DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING)

II B.Tech I Semester

Subject Name: DATA STRUCTURES LAB
Subject Code: C0512
Regulations: MR-22

Lab Manual

Academic Year: 2024-25

MALLA REDDY ENGINEERING COLLEGE (AUTONOMOUS)
MAIN CAMPUS
(An UGC Autonomous Institution, Approved by AICTE and Affiliated to JNTUH, Hyderabad, Accredited by NAAC with 'A++' Grade (III Cycle))
NBA Accredited Programmes – UG (CE, EEE, ME, ECE, & CSE), PG (CE-SE, EEE, EPS, ME-TE)
Maisammaguda(H), Gundlapochampally Village, Medchal Mandal,
Medchal-Malkajgiri District, Telangana State – 500100

MALLA REDDY ENGINEERING COLLEGE (AUTONOMOUS)

MR22 – ACADEMIC REGULATIONS (CBCS)

for B.Tech. (REGULAR) DEGREE PROGRAMME

Applicable for the students of B.Tech. (Regular) programme admitted from the Academic Year 2022-23 onwards

The B.Tech. Degree of Jawaharlal Nehru Technological University Hyderabad, Hyderabad shall be conferred on candidates who are admitted to the programme and who fulfill all the requirements for the award of the Degree.

VISION OF THE INSTITUTE

To be a premier center of professional education and research, offering quality programs in a socio-economic and ethical ambience.

MISSION OF THE INSTITUTE

- To impart knowledge of advanced technologies using state-of-the-art infrastructural facilities.
- To inculcate innovation and best practices in education, training and research.
- To meet changing socio-economic needs in an ethical ambience.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING – ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

DEPARTMENT VISION

To attain global standards in Computer Science and Engineering education, training and research to meet the growing needs of the industry with socio-economic and ethical considerations.

DEPARTMENT MISSION

- To impart quality education and research to undergraduate and postgraduate students in Computer Science and Engineering.
- To encourage innovation and best practices in Computer Science and Engineering utilizing state-of-the-art facilities.
- To develop entrepreneurial spirit and knowledge of emerging technologies based on ethical values and social relevance.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO1: Graduates will demonstrate technical skills, competency in AI & ML and exhibit team management capability with proper communication in a job environment

PEO2: Graduates will function in their profession with social awareness and responsibility

PEO3: Graduates will interact with their peers in other disciplines in industry and society and contribute to the economic growth of the country

PEO4: Graduates will be successful in pursuing higher studies in engineering or management

PROGRAMME OUTCOMES (POs)

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, review research literature and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

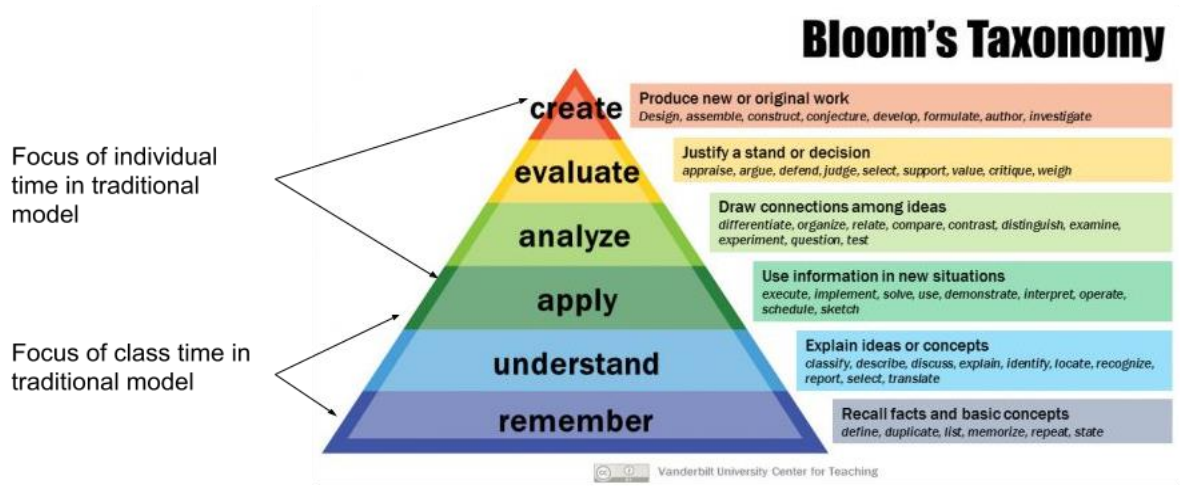
PROGRAMME SPECIFIC OUTCOMES (PSOs)

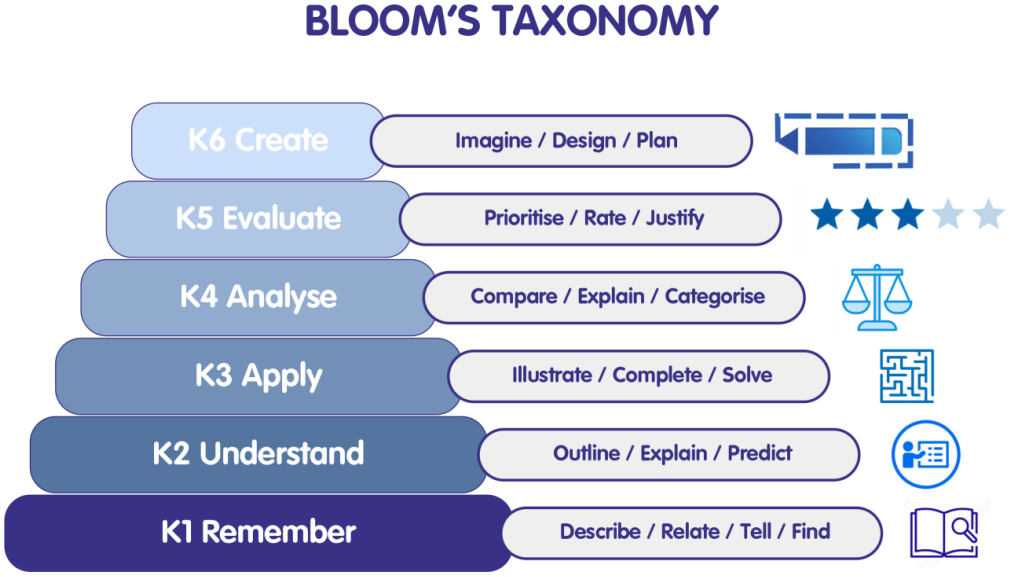
PSO1: Design and develop intelligent automated systems applying mathematical, analytical, programming and operational skills to solve real world problems

