MALLAREDDY ENGINEERING COLLEGE
(Autonomous)

Maisammaguda, Dhulapally (post&viaKompally), Secunderabad-500100.

Department of Computer Science & Information Technology

DATA STRUCTURES

SUBJECT CODE: C0512

B .Tech-III Semester (MR23)

ACADEMIC YEAR: 2024-2025

Malla Reddy Engineering College

(UGC Autonomous Institution, Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad and Accredited 3rd cycle by NAAC with 'A++' Grade)
Maisammaguda (H), Medchal-Malkajgiri District, Secunderabad, Telangana State – 500100, www.mrec.ac.in

Department of Computer Science & Information Technology

INSTITUTION VISION

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

INSTITUTION MISSION

- 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.

	Malla Reddy Engineering College (UGC Autonomous Institution, Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad and Accredited 3rd cycle by NAAC with ‘A++’ Grade) Maisammaguda (H), Medchal-Malkajgiri District, Secunderabad, Telangana State – 500100, www.mrec.ac.in

Department of Computer Science & Information Technology

DEPARTMENT VISION

To Attain Global Standards in the Teaching, Training, and Research of the IT Industry that Strike a Balance between the Rising Needs of the Sector and the Socio-Economic and Ethical Needs of the Society.

DEPARTMENT MISSION

M1: To Impart Quality Education and Research to the Students in Information Technology.

M2: To Train the Students in Advanced Technologies using State-of-the-Art Facilities.

M3:To Build the Knowledge, Skills and Aptitude to Succeed in the Information Technology Industry through Ethical Values and Social Relevance.

.

	Malla Reddy Engineering College (UGC Autonomous Institution, Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad and Accredited 3rd cycle by NAAC with 'A++' Grade) Maisammaguda (H), Medchal-Malkajgiri District, Secunderabad, Telangana State – 500100, www.mrec.ac.in

Department of Computer Science & Information Technology

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO1: To outshine in professional career with sound problem solving ability for providing IT solutions by proper plan, analysis, design, implementation and validation.

PEO2: To pursue training, advance study and research using scientific, technical and communication base to cope with the evolution in the technology.

PEO3: To utilize the acquired technical skills and knowledge for the benefit of society

PROGRAMME SPECIFIC OUTCOMES (PSOs)

PSO1: Identify the mathematical abstractions and algorithm design techniques together with emerging Software Tools to solve complexities indulged in efficient programming.

PSO2: Apply the core concepts of current technologies in the hardware, software domains in accomplishing IT enabled services to meet out societal needs.

PSO3: Practice modern computing techniques by continual learning process with ethical concerns in establishing innovative career path.

	Malla Reddy Engineering College (UGC Autonomous Institution, Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad and Accredited 3rd cycle by NAAC with 'A++' Grade) Maisammaguda (H), Medchal-Malkajgiri District, Secunderabad, Telangana State – 500100, www.mrec.ac.in

Department of Computer Science & Information Technology

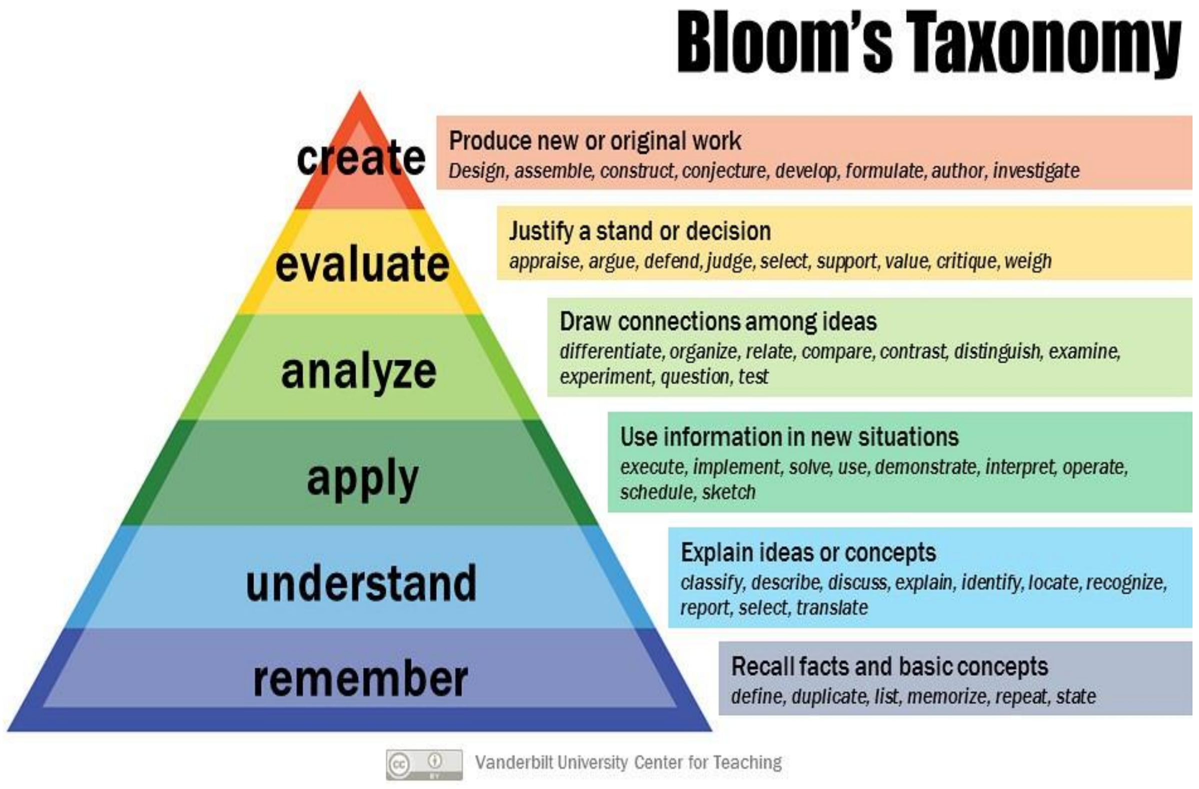
PROGRAMME OUTCOMES (POs)

