1) Create structure item details with members itemname, quality, price, total amount. Calculate party expenses

pgm:-

```
#include <stdio.h>
struct itemDetails {
        char name[10];
        int price; int quantity;
        int totalamount;
    } item[30]

void main ()
  {
    int a, i, Totalexpense=0;
    printf ("Enter the number of items \n");
    scanf (" %d", &a);
    for (i= 1; i<=a; i++) {
        printf ("Enter the name of item %d : \n", i);
        scanf ("%s", item[i]. name);
        printf (" Enter the quantity of item %d = \n", i);
        scanf (" %d", & item[i]quantity );
        printf (" Enter the price of item %d = \n", i);
        scanf (" %d", & item[i]. price);
        item[i]. total expense Amount = item[i]price * item[i]. quantity;
        Totalexpance += item[i] . totalamount;
    }
    printf (" The total expence of the party is
            %d ", Total expense);
}
```

Output:-

Enter the number of items.

2

Enter the name of item 1 =

starter

Enter the quantity of item 1 =

4

Enter the price of item 1

300

Total amount of item 1 is 1200

Enter the name of item 2

Main Course

Enter the quantity of item 2

8

Enter the price of item 2

250

Total amount of item 2 is 2000

The total expense of the party is 3200

4) Given an array arr[] containing N distances of the inch-feet system, such that each element of the array represents a distance in the form of {inch, feet}. The task is to add all the N inch-feet distances using structure.

```
#include <stdio.h>
struct distances {
        float arr[10][10];
} dist;
```

```
void main()
{
  int a, i, j;
  float feetsum=0, Inchsum=0;
  printf(" Enter the number of inch-feet pairs to be
       added \n ");
  scanf(" %d ", &a);
  printf(" Enter the pairs ");
  for (i=0 ; i<a ; i++)
    {
      for(j=0 ; j<2; j++) {
          scanf(" %f ", &dist.arr[i][j]);
      }
    }
    for(i=0 ; i<a ; i++)
      {
        Feetsum += dist.arr[i][0];
        Inchsum += dist.arr[i][1];
      }
    printf(" total feet sum= %f \n ", Feetsum);
    printf(" total Inch sum = %f \n ", Inchsum);
}
```

Output :-

Enter the number of inch-feet pairs to be added : 3

Enter the pairs : 2 2.5

3.8 4.2

3.1 4.8

total feet sum = 10.900000

total inch sum = 11.500000

2)

```
#include <stdio.h>
struct gradelist {
        char courseCode[8][10];
        int credits[8];
        char grade[8];
};
struct std{
        char name[20];
        char usn[10];
        struct gradelist sem1;
        struct gradelist sem2;
} std1;
int blcredits(struct std s){
    int bl=0;
    for(int i=0; i<8; i++) {
        if(s.sem2.grade[i]=='F') {
            bl= bl+ s.sem2.credits[i];
        }
    }
    return(bl);
};
int main(){
    printf("Enter student name:\n");
    scanf("%s", std1.name);
    printf("Enter the student usn\n");
    scanf("%s", std1.usn);
    printf("Enter course code, credits, and
        grades for sem1\n");
    scanf("for(int i=0; i<8; i++){
```

```
printf (" course code :");
scanf (" %s" , std1. sem1. course code [i]
printf ("credits")
scanf ("%d", & std1. sem1. credits[i]);
printf (" grade");
scanf (" %c", & std1. sem1. grade[i]);
}
:
Similarly for sem 2
:
int blc = blc credits (std1);
if (blc >= 16) {
    printf (" fail");
}
else {
    printf (" Pass")
}
return 0;
}
```