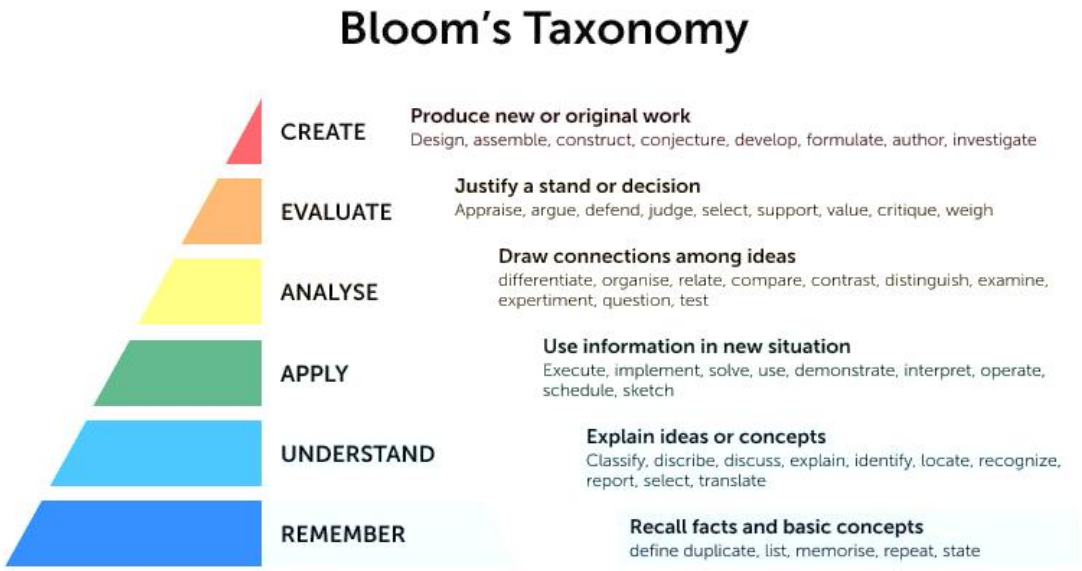
PSO2: Apply machine learning techniques, software tools to conduct experiments, interpret data and to solve complex problems

PSO3: Implement engineering solutions for the benefit of society by the use of AI and ML

BLOOM'S TAXONOMY (BT) TRIANGLE &
BLOOM'S ACTION VERBS







BLOOM'S ACTION VERBS

REVISED Bloom's Taxonomy Action Verbs

Definitions	I. Remembering	II. Understanding	III. Applying	IV. Analyzing	V. Evaluating	VI. Creating
Bloom's Definition	Exhibit memory of previously learned material by recalling facts, terms, basic concepts, and answers.	Demonstrate understanding of facts and ideas by organizing, comparing, translating, interpreting, giving descriptions, and stating main ideas.	Solve problems to new situations by applying acquired knowledge, facts, techniques and rules in a different way.	Examine and break information into parts by identifying motives or causes. Make inferences and find evidence to support generalizations.	Present and defend opinions by making judgments about information, validity of ideas, or quality of work based on a set of criteria.	Compile information together in a different way by combining elements in a new pattern or proposing alternative solutions.
Verbs	• Choose • Define • Find • How • Label • List • Match • Name • Omit • Recall • Relate • Select • Show • Spell • Tell • What • When • Where • Which • Who • Why	• Classify • Compare • Contrast • Demonstrate • Explain • Extend • Illustrate • Infer • Interpret • Outline • Relate • Rephrase • Show • Summarize • Translate	• Apply • Build • Choose • Construct • Develop • Experiment with • Identify • Interview • Make use of • Model • Organize • Plan • Select • Solve • Utilize	• Analyze • Assume • Categorize • Classify • Compare • Conclusion • Contrast • Discover • Dissect • Distinguish • Divide • Examine • Function • Inference • Inspect • List • Motive • Relationships • Simplify • Survey • Take part in • Test for • Theme	• Agree • Appraise • Assess • Award • Choose • Compare • Conclude • Criteria • Criticize • Decide • Deduct • Defend • Determine • Disprove • Estimate • Evaluate • Explain • Importance • Influence • Interpret • Judge • Justify • Mark • Measure • Opinion • Perceive • Prioritize • Prove • Rate • Recommend • Rule on • Select • Support • Value	• Adapt • Build • Change • Choose • Combine • Compile • Compose • Construct • Create • Delete • Design • Develop • Discuss • Elaborate • Estimate • Formulate • Happen • Imagine • Improve • Invent • Make up • Maximize • Minimize • Modify • Original • Originate • Plan • Predict • Propose • Solution • Solve • Suppose • Test • Theory

Anderson, L. W., & Krathwohl, D. R. (2001). A taxonomy for learning, teaching, and assessing, Abridged Edition. Boston, MA: Allyn and Bacon.

2022-23 Onwards (MR-22)	MALLA REDDY ENGINEERING COLLEGE (AUTONOMOUS)	B.Tech. VI Semester		
Code: C0512	DATA STRUCTURES LAB	L	T	P
Credits: 1.5		-	-	3

Course Objectives:

1. To learn linear data structures such as linked list, stack and queues with its operations
2. Ability to learn programs on binary search tree and graph traversal strategies.
3. To understand the pattern matching and hashing techniques.