PO 1	**Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO 2	**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.
PO 3	**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.
PO 4	**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.
PO 5	**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.
PO 6	**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.
PO 7	**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.
PO 8	**Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO 9	**Individual and team work**: Function effectively as an individual and as a member or leader in diverse teams, and in multidisciplinary settings.
PO 10	**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.
PO 11	**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.

PO 12	**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.

Department of Computer Science & Information Technology

Bloom’s Taxonomy Triangle


Bloom’s Taxonomy
create
Produce new or original work
Design, assemble, construct, conjecture, develop, formulate, author, investigate
evaluate
Justify a stand or decision
appraise, argue, defend, judge, select, support, value, critique, weigh
analyze
Draw connections among ideas
differentiate, organize, relate, compare, contrast, distinguish, examine, experiment, question, test
apply
Use information in new situations
execute, implement, solve, use, demonstrate, interpret, operate, schedule, sketch
understand
Explain ideas or concepts
classify, describe, discuss, explain, identify, locate, recognize, report, select, translate
remember
Recall facts and basic concepts
define, duplicate, list, memorize, repeat, state
Vanderbilt University Center for Teaching

2024-25 Onwards (MR-23)	Malla Reddy Engineering College	B.Tech. III Semester		
Code: C0512	**Data Structures Lab** (Common for CSE, CSE(DS), CSE(AI and ML), CSE(Cyber Security), CSE(IOT) CSITand IT)	L	T	P
Credits: 1.5		-	-	3

Prerequisites: A Course on “Programming for problem solving”

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: 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 Linear Queue (its operations) using
i) Arrays ii) Pointers
6 Write a program that implement Deque (its operations) using
i) Arrays ii) Pointers
7 Write a program to implement all the functions of a dictionary using hashing.
8 Write a program that implement Binary Search Trees to perform the following operations
i) Creation ii) Insertion iii) Deletion iv) Traversal
9 Write a program to implement the tree traversal methods using recursion.
10 Write a program that implements the following sorting methods to sort a given list ofintegers in ascending order
i) Heap sort ii) Merge sort
11 Write a program to implement the graph traversal methods such as BFS and DFS.
12 Write a program to implement the Knuth-Morris-Pratt pattern matching algorithm.

Text Books

1. Fundamentals of data structures in C, E.Horowitz, S.Sahni and Susan Anderson Freed, 2ndEdition, Universities Press.
2. Data structures using C, A.S.Tanenbaum, Y. Langsam, and M.J. Augenstein, PHI/pearsoneducation.

References

1. Data structures: A Pseudocode Approach with C, R.F.Gilberg And B.A.Forouzan, 2nd Edition,Cengage Learning.
2. Introduction to data structures in C, Ashok Kamthane, 1st Edition, PEARSON.

Outcomes:

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

1. Develop C programs for computing and real life applications using basic data structures likestacks, queues, linked lists, Binary Search Trees.
2. Make use of basic data structures implementing various tree and graph traversal operations andalgorithms.
3. Apply the concepts of basic data structures and implement advanced operations AVL Trees, Red–Black Trees, and Splay Trees concepts.

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**	**PSO 1**	**PSO 2**	**PSO 3**
CO 1	2	3	2										3	2	
CO 2	2	2	3										2	3	
CO 3		2	3										2	3	1

1. **Write a C program that uses functions to perform the following:**
    a) **Create a singly linked list of integers.**
    b) **Delete a given integer from the above linked list.**
    c) **Display the contents of the above list after deletion.**