22/11/24

Stack

1) C-Program to implement Stack (using arrays (using Global variables)

```
#include <stdio.h>
# include < process.h>
# include < conio.h>
# define STACK_SIZE 5
 int top = -1;
 int s[10];
 int item;
 void push ()
 {
  if (top == STACK_SIZE -1)
    {
    print (" stack overflow \n")
    return;
    }
   top = top+1
   s[top] = item
  }
 int pop()
 {
   if (top == -1) return -1;
   return s[top--];
 }
 void display
 {
  int i;
  if (top == -1) {
```

```
printf("Stack is empty \n");
return;
}
printf("Contents of the stack \n");
for(i=0 ; i<=top ; i++)
{
 printf("%d \n", s[i]);
}
}
void main()
{
 int item_deleted;
 int choice;
 ~~scanf("%d~~
 clrscr();
 for(;;)
 {
 printf("\n 1: push\n 2: pop \n 3: display \n 4:exit \n")
 printf("Enter the choice \n");
 scanf("%d", &choice);
 switch(choice)
 {
 case 1: printf("enter the item to be inserted \n");
 scanf("%d", &item);
 push();
 break;
 case 2: item_deleted = pop();
 if(item_deleted==-1)
 printf("Stack is empty \n");
```

```
        else
            printf("\nItem deleted is %d \n", item_deleted);
            break
        case 3: display ();
                break;
        default : exit(0);
        }
    }
    getch()
    }
```

2) WAP to convert infix to postfix expression

```
#include <stdio.h>
#include <string.h>
int index1=0, pos=0, top=-1, length, pred(char symbol);
char symbol, temp, infix[20], postfix[20], stack[20];
void infix to postfix ()
{
    length = strlen(infix);
    push ('#');
    while (index1 < length)
    {
        switch (symbol)
        {
            case '(': push (symbol);
                      break;
            case ')': temp = pop()
                      while (temp != '('
```

```
{
postfix[pos] = temp
pos++;
temp = pop();
}
break;
case '+':
case '-':
case '*':
case '/':
case '^': while (pred(stack[top]) >= pred(symbol))
{
temp = pop()
postfix[pos] = temp;
pos++;
(temp = pop())
}
push(symbol);
break;
default: postfix[pos++] = symbol;
}
index++;
}
while (top>0)
{
temp= pop();
postfix[pos++]= temp;
}
}
```

```
void push (char symbol)
{
  top = top+1;
  stack[top] = symbol;
}

char pop()
{
   char symb;
   symb = stack[top];
   top = top-1;
   return (symb);
}

int pred (char symbol)
{
   int p;
   switch symbol
   {
    case '^' : p=3;
               break
    case '/' :
    case '*' : p=2;
               break
    case '+' :
    case '-' : p = 1;
               break;
    case '(' : p=0;
               break
```

```
        case '#' : p = -1;
                   break;
        }
    return(p);
}
```

3) C-program to implement Linear Queue (Ordinary Queue)

```
#include <stdio.h>
#include <conio.h>
#include <process.h>
#define Que_size 5
int item, front=0, rear=-1, q[10];
void insert rear()
 {
  if (rear = Que_size -1)
    {
     printf("queue overflow\n");
     return;
    }
    rear = rear+1;
    q[rear] = item;
  }
int delete_front()
{
 if (front > rear) retur -1;
  return q[front++];
}
```

```
void displayQ()
{
 int i;
 if (front > rear)
 {
  printf(" queue is empty \n ");
  return;
 }
 printf(" contents of queue \n ");
 for (i=front ; i<=rear ; i++)
 {
   printf(" %d \n ", q[i]);
 }
} }
void main()
{
 int choice;
 clrscr();
 for (;;)
 {
  printf("\n 1: insert rear, \n 2: delete front \n
          3: display \n 4: exit \n");
  scanf(" %d ", &choice);
  switch (choice)
  {
    case 1: printf(" enter the item to be inserted
            scanf(" %d ", &item);
            insertrear();
            break;
```

```
        case 2: item = deletefront();
                if(item == -1)
                    printf("queue is empty \n");
                else
                    printf("Item deleted = %d \n", item);
                break;
        case 3: display();
                break;
        default: exit(0);
        }
    }
    getch()
}
```

