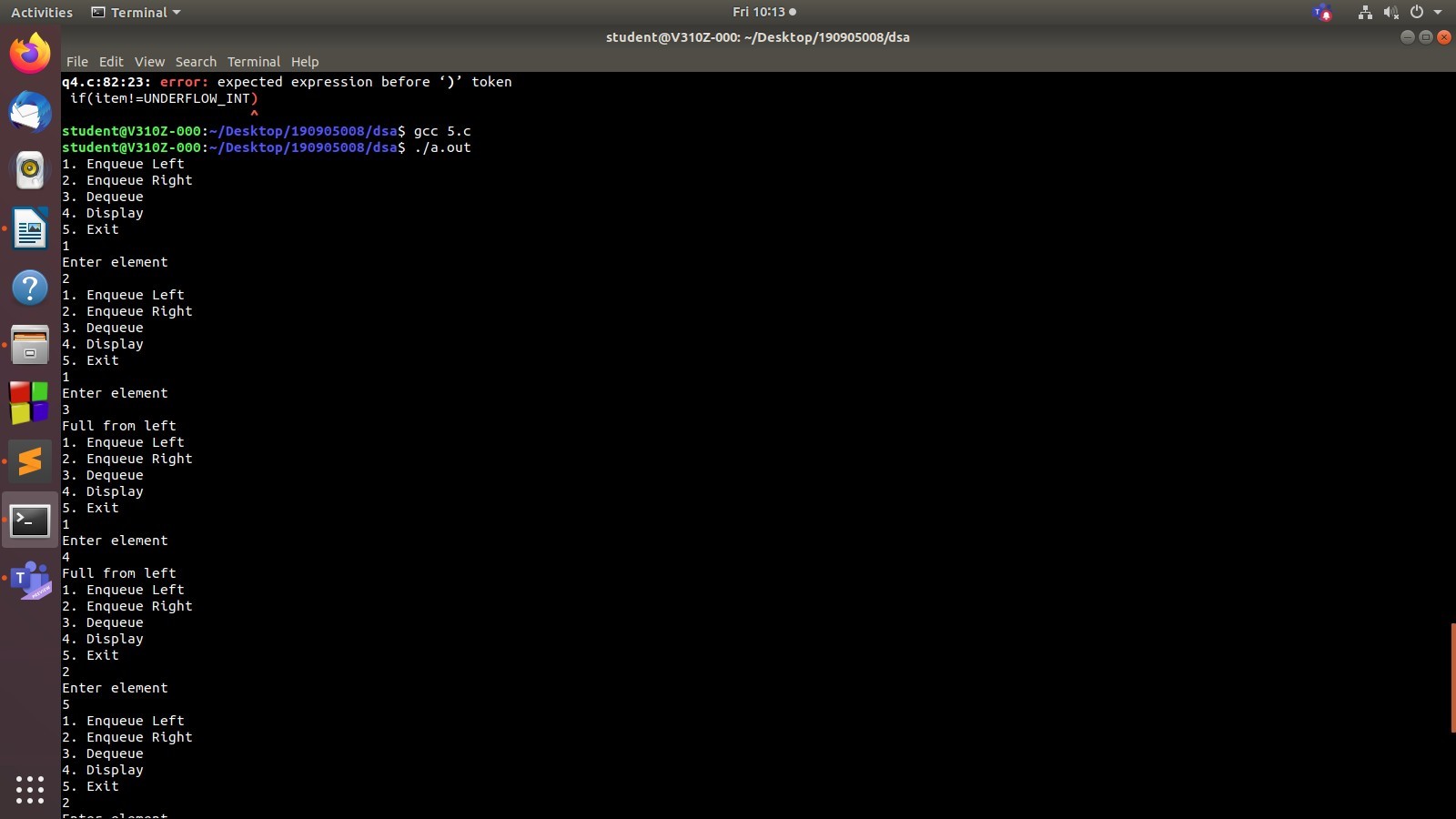
{

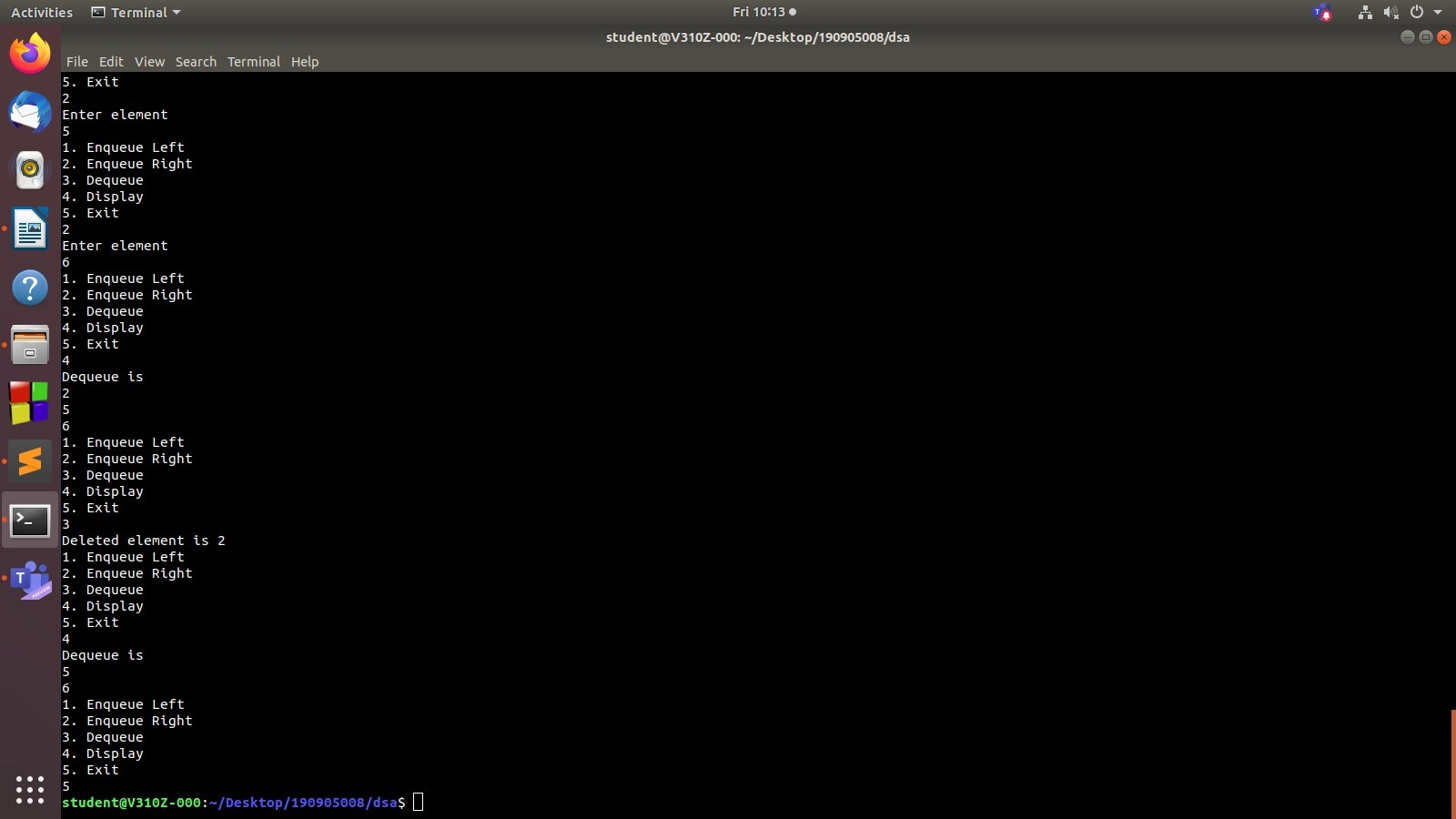
printf("Dequeue is\n");

for (i=q.front; i<=q.rear; i++) printf("%s\n", q.x[i]);

}

}





Q3.

#include <stdio.h> #include <stdlib.h> #include <string.h> #define MAX 30 typedef struct dequeue

{

char data[MAX]; int rear,front;

}dequeue;

void initialize(dequeue \*P)

{

P->rear=-1; P->front=-1;

}

int empty(dequeue \*P)

{

if(P->rear==-1) return(1); return(0);}

int full(dequeue \*P)

{

if((P->rear+1)%MAX==P->front) return(1);

return(0);

}

void enqueueR(dequeue \*P,char x)

{

if(empty(P))

{

P->rear=0; P->front=0;

P->data[0]=x;

}

else

{

P->rear=(P->rear+1)%MAX; P->data[P->rear]=x;

}

}

void enqueueF(dequeue \*P,char x)

{

if(empty(P))

{

P->rear=0; P->front=0;

P->data[0]=x;

}

else{

P->front=(P->front-1+MAX)%MAX; P->data[P->front]=x;

}

}

char dequeueF(dequeue \*P)

{

char x;

x=P->data[P->front];

if(P->rear==P->front) initialize(P);

else

P->front=(P->front+1)%MAX; return(x);

}

char dequeueR(dequeue \*P)

{

char x;

x=P->data[P->rear]; if(P->rear==P->front) initialize(P);

else

P->rear=(P->rear-1+MAX)%MAX;

return(x);

}

void print(dequeue \*P)

{

if(empty(P))

{

printf("\nQueue is empty!!");exit(0);

}

int i;

i=P->front; while(i!=P->rear)

{

printf("\n%c",P->data[i]); i=(i+1)%MAX;

}

printf("\n%c\n",P->data[P->rear]);

}

void main()

{

int i,x,n; int op=0; char c[20]; dequeue q;

initialize(&q);

printf("Enter string to check for palindrome\n"); scanf("%s",c);

n= strlen(c); for(i=0;i<n;i++)

{

enqueueF(&q,c[i]);

}

for(i=0;i<n/2;i++)

{

if(dequeueF(&q)!=dequeueR(&q))

{

op = 1; break;

}

}

if(op == 0)

printf("%s is palindrome\n",c); else

printf("%s is not palindrome\n",c);

}