Aim: Program to implement Single Linked List Operations

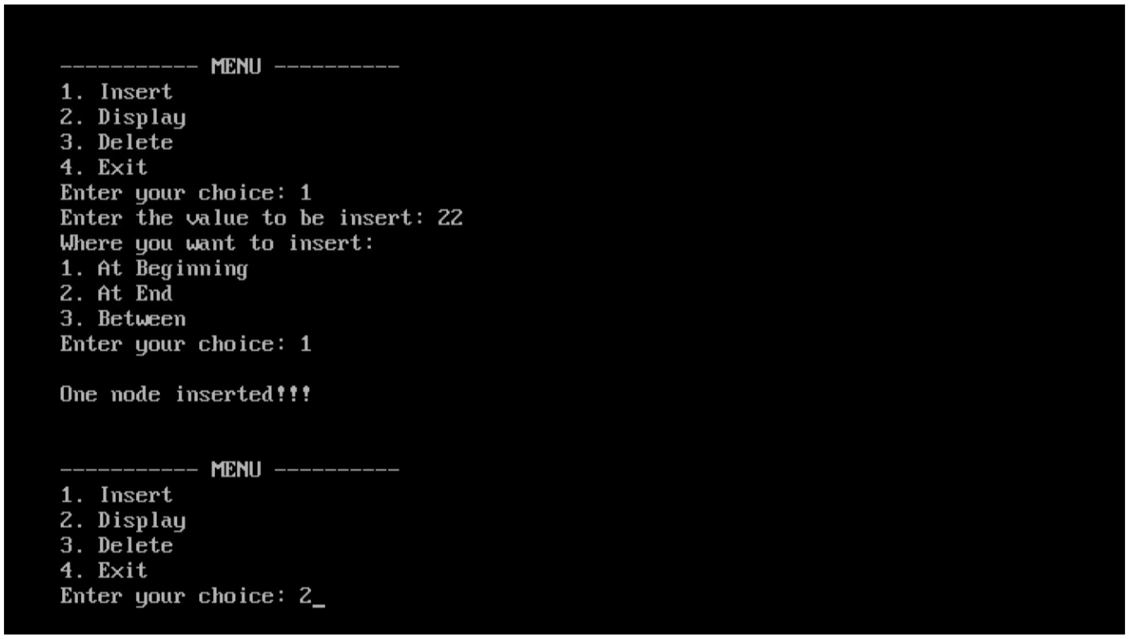
Program:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
// User define functions
void insertAtBeginning(int);
void insertAtEnd(int);
void insertAtSpecific(int,int,int);
void display();
void deleteAtBeginning();
void deleteAtEnd();
void deleteAtSpecific(int);

struct Node    //self reference structure
{
  int data;
  struct Node *next;
}*head = NULL;

void main()
{
  int choice,value,choice1,pos1,pos2;
  clrscr();
  while(1){
  mainMenu:
  printf("\n\n---------------- MENU ------------- “)
  printf(“n1. Insert\n2. Display\n3. Delete\n4.      Exit\n Enter your choice: ");
  scanf("%d",&choice);
  switch(choice)
  {
    case 1:       printf("Enter the value to be insert: ");
                  scanf("%d",&value);
                  while(1){
                  printf("Where you want to insert: \n 1. At Beginning \n2. At End \n3. Between
                  \n Enter your choice: ");
                  scanf("%d",&choice1);
                  switch(choice1)
                  {
                    case 1:         insertAtBeginning(value);
                                    break;
                     case 2:        insertAtEnd(value);
                                    break;
                    case 3:     printf("Enter the two values where you wanto insert: ");
                                scanf("%d%d",&pos1,&pos2);
                                insertAtSpecific(value,pos1,pos2);
                                break;
                  default:      printf("\nEntercorrect Options !!!\n\n");
                                goto mainMenu;
                }
                goto subMenuEnd;
                }
                subMenuEnd:
                break;
     case 2:    display();
                break;
    case 3:
                printf("How do you want to Delete: \n1. From Beginning\n2. From End\n3.
                Spesific\nEnter your choice: ");
                scanf("%d",&choice1);
                switch(choice1)
                {
                  case 1:       deleteAtBeginning();
                                break;
                  case 2:       deleteAtEnd(value);
                                break;
                  case 3:     printf("Enter the value which you wanto delete: ");
                                scanf("%d",&pos2);
                                deleteAtSpecific(pos2);
                                break;
                  default:      printf("\nWrong Input!! Try again!!!\n\n");
                                goto mainMenu;
                }
                break;
     case 4:
                exit(0);
     default:
                printf("\nWrong input!!! Try again!!\n\n");
   }
   }
}
```

void insertAtBeginning(int value)

```
{
  struct Node *newNode; //create new node
  newNode = (struct Node*)malloc(sizeof(struct Node));      //Memory allocation for new
node
  newNode->data = value;       // assign value to new node
  if(head == NULL)       // to check wether the list is empty or not

  {
    newNode->next = NULL;
    head = newNode;
  }
  else
  {
    newNode->next = head;
    head = newNode;
  }
  printf("\nOne node inserted!!!\n");
}
```
void insertAtEnd(int value)
```
{
  struct Node *newNode;
  newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = value;
  newNode->next = NULL;
  if(head == NULL)
        head = newNode;
  else
  {
    struct Node *temp = head;
    while(temp->next != NULL)
        temp = temp->next;
    temp->next = newNode;
  }
  printf("\nOne node inserted!!!\n");
}
```
void insertAtSpecific(int value, int pos1, int pos2)
```
{
  struct Node *newNode;
  newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = value;
  if(head == NULL)
  {
    newNode->next = NULL;
    head = newNode;
  }
  else
  {
    struct Node *temp = head;    //initilize temp ptr with head
    while(temp->data != pos1 && temp->data != pos2)
        temp = temp->next;
    newNode->next = temp->next;
    temp->next = newNode;
  }
  printf("\nOne node inserted!!!\n");
}
```

void deleteAtBeginning()
```
{
  if(head == NULL)
        printf("\n\nList is Empty!!!");
  else
  {
    struct Node *temp = head;
    if(head->next == NULL)
    {
         head = NULL;
         free(temp);
    }
    else
    {
        head = temp->next;
        free(temp);
        printf("\nOne node deleted!!!\n\n");
    }
  }
}
```
void deleteAtEnd()
```
{
  if(head == NULL)
  {
    printf("\nList is Empty!!!\n");
  }
  else
  {
    struct Node *temp1 = head,*temp2;
    if(head->next == NULL)
        head = NULL;
    else
    {
         while(temp1->next != NULL)
         {
           temp2 = temp1;
           temp1 = temp1->next;
         }
         temp2->next = NULL;
    }
    free(temp1);
    printf("\nOne node deleted!!!\n\n");
  }
}
```
void deleteAtSpecific(int delValue)
```
{
  struct Node *temp1 = head, *temp2;
  while(temp1->data != delValue)
  {
   if(temp1 -> next == NULL){
        printf("\nGiven node not found in the list!!!");
        goto functionEnd;
   }
   temp2 = temp1;
   temp1 = temp1 -> next;
  }
  temp2 -> next = temp1 -> next;
  free(temp1);
  printf("\nOne node deleted!!!\n\n");
  functionEnd:
}
```
void display()
```
{
  if(head == NULL)
  {
    printf("\nList is Empty\n");
  }
  else
  {
    struct Node *temp = head;
    printf("\n\nList elements are - \n");
    while(temp->next != NULL)
    {
         printf("%d --->",temp->data);
         temp = temp->next;
    }
    printf("%d --->NULL",temp->data);
  }
}
```