4) C-program to implement using Circular Queue (using global variable)

```
#include <stdio.h>
#include<conio.h>
#include <process.h>
#define Que_size 3
int item, front=0, rear=-1, q[Que_size], count=0;
void insert_rear()
{
  if(count == Que_size)
    {
     printf("queue overflow \n");
     return;
    }
```

```
    rear = (rear + 1) % Que_size;
    q[rear] = item;
    count++
}
int delete_front()
{
if (count == 0) return -1;
item = q[front]
front = (front+1) % Que_size
count = count - 1;
return item;
}
void display()
{
int i, f;
if (count == 0)
{
printf("queue is empty \n")
return;
}
f = front
printf("contents of queue \n")
for (i=1, i<= count; i++)
{
    print("%d \n", q[f]);
    f = (f+1) % Que_size;
}
}
```

```
void main()
{
 int choice;
 clrscr();
 for(;;)
 {
 printf("\n 1: insert rear \n 2: delete front \n 3: display \n 4: exit\n");
 printf("enter the choice \n");
 scanf("%d", &choice
 switch (choice)
 {
  case 1: printf("enter the item to be inserted \n");
          scanf("%d", &item);
          insertrear();
           break;
  case 2: item = deletefront();
          if (item == -1)
              printf("queue is empty \n");
          else
              printf("item deleted = %d", item
          break
  case 3: displayQ();
          break;
  default : exit(0)
 }
}
getch();
}
```

53

Lab 11 - Linked List

~~Output (circular que~~

```
#include <stdio.h>
#include <stdlib.h>
#include <stdlib.h>
#include <malloc.h>
struct node {
        int data;
        struct node *next;
};
struct node *start = NULL;
Struct node *insert_beg (struct node *);
struct node *insert_end (struct node *);
struct node *delete_beg (struct node *);
struct node *delete_end (struct node *);

struct node *insert_beg (struct node *);
{
struct node *newnode;
int num;
printf ("Enter the data");
scanf ("%d", &num);
newnode -> data = num;
newnode -> next = start;
start = newnode;
printf ("Inserted at the begining");
return start;
}
```

```
struct node * insert_end (struct node *);
{
 struct node * newnode;
 struct node * ptr ;
 int num;
 printf (" Enter data to be inserted at the end");
 scanf (" %d ", &num);
  ptr = start ;
  newnode → data = num ;
 while (ptr → next != NULL ) {
        ptr = ptr → next ;
 }
  ptr → next = newnode ;
printf (" Inserted in the ~~beginning~~ end ");
 return start;
}


struct node * delete_beg (struct node * );
{
 struct node * newnode
 struct node * ptr;
 int num;
 ~~p~~ ptr = start;
  ~~start = start → next~~
  if (ptr → next == NULL)
       printf (" No data to be deleted ");
  else {
       start = start → next ;
       free (ptr);
       printf (" deleted at the begining ");
```

```
struct node * delete_end (struct node *){
struct node * ptr;
struct node * ptr1;
ptr = start;
if (start -> next
while ( ptr -> next != NULL){
        ptr1 = ptr;
        ptr = ptr -> next;
    }
ptr1 -> next = NULL;
free(ptr);
}if (start -> next == NULL)
        printf ("no data to be deleted");
else {
        while (ptr -> next != NULL){
            ptr1 = ptr;
            ptr = ptr -> next;
        }
        ptr1 -> next = NULL;
        free (ptr);
        printf ("Deleted at the end");
        return start;
}
```

```
int main ()
{
    int option;
    while (option != 6) {
        printf (" select an option: ");
        printf ("1 for insertion at the bagining ");
        printf (" 2 for insertion at the end ");
        printf (" 3 for deletion at the begining ");
        printf (" 4 for deletion at the end ");
        printf (" 5 for display ");
        printf (" 6 for exit");
        scanf (" %d ", & option);
        switch option {
            case 1 : start = insert_beg (start);
                break;
            case 2 : start = insert_end (start);
                break;
            case 3 : start = delete_beg (start);
                break;
            case 4 : start = delete_end (start);
                break
            case 5 : start = display (start);
                break;
            default : printf (" Invalid Input ");
        }
}return 0;
}
```

```
struct node * display (struct node * start)
{
    struct node *ptr;
    ptr = start;
    if (ptr -> next == NULL) {
        printf("Empty list");
        return start;
    } else {
        while (ptr -> next != NULL)
        {
            printf("%d", ptr -> data);
            ptr = ptr -> next;
        }
        return start;
    }
}
```

P.E

22/1/24

classmate
Date
Page

Lab-6 - Linked list implementation

1) Stack implementation using linked list

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
struct stack
{
int data;
struct stack* next;
};
struct stack *top = NULL;
struct stack *push(struct stack*, int);
struct stack *display(struct stack*);
struct stack *pop(struct stack*);

struct *stack *push(struct stack *top, int val)
{
struct stack *ptr;
ptr = (struct stack* push) malloc (sizeof (struct stack));
ptr -> data = val;
if (top == NULL)
{
ptr -> next = NULL;
top = ptr
printf ("The value %d has been iserted", val);
}
```

SLL / stack / queue

Entry NULL

29/11/24