Software Requirements: Turbo C / C++

List of Programs:

1. Write a program that uses functions to perform the following operations on singly linkedlist.:
   i) Creation ii) Insertion iii) Deletion iv) Traversal
2. Write a program that uses functions to perform the following operations on doubly linkedlist.:
   i) Creation ii) Insertion iii) Deletion iv) Traversal
3. Write a program that uses functions to perform the following operations on circular linkedlist.:
   i) Creation ii) Insertion iii) Deletion iv) Traversal
4. Write a program that implement stack (its operations) using
   i) Arrays ii) Pointers
5. Write a program that implement Queue (its operations) using
   i) Arrays ii) Pointers
6. Write a program that implements the following sorting methods to sort a given list of integers in ascending order
   i) Quick sort ii) Heap sort iii) Merge sort
7. Write a program to implement the tree traversal methods(Recursive and Non Recursive).
8. Write a program to implement
   i) Binary Search tree ii) B Trees iii) B+ Trees iv) AVL trees
   v) Red - Black trees
9. Write a program to implement the graph traversal methods.
10. Implement a Pattern matching algorithms using Boyer- Moore, Knuth-Morris-Pratt

Text Books

1. Fundamentals of Data Structures in C, 2nd Edition, E. Horowitz, S. Sahni and Susan Anderson Freed, Universities Press.
2. Data Structures using C – A. S. Tanenbaum, Y. Langsam, and M. J. Augenstein, PHI/Pearson Education.

References

1. Data Structures: A Pseudocode Approach with C, 2nd Edition, R. F. Gilberg and B. A. Forouzan, Cengage Learning.

Course Outcomes:

At the end of the course, students will be able to

- Ability to develop C programs for computing and real-life applications using basic elements like control statements, arrays, functions, pointers and strings, and data structures like stacks, queues and linked lists.

CO- PO, PSO Mapping (3/2/1 indicates strength of correlation) 3-Strong, 2-Medium, 1-Weak															
COs	**Programme Outcomes (POs)**												**PSOs**		
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**	**PSO3**
CO1	2	3	2										3	2	
CO2	2	2	3										2	3	
CO3		2	3										2	3	1

1. Write a program that uses functions to perform the following operations on singly linked list.:

i) Creation ii) Insertion iii) Deletion iv) Traversal

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node *next;
}*start=NULL,*q,*t;

int main()
{
   int ch;
   void insert_beg();
   void insert_end();
   int insert_pos();
   void display();
   void delete_beg();
   void delete_end();
   int delete_pos();

   while(1)
   {
      printf("\n\n---- Singly Linked List(SLL) Menu ----");
      printf("\n1.Insert at beginning\n2.Insert at end\n3.Insert at specified position\n4.Delete from beginning\n5.Delete from end\n6.Delete from specified position\n7.Display\n8.Exit");
      printf("\nEnter your choice(1-8):");
```

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

            switch(ch)
               {
                  case 1: insert_beg();
                        break;
                  case 2: insert_end();
                        break;
                  case 3: insert_pos();
                        break;
                     case 4: delete_beg();
                        break;
                  case 5: delete_end();
                        break;
                  case 6: delete_pos();
                        break;
                     case 7: display();
                     break;
                  case 8: exit(0);
                              break;
                  default: printf("Wrong Choice!!");
               }

               }
   return 0;
}

void insert_beg()
{
```

```
    int num;
    t=(struct node*)malloc(sizeof(struct node));
    printf("Enter data:");
    scanf("%d",&num);
    t->data=num;
     if(start==NULL)        //If list is empty
    {
       t->next=NULL;
       start=t;
    }
    else
    {
       t->next=start;
       start=t;
    }
}
 void insert_end()
{
    int num;
    t=(struct node*)malloc(sizeof(struct node));
    printf("Enter data:");
    scanf("%d",&num);
    t->data=num;
    t->next=NULL;

    if(start==NULL)        //If list is empty
    {
       start=t;
    }
```

```
    else
    {
        q=start;
        while(q->next!=NULL)
        q=q->next;
        q->next=t;
    }
}

int insert_pos()
{
    int pos,i,num;
    if(start==NULL)
    {
        printf("List is empty!!");
        return 0;
    }

    t=(struct node*)malloc(sizeof(struct node));
    printf("Enter data:");
    scanf("%d",&num);
    printf("Enter position to insert:");
    scanf("%d",&pos);
    t->data=num;

    q=start;
    for(i=1;i<pos-1;i++)
    {
        if(q->next==NULL)
```

```
        {
            printf("There are less elements!!");
            return 0;
        }

        q=q->next;
    }

    t->next=q->next;
    q->next=t;
    return 0;
}

void display()
{
    if(start==NULL)
    {
        printf("List is empty!!");
    }
    else
    {
        q=start;
        printf("The linked list is:\n");
        while(q!=NULL)
        {
            printf("%d->",q->data);
            q=q->next;
        }
    }
```

```
}

void delete_beg()
{
   if(start==NULL)
   {
      printf("The list is empty!!");
   }
   else
   {
      q=start;
      start=start->next;
      printf("Deleted element is %d",q->data);
      free(q);
   }
}

void delete_end()
{
   if(start==NULL)
   {
      printf("The list is empty!!");
   }
   else
   {
      q=start;
      while(q->next->next!=NULL)
      q=q->next;
```

```
        t=q->next;
        q->next=NULL;
        printf("Deleted element is %d",t->data);
        free(t);
    }
}

int delete_pos()
{
    int pos,i;

    if(start==NULL)
    {
        printf("List is empty!!");
        return 0;
    }

    printf("Enter position to delete:");
    scanf("%d",&pos);

    q=start;
    for(i=1;i<pos-1;i++)
    {
        if(q->next==NULL)
        {
            printf("There are less elements!!");
            return 0;
        }
        q=q->next;
```

```
    }

    t=q->next;
    q->next=t->next;
    printf("Deleted element is %d",t->data);
    free(t);

    return 0;
}
```

OUTPUT

```
---- Singly Linked List(SLL) Menu ----
1.Insert at beginning
2.Insert at end
3.Insert at specified position
4.Delete from beginning
5.Delete from end
6.Delete from specified position
7.Display
8.ExitEnter your choice(1-8):8
root@cse:~# gcc singlelinked.c
root@cse:~# ./a.out
```