```
----------- MENU ----------
1. Insert
2. Display
3. Delete
4. Exit
Enter your choice: 1
Enter the value to be insert: 22
Where you want to insert:
1. At Beginning
2. At End
3. Between
Enter your choice: 1

One node inserted!!!


----------- MENU ----------
1. Insert
2. Display
3. Delete
4. Exit
Enter your choice: 2_
```

2. Write a program that uses functions to perform the following:

- **a)** Create a doubly linked list of integers.
- **b)** Delete a given integer from the above doubly linked list.
- **c)** Display the contents of the above list after deletion.

Aim:

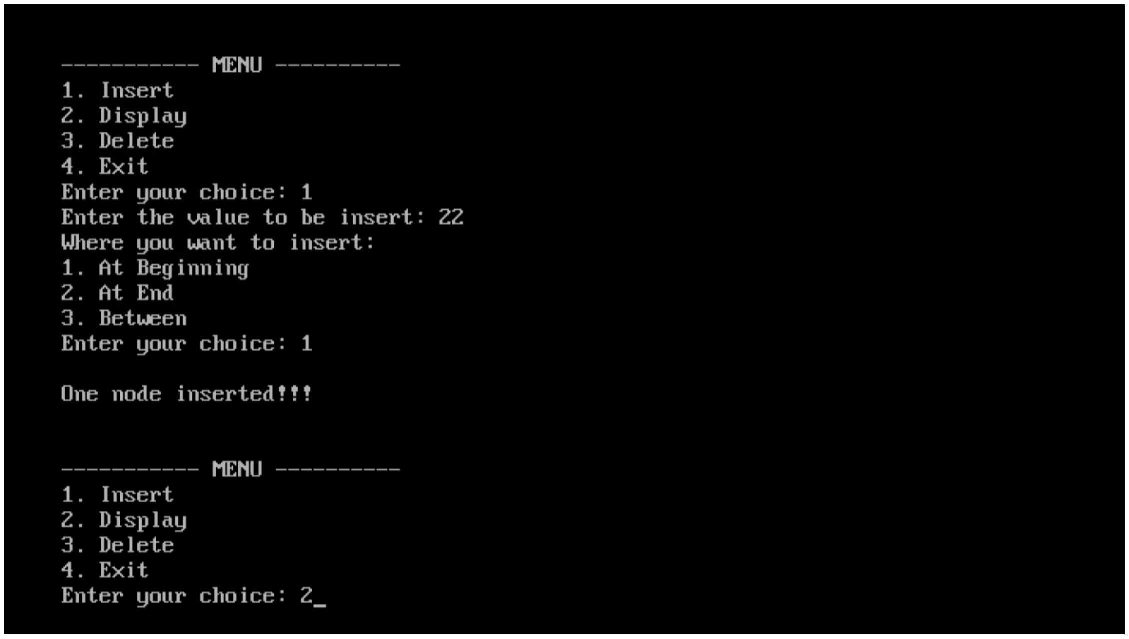
Program