```
else
{
    ptr -> next = top;
    top = ptr;
    printf("The value %d is inserted", val);
}
return top;
}

struct stack *pop (struct stack *top)
{
    struct stack *ptr;
    ptr = top;
    if (top == NULL)
        printf("\n stack underflow");
    else
        {
            top = top -> next
            printf("The value deleted is : %d", ptr->
            free(ptr);
        }
    return top;
}

struct stack *display (struct stack *top)
{
    struct stack *ptr;
    ptr = top;
```

```
if (* top == NULL)
    printf ("\n stack is empty");
else
{
    while (ptr != null)
      {
        printf ("\n stack elements are: ");
        printf ("\n %d", ptr -> data);
        ptr = ptr -> next
      }
}
return top;
}

Void main ()
{
 while (1)
  {
    printf ("\n SELECT THE OPERATION");
    printf ("\n 1. PUSH");
    printf ("\n 2. POP");
    printf ("\n 3. DISPLAY");
    printf ("\n 4. EXIT");
    printf ("\n enter your option:");
    scanf ("%d", &option);
    switch (option)
    {
        case 1 : printf ("\n Enter the numbe to be
                  pushed on stack");
```

```
            scanf ("%d", &val);
            top = push(top, val);
            break
        case 2: top = pop(top);
                break;
        case 3: top = display(top);
                break;
        case 4: exit(0);
        default: printf("Invalid Input");
    }
  }
}
```

2) Queue implementation using single linked list

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

struct node
 {
  int data;
  struct node *next;
 };
struct queue
 {
  struct node *front
  struct node *rear
 };
```

```
struct queue *createQueue()
{
    struct queue *q = (struct queue *) malloc (struct queue);
    q -> front = q -> rear = NULL;
    return q;
}

struct queue *q;
struct queue *insert (struct queue *, int);
struct queue *delete (struct queue*);
struct queue * display (struct queue

struct queue * insert (struct queue *q , int)
{
  struct node *ptr
  ptr = (struct node *) malloc (sizeof (struct node));
  ptr -> data = val;
  if (q -> front == NULL)
  {
    q -> front = ptr
    q -> rear = ptr
    q -> front -> next = q -> rear -> next = NULL;
  }
  else
  {
      q -> rear -> next = ptr ;
      q -> rear = ptr;
      q -> rear -> next = NULL;
  }
```

```
struct queue *createQueue()
{
    struct queue *q = (struct queue *) malloc (struct queue);
    q -> front = q -> rear = NULL;
    return q;
}


struct queue *q;
struct queue *insert (struct queue *, int);
struct queue *delete (struct queue*);
struct queue * display (struct queue

struct queue *insert (struct queue *q , int)
{
  struct node *ptr
  ptr = (struct node *) malloc (size of (struct node));
  ptr -> data = val;
  if (q -> front == NULL)
  {
     q -> front = ptr
     q -> rear = ptr
     q -> front -> next = q -> rear -> next = NULL;
  }
  else
   {
       q -> rear -> next = ptr ;
       q -> rear = ptr;
       q -> rear -> next = NULL;
    }
    return q;
}
```

```
struct queue * delete (struct queue * q)
{
 struct node * ptr;
 ptr = q -> front
  if (q -> front == NULL)
     printf ("\n underflow ");
  else
    {
      q -> front = q -> front -> next
      printf ("\nThe value being deleted is: %d ",ptr->data);
       free (ptr);
    }
   return q;
}

struct queue * display (struct queue * q)
 {
    struct node *ptr;
    ptr = q -> front;
    if (ptr == NULL)
         printf ("\nQueue is empty ");
    else
    {
      q -> front. q -> front -> next
      printf ("\n The value being deleted is: %d",ptr->data);
      free (ptr);
    }
return q;
}
```

```
void main()
{
  int val, option;
  q = creatQueue(q);
  while (1)
  {
    printf("\n SELECT THE OPTION");
    printf("\n 1.Insert");
    printf("\n 2. Delete");
    printf("\n 3.Display");
    printf("\n 4. Exit");
    printf("\n enter the option:");
    scanf("%d", &option);
    switch(option)
    {
    case 1: printf("\nEnter the number to be
            inserted");
            scanf("%d", &val);
            q = insert(q, val);
            printf("\n value inserted");
            break;
    case 2: q = delete(q)
            break;
    case 3: q = display(q);
            break;
    case 4: exit(0);
    default: printf("\nInvalid Input");
    }
  }
}
```