2. Write a program that uses functions to perform the following operations on doubly linked list.:

i) Creation ii) Insertion iii) Deletion iv) Traversal

```
#include<stdio.h>
#include<stdlib.h>
typedef struct node1
{
   int info;
   struct node1* next;
   struct node1* prev;
}node;
node *head=NULL;
node *tail=NULL;
node* create(int i)
{
   node* tmp=(node*)malloc(sizeof(node));
   tmp->info=i;
   tmp->next=NULL;
   tmp->prev=NULL;
   printf("\nNode Cretaed...!!!!");
   return tmp;
}
void addtohead(node *tmp)
{
   node *t;
   t=head;
   t->prev=tmp;
   tmp->next=t;
```

```
    head=tmp;

    printf("\nNode inserted at the start...!!");
}
void addtoend(node *tmp)
{

    if(head==tail)
    {
        head->next=tmp;
        tmp->prev=head;
        tail=tmp;
    }
    else
            {
        tail->next=tmp;
        tmp->prev=tail;
        tail=tmp;

    }
    printf("\nNode inserted at the end...!!!");
}
void atpos(node *newnode)
   {
    node *nextp,*curr,*tmp;
    int i,pos;
    printf("\nEnter the position at which you want to insert the node ?? ");
    scanf("%d",&pos);
        if(pos==1)
```

```
      {
       addtohead(newnode);
       }

   else
    {
      curr=head;
      nextp=head->next;
      for(i=1;i<pos-1;i++)
       {
         nextp=nextp->next;
         curr=curr->next;
       }
       tmp=nextp;
       curr->next=newnode;
       newnode->prev=curr;
       newnode->next=tmp;
       tmp->prev=newnode;
       printf("\nNode inserted at position %d !!!",pos);
   }
}
void insert()
{
   int data,i,p,ch;
   do{
      printf("\nEnter the element you want to insert : ");
      scanf("%d",&data);
      node* tmp=create(data);
      if(head==NULL){
```

```
            head=tail=tmp;
           printf("\nList was Empty....Item inserted at first position !!!");
        }
       else{
            printf("\nAt which position you want to insert the item ??? ");
            printf("\n1.Begining");
            printf("\n2.Ending");
            printf("\n3.At specific position");
            printf("\nEnter your choice : ");
            scanf("%d",&p);
            switch(p){
               case 1: addtohead(tmp);
                     break;
               case 2: addtoend(tmp);
                     break;
               case 3: atpos(tmp);
                     break;
               default: printf("\nWrong choice ...Enter again...!!!");
            }
       }
       printf("\nDo you want to insert another node (1/0) ??");
       scanf("%d",&ch);
   }while(ch==1);
}

void deletefromhead()
{
   node *tmp=head;
   head=head->next;
```

```
    head->prev=NULL;
    printf("\nNode deleted from head is : %d",tmp->info);
    free(tmp);
}

void deletefromend()
{
    node *tmp=tail;
    tail=tail->prev;
    printf("\nNode deleted from end is : %d",tmp->info);
    free(tmp);
    tail->next=NULL;
}

void deletepos()
{
    int pos,i,c;
    node *pre,*curr;
    pre=head;
    curr=head->next;
    printf("\nEnter the position of node you want to delete : ");
    scanf("%d",&pos);
       if(pos==1)
       deletefromhead();
       else
          {
       for(i=1;i<pos-1;i++)
          {
          curr=curr->next;
```

```
            pre=pre->next;
        }
        printf("\nNode deleted from %d is : %d",pos,curr->info);
        pre->next=curr->next;
        curr->next->prev=pre;
        free(curr);
    }
}
void deleten(){
    int p,ch;
    do{
        if(head==NULL){
            printf("\nList is empty...!!!!");
        }
        else if(head==tail){
            printf("\nOnly one element in the list.. delelted node is : %d ",head->info);
            free(head);
            head=NULL;
            tail==NULL;
        }
        else{
            system("cls");
            printf("\nFrom which position you want to delete the node ?? ");
            printf("\n1.Begining");
            printf("\n2.Ending");
            printf("\n3.At specific position");
            printf("\nEnter your choice : ");
            scanf("%d",&p);
            switch(p){
```

```
            case 1: deletefromhead();
                    break;
            case 2: deletefromend();
                    break;
            case 3: deletepos();
                    break;
            default: printf("\nWrong choice ...Enter again...!!!");
          }
       }
       printf("\nDo you want to delete another node (1/0)??");
       scanf("%d",&ch);
   }while(ch==1);
}
void display()
{
    node *tmp=head;
   if(head==NULL)
      printf("\nList is empty..!!!");
   else{
     for(;tmp!=NULL;tmp=tmp->next){

         if(tmp->next==NULL)
            printf("%d",tmp->info);
         else
           printf("%d -> ",tmp->info);
      }
   }
}
```

```
int main(){
   int c,ch;
   do{

      printf("\n-------DOUBLY LINKED LIST-------");
      printf("\n1.INSERT INTO LIST");
      printf("\n2.DELETE FROM LIST");
      printf("\n3.DISPLAY THE LIST");
      printf("\n4.EXIT");
      printf("\nEnter your choice : ");
      scanf("%d",&c);
      switch(c){
         case 1: insert();
               break;
         case 2: deleten();
               break;
         case 3: display();
               break;
         case 4: exit(0);
      }
      printf("\nDo you want to perform any function again (1/0)???");
      scanf("%d",&ch);
   }while(ch==1);
   return 0;
}
```

OUTPUT

-------DOUBLY LINKED LIST-------

1.INSERT INTO LIST

2.DELETE FROM LIST

3.DISPLAY THE LIST

4.EXIT

Enter your choice : 1

3. Write a program that uses functions to perform the following operations on circular linked list.:

i) Creation ii) Insertion iii) Deletion iv) Traversal

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

typedef struct Node
{
        int info;
        struct Node *next;
}node;

node *front=NULL,*rear=NULL,*temp;

void create();
void del();
void display();

int main()
{
        int chc;
        do
        {
        printf("\nMenu\n\t 1 to create the element : ");
        printf("\n\t 2 to delete the element : ");
        printf("\n\t 3 to display : ");
        printf("\n\t 4 to exit from main : ");
        printf("\nEnter your choice : ");
```

```
        scanf("%d",&chc);

                switch(chc)
                {
                        case 1:
                        create();
                        break;

                        case 2:
                        del();
                        break;

                        case 3:
                        display();
                        break;

                        case 4:
                        return 1;