```
#include<stdio.h>
#include<conio.h>
void insertAtBeginning(int);
void insertAtEnd(int);
void  insertAfter(int,int);
void   deleteBeginning();
void deleteEnd();
void  deleteSpecific(int);
void display();
struct Node
{
int data;
struct Node *previous, *next;
}*head = NULL;
void main()
{
int choice1, choice2, value, location;
clrscr();
while(1)
{
printf("\n*********** MENU *************\n");
printf("1. Insert\n2. Delete\n3. Display\n4. Exit\nEnter your choice: ");
scanf("%d",&choice1);
switch(choice1)
{
case 1: printf("Enter the value to be inserted: ");
scanf("%d",&value);
while(1)
{
printf("\nSelect from the following Inserting options\n");
printf("1. At Beginning\n2. At End\n3. After a Node\n4. Cancel\nEnter your choice: ");
scanf("%d",&choice2);
switch(choice2)
{
case 1: insertAtBeginning(value);
break;
case 2: insertAtEnd(value);
break;
case 3: printf("Enter the location after which you want to insert: ");
scanf("%d",&location);
insertAfter(value,location);
break;
case 4: goto EndSwitch;
default: printf("\nPlease select correct Inserting option!!!\n");
}
}
case 2: while(1)
{
printf("\nSelect from the following Deleting options\n");
printf("1. At Beginning\n2. At End\n3. Specific Node\n4. Cancel\nEnter your choice: ");
scanf("%d",&choice2);
switch(choice2)
{
case 1: deleteBeginning();
break;
case 2: deleteEnd();
break;
case 3: printf("Enter the Node value to be deleted: ");
scanf("%d",&location);
deleteSpecific(location);
break;
case 4: goto EndSwitch;
default: printf("\nPlease select correct Deleting option!!!\n");
}
}
EndSwitch: break;
case 3: display();
break;
case 4: exit(0);
default: printf("\nPlease select correct option!!!");
}
}
}
void insertAtBeginning(int value)
{
struct Node *newNode;
newNode = (struct Node*)malloc(sizeof(struct Node));
newNode -> data = value;
newNode -> previous = NULL;
if(head == NULL)
{
newNode -> next = NULL;
head = newNode;
}
else
{
newNode -> next = head;
head = newNode;
}
printf("\nInsertion success!!!");
}
void insertAtEnd(int value)
```

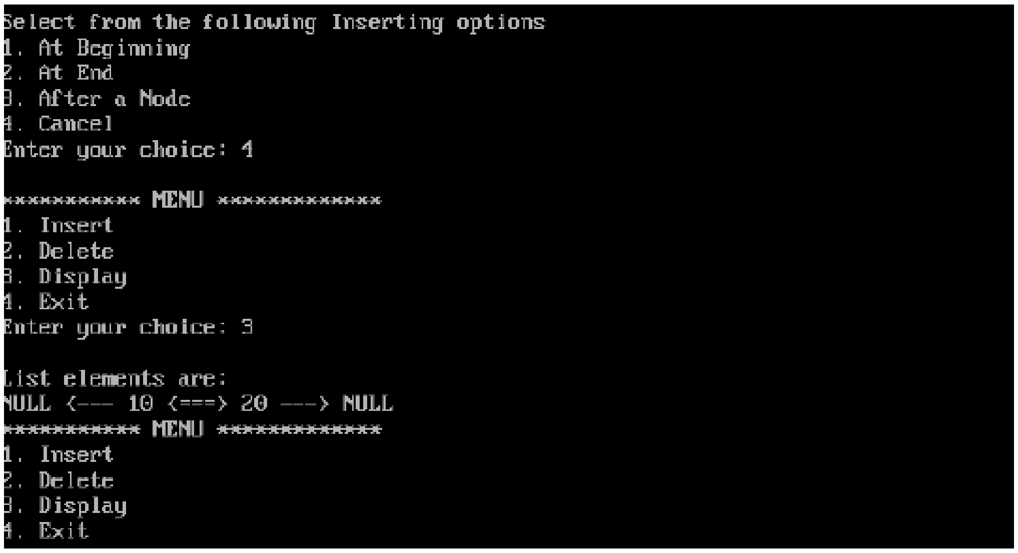
```
{
struct Node *newNode;
newNode = (struct Node*)malloc(sizeof(struct Node));
newNode -> data = value;
newNode -> next = NULL;
if(head == NULL)
{
newNode -> previous = NULL;
head = newNode;
}
else
{
struct Node *temp = head;
while(temp -> next != NULL)
temp = temp -> next;
temp -> next = newNode;
newNode -> previous = temp;
}
printf("\nInsertion success!!!");
}
void insertAfter(int value, int location)
{
struct Node *newNode;
newNode = (struct Node*)malloc(sizeof(struct Node));
newNode -> data = value;
if(head == NULL)
{
newNode -> previous = newNode -> next = NULL;
head = newNode;
}
else
{
struct Node *temp1 = head, *temp2;
while(temp1 -> data != location)
{
if(temp1 -> next == NULL)
{
printf("Given node is not found in the list!!!");
goto EndFunction;
}
else
{
temp1 = temp1 -> next;
}
}
temp2 = temp1 -> next;
temp1 -> next = newNode;
newNode -> previous = temp1;
newNode -> next = temp2;
temp2 -> previous = newNode;
printf("\nInsertion success!!!");
}
EndFunction:
}
void deleteBeginning()
{
if(head == NULL)
printf("List is Empty!!! Deletion not possible!!!");
else
{
struct Node *temp = head;
if(temp -> previous == temp -> next)
{
head = NULL;
free(temp);
}
else{
head = temp -> next;
head -> previous = NULL;
free(temp);
}
printf("\nDeletion success!!!");
}
}
void deleteEnd()
{
if(head == NULL)
printf("List is Empty!!! Deletion not possible!!!");
else
{
struct Node *temp = head;
if(temp -> previous == temp -> next)
{
head = NULL;
free(temp);
}
else{
while(temp -> next != NULL)
temp = temp -> next;
temp -> previous -> next = NULL;
free(temp);
}
printf("\nDeletion success!!!");
}
}
void deleteSpecific(int delValue)
{
if(head == NULL)
printf("List is Empty!!! Deletion not possible!!!");
else
```

```
{
struct Node *temp = head;
while(temp -> data != delValue)
{
if(temp -> next == NULL)
{
printf("\nGiven node is not found in the list!!!");
goto FuctionEnd;
}
else
{
temp = temp -> next;
}
}
if(temp == head) {
head = NULL;
free(temp);
}
else
{
temp -> previous -> next = temp -> next;
free(temp);
}
printf("\nDeletion success!!!");
}
FuctionEnd:
}
void display(){
if(head == NULL)
printf("\nList is Empty!!!");
else
{
struct Node *temp = head;
printf("\nList elements are: \n");
printf("NULL <--- ");
while(temp -> next != NULL)
{
printf("%d <===> ",temp -> data);
}
printf("%d ---> NULL", temp -> data);
}
}
```

OUTPUT:



```
----------- MENU ----------
1. Insert
2. Display
3. Delete
4. Exit
Enter your choice: 1
Enter the value to be insert: 22
Where you want to insert:
1. At Beginning
2. At End
3. Between
Enter your choice: 1

One node inserted!!!