Lab - 07

Q) WAP to implement doubly linked list with operations like:

(a) create a doubly linked list
(b) insert a newnode to the left of a node
(c) Delete node on specific value.

```
#include <stdio.h>

struct node {
   int data;
   struct node * next;
   struct node * prev;
  };
struct node *head=0, *newnode, *temp;
void create()
{
int i,n;
printf("Enter the no. of elements: \n");
scanf("%d", &n);
for(i=0; i<n; i++)
{
newnode = (struct node *) malloc (sizeof(struct
printf("Enter the %d element: \n", i+1);
scanf("%d", &newnode->data);
newnode->prev=0
newnode->next=0
if(head==0)
    temp=head=newnode
```

```
else {
    temp -> next = newnode;
    newnode -> prev = temp
    temp = newnode;
  }
 }
}

void display()
 {
  temp = head;
  while(temp != 0)
  {
    printf("%d \n", temp->data);
    temp = temp ->next;
  }
}

void insert_left()
 {
  int node, i=1;
  printf("Enter the node \n");
  scanf("%d", &node);
  temp = head;
  if (node<1)
        printf("invalid position \n");
```

```
else if (node == 1)
 {
   new node = (struct node *) malloc (sizeof (struct node
   printf (" enter data \n ");
    scanf (" %d, & new node → data);
   new node → prev = 0;
    head → prev = new node
     new node → next = head
      head = new node.;
  }
else
  {
   new node - (struct node *) malloc (sizeof (struct
   printf (" Enter data \n ");
   scanf (" %d ", & new node → data);
   while (i < node - 1)
      {
        temp = temp → next;
         i++;
      }
      new node → prev = temp;
       new node → next = temp → next;
        temp → next = new node;
        new node → next → prev = new node;
  }
}
```

```
void delete_pos ()
{
  int pos, i=1;
  temp = head;
  printf ("Enter position \n");
  scanf ("%d", &pos);
  while (i<pos)
    {
      temp = temp->next;
      i++;
    }
  temp->prev->next = temp->next;
  temp->next->prev = temp->prev;
  free (temp);
}

void main ()
{
  int choice, num;
  printf ("operations: \n 1. create \n 2 display
          \n 3. insert before \n 4. delete at \n 5. exit");
  while (1)
  {
  printf ("enter operation \n");
  scanf ("%d", &choice);
  if (choice == 5)
    {
      printf ("completed \n");
      break;
    }
```

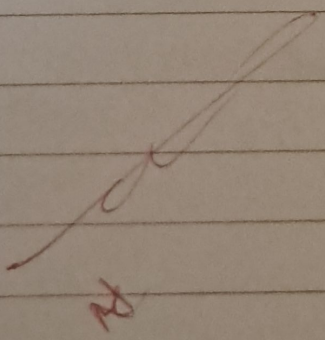
```
else
  {
    switch (choice)
    {
       case 1: create();
               break;
       case 2: display();
               break();
       case 3: insert_left();
               break;
       case 4: delete_pos();
               break;
       default: printf("Invalid input \n");
    }
}
}
}
}
```

Output

operations:
1. create
2. display
3. insert before
4. delete at
enter operation
1

enter the no. of elements
4
enter the 1 element:
3
Enter the 2 element.
4
enter the 3 element:
2
Enter the 4 element:
5
enter operation
2
3
4
2
5
enter operation
3
enter the node:
2
enter data
7
enter operation
2
3
7
4
2
5

enter operation
4
enter position
3
enter operation
2
3
7
2
5
enter operation
5
completed.



classmate
Date
Page

Lab -8 - Binary Search Tree

1) Write a program to:
(a) Construct Binary Search Tree
(b) Traverse the tree using inorder, postorder, preorder
(c) Display the elements in the tree