                        default:
                                printf("\nInvalid choice :");
                }
        }while(1);

        return 0;
}
void create()
{
        node *newnode;
```

```
        newnode=(node*)malloc(sizeof(node));
        printf("\nEnter the node value : ");
        scanf("%d",&newnode->info);
        newnode->next=NULL;
        if(rear==NULL)
        front=rear=newnode;
        else
        {
                rear->next=newnode;
                rear=newnode;
        }

        rear->next=front;
}
void del()
{
        temp=front;
        if(front==NULL)
                printf("\nUnderflow :");
        else
        {
                if(front==rear)
                {
                        printf("\n%d",front->info);
                        front=rear=NULL;
                }
                else
                {
                        printf("\n%d",front->info);
```

```
                front=front->next;
                rear->next=front;
            }

        temp->next=NULL;
        free(temp);
        }
}
void display()
{
        temp=front;
        if(front==NULL)
            printf("\nEmpty");
        else
        {
            printf("\n");
            for(;temp!=rear;temp=temp->next)
                printf("\n%d \t",temp->info);
                printf("\n%d \t",temp->info);
        }
}
```

4. Write a program that implement stack (its operations) using

i) Arrays ii) Pointers

STACK USING ARRAYS

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

#define MAX 5        //Maximum number of elements that can be stored

int top=-1,stack[MAX];
void push();
void pop();
void display();
void topelement();

void main()
{
        int ch;

        while(1)        //infinite loop, will end when choice will be 4
        {
                printf("\n*** Stack opeartions ***");
                printf("\n\n1.Push\n2.Pop\n3.Display\n4.top element\n5.Exit");
                printf("\n\nEnter your choice(1-5):");
                scanf("%d",&ch);

                switch(ch)
                {
```

```
                        case 1: push();
                                break;
                        case 2: pop();
                                break;
                        case 3: display();
                                break;
case 4:topelement();
break;
                        case 5: exit(0);

                        default: printf("\nWrong Choice!!");
                }
        }
}

void push()
{
        int val;

        if(top==MAX-1)
        {
                printf("\nStack is full!!");
        }
        else
        {
                printf("\nEnter element to push:");
                scanf("%d",&val);
                top=top+1;
                stack[top]=val;
```

```
        }
}

void pop()
{
        if(top==-1)
        {
                printf("\nStack is empty!!");
        }
        else
        {
                printf("\nDeleted element is %d",stack[top]);
                top=top-1;
        }
}
void topelement()
{
if(top==-1)
        {
                printf("\nStack is empty!!");
        }
        else
        {
                printf("\ntopelement is %d",stack[top]);
}
}

void display()
{
```

```
        int i;

        if(top==-1)
        {
                printf("\nStack is empty!!");
        }
        else
        {
                printf("\nStack is...\n");
                for(i=top;i>=0;--i)
                        printf("%d\n",stack[i]);
        }
}
```

OUTPUT

*** Stack opeartions ***

1.Push

2.Pop

3.Display

4.top element

5.Exit

Enter your choice(1-5):4

STACK USING POINTERS

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 50
int size;
// Defining the stack structure
struct stack {
  int arr[MAX];
  int top;
};

// Initializing the stack(i.e., top=-1)
void init_stk(struct stack *st) {
  st->top = -1;
}

// Entering the elements into stack
void push(struct stack *st, int num) {
  if (st->top == size - 1) {
    printf("\nStack overflow(i.e., stack full).");
    return;
  }
  st->top++;
  st->arr[st->top] = num;
}

//Deleting an element from the stack.
int pop(struct stack *st) {
```

```
  int num;
  if (st->top == -1) {
    printf("\nStack underflow(i.e., stack empty).");
    return 0;
  }
  num = st->arr[st->top];
  st->top--;
  return num;
}

void display(struct stack *st)
{
 if (st->top == -1)
 {
    printf("\nStack underflow(i.e., stack empty).");
}
else
{
 int i;
  printf("\nThe current stack elements are:");
  for (i = st->top; i >= 0; i--)
  printf("\n%d", st->arr[i]);
}
}

int main()
{
  int element, opt, val;
  struct stack ptr;
```

```
init_stk(&ptr);
printf("\nEnter Stack Size :");
scanf("%d", &size);
while (1) {
  printf("\n\nSTACK OPERATIONS");
  printf("\n1.PUSH");
  printf("\n2.POP");
  printf("\n3.DISPLAY");
  printf("\n4.QUIT");
  printf("\n");
  printf("\nEnter your option : ");
  scanf("%d", &opt);
  switch (opt) {
  case 1:
    printf("\nEnter the element into stack:");
    scanf("%d", &val);
    push(&ptr, val);
    break;
  case 2:
    element = pop(&ptr);
    printf("\nThe element popped from stack is : %d", element);
    break;
  case 3:

    display(&ptr);
    break;
  case 4:
    exit(0);
  default:
```

```
        printf("\nEnter correct option!Try again.");
      }
   }
   return (0);
}
```

OUTPUT

```
STACK OPERATIONS
1.PUSH
2.POP
3.DISPLAY
4.QUIT

Enter your option : 3
```

5. Write a program that implement Queue (its operations) using

i) Arrays ii) Pointers

Queue (its operations) using Arrays

```
#include <stdio.h>
#include<stdlib.h>
#define MAX 50
void insert();
void delete();
void display();
int queue_array[MAX];
int rear = - 1;
int front = - 1;
main()
{
   int choice;
   while (1)
   {
      printf("QUEUE OPERATIONS");
      printf("\n 1.Insert element to queue \n2.Delete element from queue \n3.Display elements of queue \n4.Quit \n");
      printf("Enter your choice : ");
      scanf("%d", &choice);
       switch (choice)
       {
          case 1:
          insert();
          break;
          case 2:
```

```
            delete();
            break;
            case 3:
            display();
            break;
            case 4:
            exit(1);
            default:
            printf("Wrong choice \n");
        }
    }
}
void insert()
{
    int element;
    if (rear == MAX - 1)
    printf("Queue Overflow \n");
    else
    {
        if (front == - 1)
        /*If queue is initially empty */
        front = 0;
        printf("Inset the element in queue : ");
        scanf("%d", &element);
        rear = rear + 1;
        queue_array[rear] = element;
    }
} /* End of insert() */
void delete()
```