----------- MENU ----------
1. Insert
2. Display
3. Delete
4. Exit
Enter your choice: 2_
```



```
Select from the following Inserting options
1. At Beginning
2. At End
3. After a Node
4. Cancel
Enter your choice: 4

*********** MENU *************
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3

List elements are:
NULL <--- 10 <===> 20 ---> NULL
*********** MENU *************
1. Insert
2. Delete
3. Display
4. Exit
```

3. Write a program that uses functions to perform the following operations on circular linkedlist.

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

```
#include<stdio.h>
#include<conio.h>
void insertAtBeginning(int);
void insertAtEnd(int);
void insertAtAfter(int,int);
void deleteBeginning();
void deleteEnd();
void deleteSpecific(int);
void display();

struct Node
{
  int data;
  struct Node *next;
}*head = NULL;

void main()
{
  int choice1, choice2, value, location;
  clrscr();
  while(1)
  {
    printf("\n*********** MENU *************\n");
    printf("1. Insert\n2. Delete\n3. Display\n4. Exit\nEnter your choice: ");
    scanf("%d",&choice1);
    switch()
    {
      case 1: printf("Enter the value to be inserted: ");
                 scanf("%d",&value);
           while(1)
           {
                printf("\nSelect from the following Inserting options\n");
                printf("1. At Beginning\n2. At End\n3. After a Node\n4. Cancel\nEnter your
choice: ");
              scanf("%d",&choice2);
              switch(choice2)
              {
                case 1:  insertAtBeginning(value);
                         break;
                case 2:  insertAtEnd(value);
                         break;
                case 3:  printf("Enter the location after which you want to insert: ");
                         scanf("%d",&location);
                         insertAfter(value,location);
                         break;
                case 4:  goto EndSwitch;
```

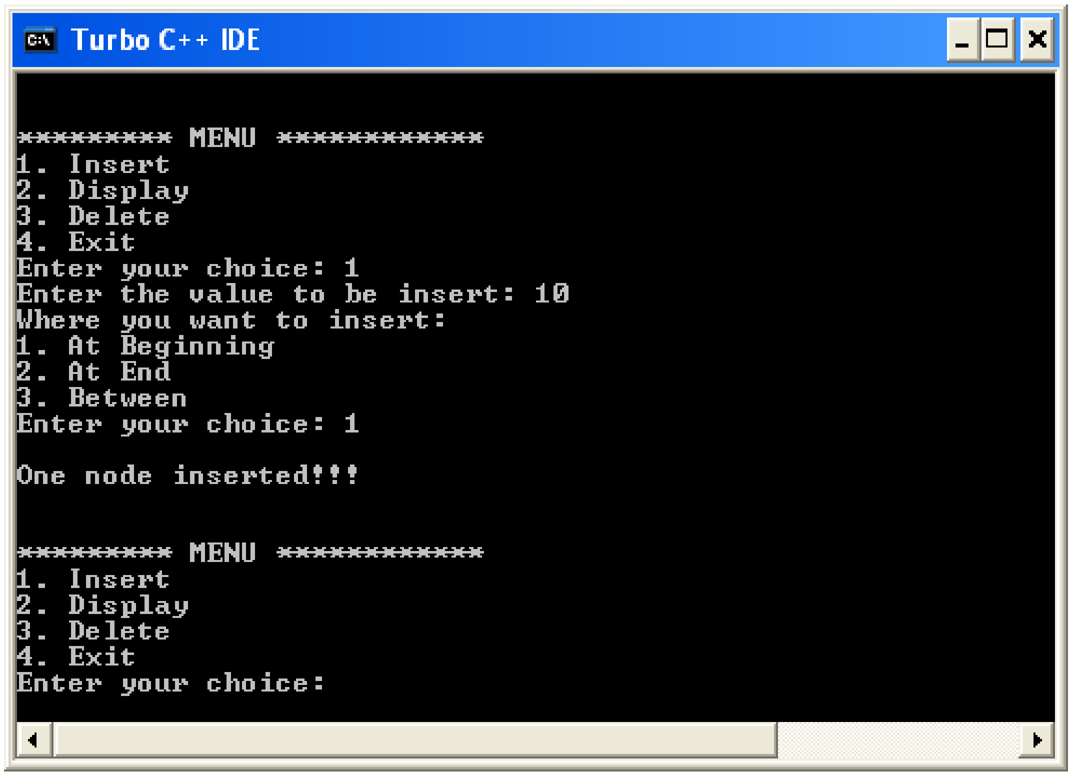
```
                  default: printf("\nPlease select correct Inserting option!!!\n");
                }
              }
        case 2: while(1)
              {
                   printf("\nSelect from the following Deleting options\n");
                   printf("1. At Beginning\n2. At End\n3. Specific Node\n4. Cancel\nEnter your
choice: ");
                scanf("%d",&choice2);
                switch(choice2)
                {
                  case 1:  deleteBeginning();
                           break;
                  case 2:  deleteEnd();
                           break;
                  case 3:  printf("Enter the Node value to be deleted: ");
                           scanf("%d",&location);
                           deleteSpecic(location);
                           break;
                  case 4:  goto EndSwitch;
                  default: printf("\nPlease select correct Deleting option!!!\n");
                }
              }
              EndSwitch: break;
        case 3: display();
           break;
        case 4: exit(0);
        default: printf("\nPlease select correct option!!!");
     }
   }
}

void insertAtBeginning(int value)
{
   struct Node *newNode;
   newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode -> data = value;
   if(head == NULL)
   {
     head = newNode;
     newNode -> next = head;
   }
   else
   {
     struct Node *temp = head;
     while(temp -> next != head)
        temp = temp -> next;
     newNode -> next = head;
     head = newNode;
     temp -> next = head;
```

