Date: / /

1) write a program to simulate the working of the que of integers using arrays. provide the following operations enque, deque, display

```
# include <stdio.h>
# include <math.h>
# include <string.h>
# define N 5
 int que[N];
 int front = -1;
 int rear = -1;
 void enque (int n) {

  if (rear == N-1) {
      printf("overflow");
 }
   else if (front == -1 && rear == -1)
 {
  front = rear = 0;
  que[rear] = x;
 }
 else {
      rear++;
      queue[rear] = x;
  }
 }
  void deque()
 {
  if (front == -1)
{
 ~~if(front=~~
  printf("underflow");
}
```

```
}
 else if (front == rear)
{
   print ("the deque element is %d", queue[fr
  front = rear = -1;
 }
 else {
 printf ("the deques element is %d" queue[fron
}
 }

void   display () {

for (int i = front; i <= rear; i = i+1)
{
   print (" %d", que [i]);
 }
 }
 int  main ()
{
 int  ch, x;
while (ch != 0)
{ print (" Enter 1: enque  2: deque \n
            3: display \n 4: teminate
       program");
scanf (" %d", &ch);
switch (ch) {
  case 1 :   printf (" Enter value :")
             scanf (" %d", &x);
             enque(x);
             break;

  case 2 :   deque();
             break;

  case 3     display ();
             break;

  case 0 :   printf (" terminating program");
             break;

default:  printf (" Invalid input"); break.
  }
  }
  return 0;
}
```

2) WAP to ~~simulate~~ Simulate the working of a circular queue using array. provide the following operation. Insert, delete & display the program should print appropriate message for queue empty and overflow condition

```
#include <stdio.h>
# include <math.h>
# define N
int queue [N];
int front = -1;
int rear = -1;

void enque (int n)
{
  if (front = = -1 && rear = = -1)
{
     front = rear = 0;
   que [rear] = n;
 }
   else if (((rear + 1) % N ) = = front)
{
  printf ("overflow ");
}
else {
     rear++ ;
   que [rear] = n;
}
```

Linked list

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

struct Node {
 int data
 struct Node* next;
};
 struct node * createNode (int data)
{
 struct Node * new node = (struct node*)
malloc (sizeof (struct Node));
 if(new node== Null)
{
 printf("memory allocation failed \n");
 exit (1);
}
New node -> data = data;
New node -> next = Null;
 return new node;
}
struct node * createLinkedlist (int values [],
int size )
{
  struct node * head = Null;
  struct node * tail=Null;

 for (int i = 0; i<size ; i++)
{
  struct node * New node = createNode
 (values[i]);
switch(ch) {

 case 1 : printf("Enter value:");
          scanf("%d", &n);
          enque(n)
          break;
 case 2 :  dequeue();
           break;

 case 3 :  display();
            break

 case 0 :  printf("terminated program");
              break

 default :  printf("invalid input");
             break;
 }
}
return 0;
}
```

8/11/24

```
void deque()
{
if (front = = -1 && rear = = -1)
{
  printf ("overflow");
}
else if ((front +1) % N) = = rear)
{
  printf (" the dequed element is
    queue [front]);
  front = rear = -1;
}
 else {
        printf ("the dequed element is %d"
  que [front])
   front = (front +1) % N;
  }
  }

void display () {

for (int i = front ; i != rear;
  i = (i +1) % N) {
   print (" %d ", que [i]);
  }
  print (" %d", queue[rear]);
}

int main () {
 int ch, x;
while (ch != 0) {
printf ("enter 1: enque 2: deque
             3: display");
 scanf ("%d", &ch);
if (head = = null)
{
  head = new node;
  tail = new node;
}
  else {
   tail -> next = newnode;
   tail = new node
 }
}
  return head;
}

void insert first (struct node ** head, int data)
{
 struct node * new node = create Node
 (data);
 new node -> next = * head;
 * head = new node;
}

void insert at position (struct Node ** head, int data
int position)
{
 if (possition = =0)
{
 insert first (head, data);
 return;
}
struct Node * new node = create Node (data);
struct Node * current = * head;
```

```
for(int i=0; i< position-1 ; i++)
{
  if (current == null)
  {
    printf ("invalid position \n");
    return;
  }
  current = current -> next;
}
  new_node -> next = current -> next;
  current -> next = new_node;
}
  void insert end (struct Node ** head,
int data)
{
    struct node* new node = create node(data);

 if (* head == null)
{
 * head = new_node;
 return;
}
struct Node * current = * head;
while (current -> next != null)
{
  current = current -> next;
}
  current -> next = new_nod;
}
```

Date: / /

```
void display (struct Node* head)
{
 while (head!=null)
 {
  printf ("%d ->", head -> data);
  head = head -> next;
 }
 print f ("Null\n");
}

int main ()
{
 int values [] = {1,2,3,4};
 int size = size of(values) / size of(values[0]);

struct Node* linked list = createLinkedList
(values, size);

insert first (&linked list, 1);
insert at position (&linkedlist, 2.5, 2);
insert end (&linkedlist, 12);
display (linkedlist);
return 0;
}
```

-> 1->2->2->3->4-5 3 12

22/11/2