```
{
    if (front == - 1 || front > rear)
    {
        printf("Queue Underflow \n");
        return ;

    }
   else
   {
        printf("Element deleted from queue is : %d\n", queue_array[front]);
        front = front + 1;
   }
 } /* End of delete() */
void display()
{
   int i;
   if (front == - 1)
        printf("Queue is empty \n");
   else
   {
        printf("Queue is : \n");
        for (i = front; i <= rear; i++)
          printf("%d ", queue_array[i]);
        printf("\n");
   }
}
```

QUEUE OPERATIONS

1.Insert element to queue

2.Delete element from queue

3.Display elements of queue

4.Quit

Enter your choice : 3

QUEUE USING LINKED LIST

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

typedef struct Node

{
        int info;
        struct Node *next;
}node;

node *front=NULL,*rear=NULL,*temp;

void create();
void del();
void display();

int main()
{
        int chc;
        do
        {
        printf("\nMenu\n\t 1 to create the element : ");
        printf("\n\t 2 to delete the element : ");
        printf("\n\t 3 to display : ");
        printf("\n\t 4 to exit from main : ");
        printf("\nEnter your choice : ");
        scanf("%d",&chc);
```

```
		switch(chc)
		{
			case 1:
			create();
			break;

			case 2:
			del();
			break;

			case 3:
			display();
			break;

			case 4:
			return 1;

			default:
				printf("\nInvalid choice :");
		}
	}while(1);

	return 0;
}
void create()
{
	node *newnode;
	newnode=(node*)malloc(sizeof(node));
```

```
        printf("\nEnter the node value : ");
        scanf("%d",&newnode->info);
        newnode->next=NULL;
        if(rear==NULL)
        front=rear=newnode;
        else
        {
                rear->next=newnode;
                rear=newnode;
        }

        rear->next=front;
}
void del()
{
        temp=front;
        if(front==NULL)
                printf("\nUnderflow :");
        else
        {
                if(front==rear)
                {
                        printf("\n%d",front->info);
                        front=rear=NULL;
                }
                else
                {
                        printf("\n%d",front->info);
                        front=front->next;
```

```
                        rear->next=front;
                }

        temp->next=NULL;
        free(temp);
        }
}
void display()
{
        temp=front;
        if(front==NULL)
                printf("\nEmpty");
        else
        {
                printf("\n");
                for(;temp!=rear;temp=temp->next)
                        printf("\n%d \t",temp->info);
                        printf("\n%d \t",temp->info);
        }
}
```

QUEUE OPERATIONS

1.Insert element to queue

2.Delete element from queue

3.Display elements of queue

4.Quit

Enter your choice : 3

Queue (its operations) using pointers

```
#define true 1
#define false 0
#include<stdio.h>
#include<stdlib.h>
struct q_point
{
  int ele;
  struct q_point* n;
};

struct q_point *f_ptr = NULL;

int e_que(void);
void add_ele(int);
int rem_ele(void);
void show_ele();

/*main function*/
int main()
{
  int ele,choice,j;
  while(1)
  {
    printf("\n\n****IMPLEMENTATION OF QUEUE USING POINTERS****\n");
    printf("==============================================");
    printf("\n\t\t  MENU\n");
    printf("==============================================");
    printf("\n\t[1] To insert an element");
```

```
printf("\n\t[2] To remove an element");
printf("\n\t[3] To display all the elements");
printf("\n\t[4] Exit");
printf("\n\n\tEnter your choice:");
scanf("%d", &choice);
switch(choice)
{
 case 1:
      printf("\n\tElement to be inserted:");
      scanf("%d",&ele);
      add_ele(ele);
      break;
 case 2:
      if(!e_que())
      {
       j=rem_ele();
       printf("\n\t%d is removed from the queue",j);
      }
      else
      {
       printf("\n\tQueue is Empty.");
       }
      break;
 case 3:
      show_ele();
      break;
 case 4:
      exit(1);
      break;
```

```
    default:
        printf("\n\tInvalid choice.");
        break;
   }
  }
}

/* Function to check if the queue is empty*/
int e_que(void)
{
  if(f_ptr==NULL)
  return true;
  return false;
}
/* Function to add an element to the queue*/
void add_ele(int ele)
{
  struct q_point *queue = (struct q_point*)malloc(sizeof(struct q_point));
  queue->ele = ele;
  queue->n = NULL;
  if(f_ptr==NULL)
    f_ptr = queue;
  else
  {
    struct q_point* ptr;
    ptr = f_ptr;
    for(ptr=f_ptr ;ptr->n!=NULL; ptr=ptr->n);
      ptr->n = queue;
  }
```

```
}

/* Function to remove an element from the queue*/
int rem_ele()
{
  struct q_point* queue=NULL;
  if(e_que()==false)
  {
    int j = f_ptr->ele;
    queue=f_ptr;
    f_ptr = f_ptr->n;
    free (queue);
    return j;
  }
  else
  {
    printf("\n\tQueue is empty.");
    return -9999;
  }
}
/* Function to display the queue*/
void show_ele()
{
  struct q_point *ptr=NULL;
  ptr=f_ptr;
  if(e_que())
  {
    printf("\n\tQUEUE is Empty.");
    return;
```

```
  }
  else
  {
    printf("\n\tElements present in Queue are:\n\t");
    while(ptr!=NULL)
    {
      printf("%d\t",ptr->ele);
      ptr=ptr->n;
    }
  }
}
```

OUTPUT

```
****IMPLEMENTATION OF QUEUE USING POINTERS****
==============================================
                    MENU
==============================================
          [1] To insert an element
          [2] To remove an element
          [3] To display all the elements
          [4] Exit

          Enter your choice:3
```

6. Write a program that implements the following sorting methods to sort a given list of integers

in ascending order

i) Bubble sort ii) Selection sort iii) Insertion sort

(a)BUBBLE SORT

```
#include<stdio.h>
int  main()
{
int a[100], n,i,j, t;
printf("enter no of elements\n");
scanf("%d",&n);
printf("enter elements one by one\n");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
for(i=0;i<n;i++)
 {
   for(j=i;j<n;j++)
    {
      if(a[i]>a[j])
      {
         t=a[i];
         a[i]=a[j];
          a[j]=t;
       }
    }
 }
```

```
printf("the sorted elements in ascending order are\n");
for(i=0;i<n;i++)
printf("%d\n",a[i]);
return 0;
}
```