```
    }
    printf("\nInsertion success!!!");
}
void insertAtEnd(int value)
{
   struct Node *newNode;
   newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode -> data = value;
   if(head == NULL)
   {
      head = newNode;
      newNode -> next = head;
   }
   else
   {
      struct Node *temp = head;
      while(temp -> next != head)
         temp = temp -> next;
      temp -> next = newNode;
      newNode -> next = head;
   }
   printf("\nInsertion success!!!");
}
void insertAfter(int value, int location)
{
   struct Node *newNode;
   newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode -> data = value;
   if(head == NULL)
   {
      head = newNode;
      newNode -> next = head;
   }
   else
   {
      struct Node *temp = head;
      while(temp -> data != location)
      {
         if(temp -> next == head)
         {
            printf("Given node is not found in the list!!!");
            goto EndFunction;
         }
         else
         {
            temp = temp -> next;
         }
      }
      newNode -> next = temp -> next;
      temp -> next = newNode;
    printf("\nInsertion success!!!");
  }
  EndFunction:
}
void deleteBeginning()
{
  if(head == NULL)
    printf("List is Empty!!! Deletion not possible!!!");
  else
  {
    struct Node *temp = head;
    if(temp -> next == head)
    {
      head = NULL;
      free(temp);
    }
    else{
      head = head -> next;
      free(temp);
    }
    printf("\nDeletion success!!!");
  }
}
void deleteEnd()
{
  if(head == NULL)
    printf("List is Empty!!! Deletion not possible!!!");
  else
  {
    struct Node *temp1 = head, temp2;
    if(temp1 -> next == head)
    {
      head = NULL;
      free(temp1);
    }
    else{
      while(temp1 -> next != head){
        temp2 = temp1;
        temp1 = temp1 -> next;
      }
      temp2 -> next = head;
      free(temp1);
    }
    printf("\nDeletion success!!!");
  }
}
void deleteSpecific(int delValue)
{
  if(head == NULL)
    printf("List is Empty!!! Deletion not possible!!!");
```

```
   else
   {
      struct Node *temp1 = head, temp2;
      while(temp1 -> data != delValue)
      {
         if(temp1 -> next == head)
         {
            printf("\nGiven node is not found in the list!!!");
            goto FuctionEnd;
         }
         else
         {
            temp2 = temp1;
            temp1 = temp1 -> next;
         }
      }
      if(temp1 -> next == head){
         head = NULL;
         free(temp1);
      }
      else{
         if(temp1 == head)
         {
            temp2 = head;
            while(temp2 -> next != head)
               temp2 = temp2 -> next;
            head = head -> next;
            temp2 -> next = head;
            free(temp1);
         }
         else
         {
            if(temp1 -> next == head)
            {
               temp2 -> next = head;
            }
            else
            {
               temp2 -> next = temp1 -> next;
            }
            free(temp1);
         }
      }
      printf("\nDeletion success!!!");
   }
   FuctionEnd:
}
void display()
{
   if(head == NULL)
```

```
      printf("\nList is Empty!!!");
   else
   {
      struct Node *temp = head;
      printf("\nList elements are: \n");
      while(temp -> next != head)
      {
         printf("%d ---> ",temp -> data);
      }
      printf("%d ---> %d", temp -> data, head -> data);
   }
}
```

Output:



```

********** MENU *************
1. Insert
2. Display
3. Delete
4. Exit
Enter your choice: 1
Enter the value to be insert: 10
Where you want to insert:
1. At Beginning
2. At End
3. Between
Enter your choice: 1

One node inserted!!!