Program:-

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

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

struct Node *createnode(int value) {
    struct Node *Newnode = (struct Node*)malloc(sizeof(
    Newnode -> data = value
    Newnode -> left = Newnode -> right = NULL;
    return Newnode
}

struct Node *insert(struct Node *root, int value) {
    if (root == Null) {
        return createnode(value);
    }
    if (value < root -> data) {
        root -> left = insert(root -> right, value)
    }
```

```
    else if (value > root->data){
        root->right = insert(root->right, value);
    }
    return root;
}

void inorder(struct Node *root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

void postorder(struct Node *root) {
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}

void preorder(struct Node *root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}
```

```
void display (struct Node *root) {
    printf ("Elements in the tree: ");
    inorder (root);
    printf ("\n");
}

int main () {
    struct Node * root = NULL;
    int choice, value;
    do {
        printf ("\n Binary Search Tree operations");
        printf ("\n 1. Insert ");
        printf ("\n 2. Inorder ");
        printf ("\n 3. Postorder ");
        printf ("\n 4. preorder ");
        printf ("\n 5. Display ");
        printf ("\n 6. Exit");
        printf ("\n Enter your choice: ");
        printf (
        scanf ("%d", &choice);

        switch (choice) {
            case 1:
                printf ("Enter value to insert: ");
                scanf ("%d", &value);
                root = insert (root, value);
                break;
```

```
        case 2:
            printf("Inorder Traversal: ");
            inorder(root);
            printf("\n");
            break;
        case 3:
            printf("Postorder Traversal: ");
            postorder(root);
            printf("\n");
            break;
        case 4: printf("Preorder Traversal: ");
            preorder(root);
            printf("\n");
            break;
        case 5:
            display(root);
            break;
        case 6:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid Input!");
    }
} while (choice != 6);
return 0;
}
```

2) leet code Odd even

Program:-

```
struct ListNode * oddevenList (struct ListNode* head) {
    if (head == NULL) {
        return head;
    }
    struct ListNode * head1;
    struct ListNode * head2;
    struct ListNode * tail1;
    struct ListNode * tail2;
    head1 = head2 = tail1 = tail2 = NULL;
    for (int i = 0; head != NULL) {
        struct ListNode * n = (struct ListNode*) malloc (
                                sizeof(struct
                                ListNode));
        n -> val = head -> val;
        n -> next = NULL;
        if (i%2 == 0) {
            if (head1 == NULL) {
                head1 = tail1 = n;
            }
            else {
                tail1 -> next = n;
                tail1 = n;
            }
        else {
            if (head2 == NULL) {
                head2 = tail2 = n
            }
```

```
else {
    tail2 -> next = n;
    tail2 = n;
  }
}
head = head -> next
}
tail1 -> next = head2;
retur head1;
}
```

3) Delete middle node - LEET CODE

Program:-

```
struct ListNode * deleteMiddle(struct ListNode * h
    if (head == Null) return Null;
    struct ListNode * prev = (struct ListNode *) malloc(
                                    sizeof(struct ListNode
    prev -> val = 0;
    preve -> next = head
    struct ListNode * ptr1 = head
    struct ListNode * ptr = prev;
    while (ptr1 != NULL && ptr1 -> next != NULL
        ptr = ptr -> next;
        ptr2 = ptr1 -> next -> next
    }
    struct ListNode * temp = ptr -> next;
    ptr -> next = ptr -> next -> next;
    free(temp);
    struct ListNode * newHead = prev -> next;
```

```
free (prev);
return newHead;
```

Lab-9

1) BFS (Breadth first Search)

Program:-

```
#include<stdio.h>
#include<stdlib.h>
#define MAX_NODES 100

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

struct graph {
    int numNodes;
    struct node * adjlists[MAX_NODES];
    int visited[MAX_NODES];
};

struct node * createNode(int data) {
    struct node* newnode = (struct node*) malloc
    newnode -> data = data;                (sizeof(struct
    newnode -> next = NULL;
    return newnode;
}
```