OUTPUT

```
enter no of elements
10
enter elements one by one
25
12
1
68
53
45
17
34
62
78
the sorted elements in ascending order are
1
12
17
25
34
45
53
62
```

68

78

b) SELECTION SORT

```
#include<stdio.h>
 int main()
{
int s,i,j,temp,a[20];
 printf("Enter no of elements: ");
scanf("%d",&s);
printf("Enter %d elements: ",s);
for(i=0;i<s;i++)
scanf("%d",&a[i]);
for(i=0;i<s;i++)
{
for(j=i+1;j<s;j++)
{
if(a[i]>a[j])
{
temp=a[i];
a[i]=a[j];
a[j]=temp;
}
}
}
printf(" the sorted elements in ascending order are ");
for(i=0;i<s;i++)
printf(" %d",a[i]);
 return 0;
}
```

OUTPUT:

Enter total elements: 5

Enter 5 elements: 25

10

68

1

7

the sorted elements in ascending order are: 1 7 10 25 68

C) INSERTION SORT

```
#include<stdio.h>
int main()
{
int i,j,s,temp,a[20];
printf("Enter no of elements: ");
scanf("%d",&s);
printf("Enter %d elements: ",s);
for(i=0;i<s;i++)
scanf("%d",&a[i]);
for(i=1;i<s;i++)
{
temp=a[i]; j=i-1;
while((temp<a[j])&&(j>=0))
{
a[j+1]=a[j];
j=j-1;
}
a[j+1]=temp;
}
printf(" the sorted elements in ascending order are ");
 for(i=0;i<s;i++)
printf(" %d",a[i]);
return 0;
}
```

OUTPUT

Enter no of elements: 10

Enter 10 elements: 25

1

0

36

14

85

73

52

19

8

the sorted elements in ascending order are 0 1 8 14 19 25 36 52 73 85

7. Write a program that use both recursive and non-recursive functions to perform the following searching operations for a Key value in a given list of integers:

i) Linear search ii) Binary search

RECURSIVE LINEAR SEARCH

```
#include<stdio.h>
int lin_search(int[],int,int);
int main()
{
int a[100],n,i,ele;
printf("Enter no of elements");
scanf("%d",&n);
 printf("Enter %d elements",n);
for(i=0;i<n;i++)
scanf("%d",&a[i]);
 printf("The array elements");
for(i=0;i<n;i++)
printf("%4d",a[i]);
printf("\nEnter element to search");
scanf("%d",&ele);
 i=lin_search(a,n-1,ele);
 if(i!=-1)
 printf("\nThe element %d found at %d location",ele,i+1);
else
 printf("\nThe element %d is not found",ele);
 return 0;
}
int lin_search(int x[100],int n,int ele)
{
if(n<=0)
```

```
 return -1;
 if(x[n]==ele)
 return n;
 else
 return lin_search(x,n-1,ele);
}
```

OUTPUT

```
Enter no of elements 5
Enter 5 elements25
36
45
58
96
The array elements  25  36  45  58  96
Enter element to search45
The element 45 found at 3 location
```

NON RECURSIVE LINEAR SEARCH

```
#include <stdio.h>
#include<stdlib.h>
int main()
{
int key;
int n;
int i,a[n];
printf("\n Enter the size of the array : ");
scanf("%d", &n);
printf("\n Enter %d integers :", n);
for(i = 0; i < n; i++)
scanf("%d", &a[i]);
printf("\n Enter a value to be searched : ");
scanf("%d", &key);
for(i=0; i<n; i++)
 if(key == a[i])
{
printf("Key value is found at position %d \n",i+1);
 exit(0);
}
printf("\n Given key is not found in the list");
return 0;
}
```

OUTPUT

```
Enter the size of the array : 8
Enter 8 integers :
45
63
85
12
```

74
95
4
6

Enter a value to be searched : 63
Key value is found at position 2

RECURSIVE BINARY SEARCH

```
#include<stdio.h>
void binary_search(int [], int, int, int);
 int main()
{
   int key, size, i;
   int list[25];
   printf("Enter size of a list:");
   scanf("%d", &size);
   printf("Enter elements\n");
   for(i =0; i<size; i++)
   {
      scanf("%d",&list[i]);
   }
   printf("\n");
   printf("Enter key to search\n");
   scanf("%d", &key);
   binary_search(list, 0, size, key);
}
void binary_search(int list[], int lo, int hi, int key)
{
   int mid;
   if (lo > hi)
   {
      printf("Key not found\n");
      return;
   }
   mid=(lo+hi)/2;
```

```
   if (list[mid] == key)
   {
      printf("Key found\ at %d n",mid+1);
   }
   else if (list[mid] > key)
   {
       binary_search(list, lo, mid-1, key);
   }
   else if (list[mid] < key)
  {
       binary_search(list, mid+1, hi, key);
   }

}
```

OUTPUT:

```
Enter size of a list:6
Enter elements
12
15
28
47
69
89

Enter key to search
56
Key not found
```

Enter size of a list:5

Enter elements

10

15

25

35

45

Enter key to search

15

Key found at 2

NON RECURSIVE BINARY SEARCH

```
#include<stdio.h>
int main()
{
 int a[20], i, n, key, low, high, mid;
printf("enter no of elelments");
scanf("%d",&n);
 printf("Enter the array elements in ascending order");
 for(i=0;i<n;i++)
 {
  scanf("%d",&a[i]);
 }
 printf("Enter the key element\n");
 scanf("%d",&key);
 low=0;
 high=n-1;
 while(high>=low)
 {
  mid=(low+high)/2;
  if(key == a[mid])
   break;
  else
  {
   if(key > a[mid])
    low = mid+1;
   else
    high = mid-1;
  }
 }
 if(key == a[mid])
  printf("The key element is found at location %d", mid+1);
 else
  printf("the key element is not found");
 return 0;
}
```

OUTPUT

```
enter no of elelments10
Enter the array elements in ascending order10
25
36
45
58
69
75
89
92
120
Enter the key element
75
The key element is found at location 7
```

