write a program to simulate the working of the queue of integers using arrays. provide the following operations enqueue, dequeue, 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] = n;
 }
 else {
     rear++;
      que [rear] = n;
  }
}
  void deque()
{
  if (front == -1)
{
 th(front=
```

```
}
else if (front == rear)
{
  printf("the deque element is %d", que[front]);
  front = rear = -1;
}
else {
printf("the deqeued element is %d" que[front]);
}
}

void display () {

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

```
switch (ch) {
 case 1: printf ("Enter value:")
         scanf ("%d", &x);
         enqueue(x);
         break;

 case 2: dequeue();
         break;

 case 3: display();
         break;

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

 default: printf ("Invalid input"); brak
 }
 }
 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 queen[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;
}
```

```
void dequeue()
{
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",
  queue[front]) ~~to~~
  front = (front+1)%N;
  }
  }

void display(){

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

int main(){
  int ch, x;
  while(ch!=0){
  Printf("Enter 1: enque 2: deque
        3: display");
  Scanf("%d", &ch);
```

```
switch(ch){

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

case 3: display();
        break

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

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

Sun 8/1/24

O/P

Date: / /

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 * create linkedlist (int values [],
int size )
{
   struct node * head = Null;
   struct node * tail = Null;

  for (int i = 0; i < size; i++)
{
   struct node * new node = create Node
   (values[i]);
```

```
if (head == null)
{
  head = new node;
  tail = new node;
}
  else {
   tail -> next = new node;
   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 insertend(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_node;
}
```

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

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

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

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

Sam 22/1/24.

→ 1→2→2→3→4→5→12

Delete pending

Sorting list (Bubble Sort)

```
void sort list (struct Node * * head)
{
  struct node* current, *nextNode;
  int temp;
  current = * head;
  while (current != Null)
  {
    nextnode = current -> Next;
    while (NextNode != Null)
  {
    if (current -> data > nextNode -> data)
  {
    temp = current -> data;
    current -> data = nextnode -> data;
    next Node -> data = temp
  }
    nextNode = nextNode -> next;
  }
    current = current -> next;
  }
  }

void reverselist (struct node * * headRef)
{
  struct Node * prev, * current, * next
  prev = Null;
  current = * head Ref;
  while (current != Null)
  {
    nextnode = current -> next;
    current -> next = prev;
    Prev = current)
```

```
        current = nextNode;
    }
    *head_ref = prev;
}

void concatenateLists(struct Node **list1,
struct Node *list2)
{
    if (*list1 == NULL)
    {
        *list1 = list2;
        return;
    }

    struct Node *temp = *list1;
    while (temp->next != NULL)
    {
        temp = temp->next;
    }
    temp->next = list2;
}
```

Date: / /

queue

~~stack~~ implementation.

```
void enque (int value)
{
    struct node *ptr;
    ptr=(struct node*)malloc(sizeof(struct node));
```

14th Aug

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

struct node{
    int data;
    struct node* left;
    struct node* right;
};

struct node* create node(int data){
struct node* newnode=(struct node*)
 malloc(sizeof(struct node));
newnode->data=data;
newnode->left = newnode->right=NULL;
return newnode;

struct node * insert(struct * root, int data
{
    if(root==NULL){
      return (createnode(data);
    }
    if data < root->data){
    root->left = insert(root->left,data);
    }
      else if (data > root->date){
        root->right=insert(root->right,data);
    }
    return root;
}
```

```
void inorder traversal (struct node* root)
{
    if (root != NULL) {
    inorder traversal (root -> left);
    print f (" %d", root -> data);
    inorder traversal ( root -> right);
}
}

void postorder traversal ( struct node* root)
if (root != NULL) {
postorder traversal (root -> left);
post order traversal (root -> right
```

Breadth first search

```
#include<stdio.h>
#include<stdlib.h>
#define size 40

struct que {
  int items[size];
  int front;
  int rear;
};

struct queue * createQueue();
void enque(struct *q, int);
int

struct node{

   int vertex;
   struct node* next;
};

struct node* createnode(int);

struct graph {
int numvertices;
struct node ** adjlists
};

void BFS(struct Graph* graph, int start vertex)
{
    struct queue *q = createqueue();
```

```
graph -> visited [start vertex] = 1;
enqueue (q, start vertex);
while (!isempty (q)) {
printqueue (q);
int currentvertex = dequeue(q);
printf ("visited %d", currentvertex);

struct node* temp = graph -> adjLists
[currentvertex];

while (temp) {
  int adjvertex = temp -> vertex;
  if (graph -> visited [adjvertex] == 0) {
  graph -> visited [adjvertex];
  }
  temp = temp -> next;
  }
}
}
struct graph * creategraph (int vertices) {

struct graph * graph = malloc (sizeof (struct graph
graph -> numvertices = vertices

graph -> adjLists = malloc (vertices * sizeof (struct
graph -> visited = malloc (vertices * sizeof (int));

int i;
for (i=0; i< vertices; i++) {
graph -> adjLists (i) = NULL;
graph -> visited (i) = 0;
return graph
```