```
struct graph* creategraph(int n) {
    struct graph* graph = (struct graph*) malloc
                          (sizeof(struct graph));
    graph->numnodes = n;
    for (int i=0; i<n; i++)
    {
        graph->adjlists[i] = NULL
        graph->visited[i] = 0
    }
    return graph;
}

void addedge(struct graph* graph, int src, int dest) {
    struct node* newnode = createNode(dest);
    newnode->next = graph->adjlists[src]
    graph->adjlists[src] = newnode;
    newnode = createNode(src);
    newnode->next = graph->adjlists[dest
    graph->adjlists[dest] = newnode;
}

void BFS(struct graph* graph, int startnode) {
    int queue[MAX_NODES];
    int front = 0, rear = 0;
    graph->visited[startnode] = 1
    queue[rear++] = startnode
    while (front<rear) {
        int current = queue[front++];
        printf("%d ", current);
```

```
struct node * temp = graph -> adjLists [curr
    while (temp) {
        int adjnode = temp -> data;
        if ( !graph -> visited [adjnode] ) {
            graph -> visited [adjnode] = 1;
            queue [rear++] = adjnode;
        }
        temp = temp -> next;
    }
  }
}

int main() {
    int numnodes;
    printf (" Enter the number of nodes: ");
    scanf ("%d", &numnodes);
    struct graph * graph = createGraph (numnod
    int numedges;
    printf (" Enter the number of edges: ");
    scanf ("%d", & numedges);
    for (int i=0 ; i< numedges ; i++) {
        int src, dest;
        printf (" Enter edge %d source
                                    , %d
        addedge (graph, src, dest);
    }
```

```
int startnode;
printf("Enter the starting node for BFS traversal:");
scanf("%d", &startnode);
printf("BFS traversal starting from node %d: ", startnode);
BFS(graph, startnode);
return 0;
}
```

Output :-

```
enter the number of node: 4
enter the number of edges: 5
enter edge 1 (source destination): 1
2
enter edge 2 (source destination): 2
4
enter edge 3 (source destination): 4
3
enter edge 4 (source destination): 3
1
enter edge 5 (source destination): 1
4
enter the starting node for BFS traversal: 2
BFS traversal starting from node 2: 2 1 3
```

2)

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_NODES 100

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

struct graph {
    int numnodes;
    struct node * adjlists [MAX_N
    int visited [MAX_NODES];
};

struct node * createNode (int data) {
    struct node * newnode = (struct node *) m
    newnode -> data = data;
    newnode -> next = NULL;
    return newnode;
};

struct graph * createGraph (int n) {
    struct graph * graph = (struct graph
                                (size o
    graph -> numnodes = n
    for (int i = 0; i < n; i++) {
        graph -> adjlists [i] = NULL;
        graph -> visited [i] = 0;
    }
return graph;
```

```
void addedge (struct graph* graph, int src, int dest) {
    struct node* newnode = createNode (dest);
    newnode->next = graph->adjlists[src];
    graph->adjlists[src] = newnode;
    newnode = createNode (src);
    newnode->next = graph->adjlists[dest];
    graph->adjlists[dest] = newnode;
}

void DFS (struct graph* graph, int startnode) {
    graph->visited[startnode] = 1
    printf ("%d ", startnode);
    struct node* temp = graph->adjlists[startnode];
    while (temp) {
        int adjnode = temp->data;
        if (!graph->visited[adjnode]) {
            DFS (graph, adjnode);
        }
        temp = temp->next;
    }
}

int main() {
    int numnodes;
    printf ("Enter the number of nodes: ");
    scanf ("%d", &numnodes);
    struct graph* graph = createGraph (numnodes);
    int numedges;
    printf ("Enter the number of edges: ");
    scanf ("%d", &numedges);
```

26/2/24

```
for (int i=0; i<numEdges; i++) {
    int src, dest;
    printf("Enter edge %d: ", i+1);
    scanf("%d %d", &src, &dest);
    addEdge(graph, src, dest);
}
int startNode;
printf("Enter the starting node for DFS traversal: ");
scanf("%d", &startNode);
printf("DFS traversal from node %d: ", startNode);
DFS(graph, startNode);
return 0;
}
```

output :-

```
enter the number of nodes : 4
enter the number of edges : 5
enter edge 1 : 1
2
Enter edge 2 : 2
4
enter edge 3 : 4
3
enter edge 4 : 3
1
enter edge 5 : 2
3
Enter starting node for DFS traversal : 2
DFS traversal from 2 : 2 3 1
```