8. Write a program to implement the tree traversal methods.

TREE TRAVERSALS

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

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

node *create();
void insert(node *, node *);
void preorder(node *);
void postorder(node *);
void inorder(node *);

int main()
{
   int ch;
   node *root = NULL, *temp, *current;
   printf("\nEnter the number of Nodes you want :\t");
   scanf("%d", &ch);
   printf("\nEnter %d Nodes data :\n", ch);
   do
   {
      temp = create();
```

```
        if (root == NULL)
            root = temp;
        else
            insert(root, temp);
        ch--;

    } while (ch != 0);

    printf("\nPreorder Traversal\n");
    preorder(root);

    printf("\nInorder Traversal\n");
    inorder(root);

    printf("\nPostorder Traversal\n");
    postorder(root);

    printf("\n");

    return 0;
}

node *create()
{
    node *temp;

    temp = (node *)malloc(sizeof(node));
    scanf("%d", &temp->data);
    temp->left = temp->right = NULL;
```

```
    return temp;
}

void insert(node *root, node *temp)
{
    if (root == NULL)
    {
        root = temp;
    }
    else
    {

        if (temp->data < root->data)
        {
            if (root->left != NULL)
                insert(root->left, temp);
            else
                root->left = temp;
        }

        if (temp->data > root->data)
        {
            if (root->right != NULL)
                insert(root->right, temp);
            else
                root->right = temp;
        }
    }
}
```

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

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

void inorder(node *root)
{
   if (root != NULL)
   {
      inorder(root->left);
      printf("%d \t", root->data);
      inorder(root->right);
   }
```

}

OUTPUT

Enter the number of Nodes you want : 8

Enter 8 Nodes data :

25

63

98

78

45

100

2

84

Preorder Traversal

25 2 63 45 98 78 84 100

Inorder Traversal

2 25 45 63 78 84 98 100

Postorder Traversal

2 45 84 78 100 98 63 25

9. Write a program to implement the graph traversal methods.

```
#include<stdio.h>

int q[20],top=-1,front=-1,rear=-1,a[20][20],vis[20],stack[20];
int delete();
void add(int item);
void bfs(int s,int n);
void dfs(int s,int n);
void push(int item);
int pop();
void main()
{
int n,i,s,ch,j;
char c,dummy;
printf("ENTER THE NUMBER VERTICES ");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
printf("ENTER 1 IF %d HAS A NODE WITH %d ELSE 0 ",i,j);
scanf("%d",&a[i][j]);
}
}
printf("THE ADJACENCY MATRIX IS\n");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
```

```
{
printf(" %d",a[i][j]);
}
printf("\n");
}
do
{
for(i=1;i<=n;i++)
vis[i]=0;
printf("\nMENU");
printf("\n1.B.F.S");
printf("\n2.D.F.S");
printf("\nENTER YOUR CHOICE");
scanf("%d",&ch);
printf("ENTER THE SOURCE VERTEX :");
scanf("%d",&s);
switch(ch)
{
case 1:bfs(s,n);
break;
case 2:
dfs(s,n);
break;
}
printf("DO U WANT TO CONTINUE(Y/N) ? ");
scanf("%c",&dummy);
scanf("%c",&c);
}while((c=='y')||(c=='Y'));
}
```

```
//**************BFS(breadth-first search) code**************//
void bfs(int s,int n)
{
int p,i;
add(s);
vis[s]=1;
p=delete();
if(p!=0)
printf(" %d",p);
while(p!=0)
{
for(i=1;i<=n;i++)
if((a[p][i]!=0)&&(vis[i]==0))
{
add(i);
vis[i]=1;
}
p=delete();
if(p!=0)
printf(" %d ",p);
}
for(i=1;i<=n;i++)
if(vis[i]==0)
bfs(i,n);
}
void add(int item)
{
```

```
if(rear==19)
printf("QUEUE FULL");
else
{
if(rear==-1)
{
q[++rear]=item;
front++;
}
else
q[++rear]=item;
}
}
int delete()
{
int k;
if((front>rear)||(front==-1))
return(0);
else
{
k=q[front++];
return(k);
}
}
//***************DFS(depth-first search) code******************//
void dfs(int s,int n)
{
int i,k;
push(s);
```

```
vis[s]=1;
k=pop();
if(k!=0)
printf(" %d ",k);
while(k!=0)
{
for(i=1;i<=n;i++)
if((a[k][i]!=0)&&(vis[i]==0))
{
push(i);
vis[i]=1;
}
k=pop();
if(k!=0)
printf(" %d ",k);
}
for(i=1;i<=n;i++)
if(vis[i]==0)
dfs(i,n);
}
void push(int item)
{
if(top==19)
printf("Stack overflow ");
else
stack[++top]=item;
}
int pop()
{
```

```
int k;
if(top==-1)
return(0);
else
{
k=stack[top--];
return(k);
}
}
```

OUTPUT:

```
ENTER THE NUMBER VERTICES 3
ENTER 1 IF 1 HAS A NODE WITH 1 ELSE 0 0
ENTER 1 IF 1 HAS A NODE WITH 2 ELSE 0 1
ENTER 1 IF 1 HAS A NODE WITH 3 ELSE 0 1
ENTER 1 IF 2 HAS A NODE WITH 1 ELSE 0 1
ENTER 1 IF 2 HAS A NODE WITH 2 ELSE 0 1
ENTER 1 IF 2 HAS A NODE WITH 3 ELSE 0 1
ENTER 1 IF 3 HAS A NODE WITH 1 ELSE 0 0
ENTER 1 IF 3 HAS A NODE WITH 2 ELSE 0 1
ENTER 1 IF 3 HAS A NODE WITH 3 ELSE 0 0
THE ADJACENCY MATRIX IS
 0 1 1
 1 1 1
 0 1 0

MENU
1.B.F.S
2.D.F.S
ENTER YOUR CHOICE1
```

ENTER THE SOURCE VERTEX :2

2 1 3 DO U WANT TO CONTINUE(Y/N) ? y

MENU

1.B.F.S

2.D.F.S

ENTER YOUR CHOICE2

ENTER THE SOURCE VERTEX :2

2 3 1 DO U WANT TO CONTINUE(Y/N) ? n