********** MENU *************
1. Insert
2. Display
3. Delete
4. Exit
Enter your choice:
```

4. WAP for stack implementation using

A. ARRAY

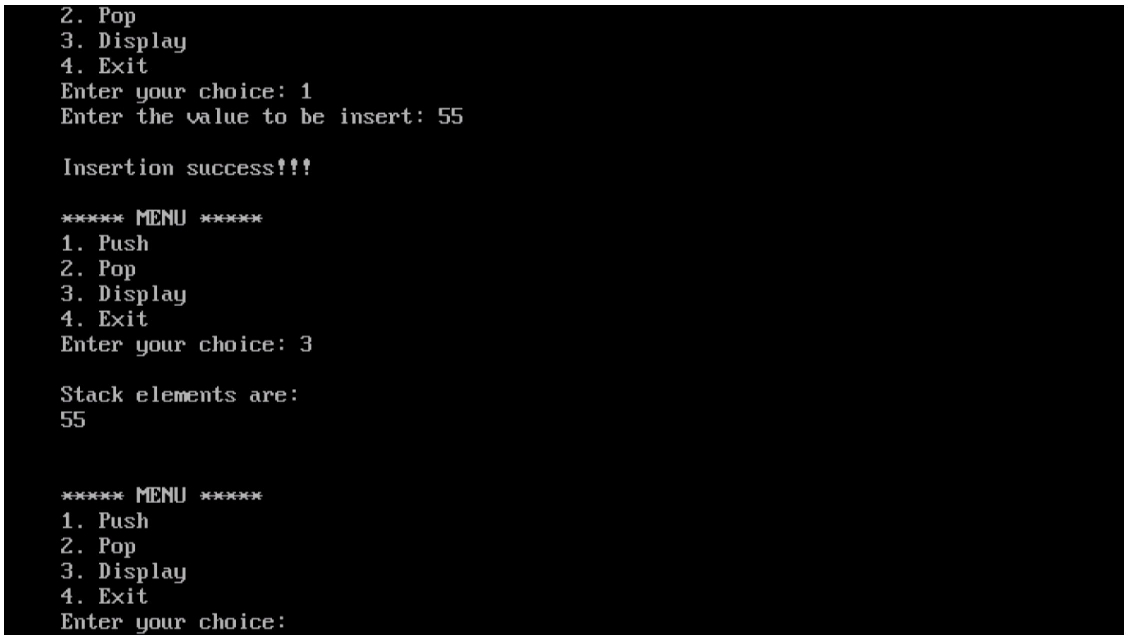
B. LIST

Program:

A. Stack using Array

```
#include<stdio.h>
#include<conio.h>
#define SIZE 10
void push(int);
void pop();
void display();

int stack[SIZE], top = -1;

void main()
{
  int value, choice;
  clrscr();
  while(1){
    printf("\n\n***** MENU *****\n");
    printf("1. Push\n2. Pop\n3. Display\n4. Exit");
    printf("\nEnter your choice: ");
    scanf("%d",&choice);
    switch(choice){
         case 1: printf("Enter the value to be insert: ");
                 scanf("%d",&value);
                 push(value);
                 break;
         case 2: pop();
                 break;
         case 3: display();
                 break;
         case 4: exit(0);
         default: printf("\nWrong selection!!! Try again!!!");
    }
  }
}



void push(int value){
  if(top == SIZE-1)
    printf("\nStack is Full!!! Insertion is not possible!!!");
  else{
    top++;
    stack[top] = value;
    printf("\nInsertion  success!!!");
  }
}
void pop(){
  if(top == -1)
    printf("\nStack is Empty!!! Deletion is not possible!!!");
  else{
    printf("\nDeleted : %d", stack[top]);
    top--;
  }
}
void display(){
  if(top == -1)
    printf("\nStack is Empty!!!");
  else{
    int i;
    printf("\nStack elements are:\n");
    for(i=top; i>=0; i--)
         printf("%d\n",stack[i]);
  }
}
```

<u>OUTPUT:</u>



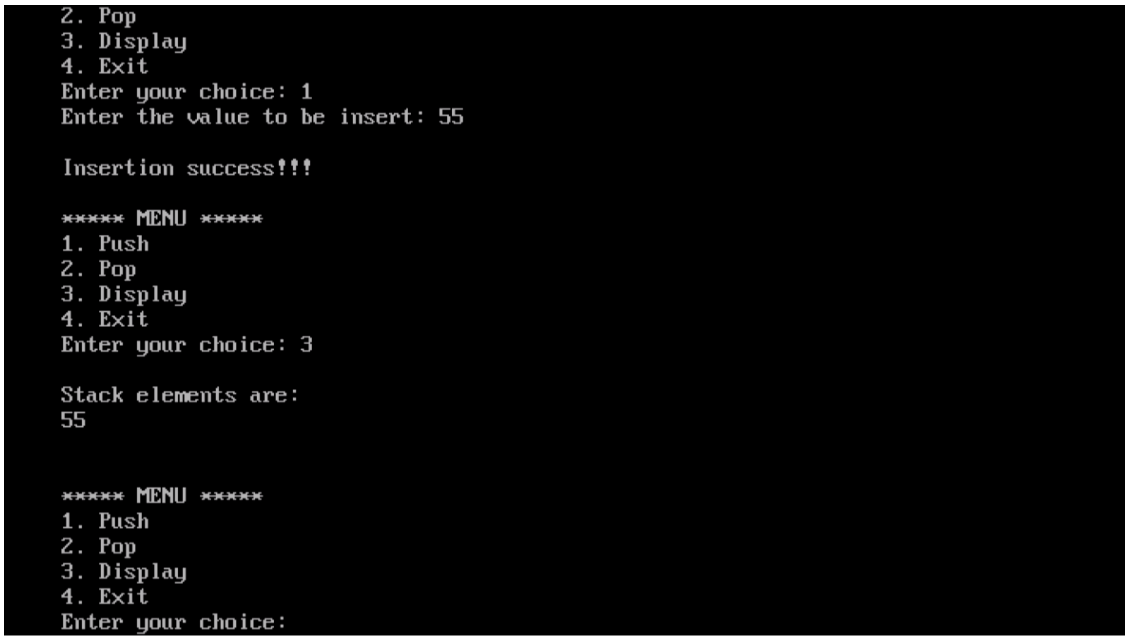
```
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 55

Insertion success!!!

***** MENU *****
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3

Stack elements are:
55


***** MENU *****
1. Push
2. Pop
3. Display
4. Exit
Enter your choice:
```

B. Stack using Linked List

```
#include<stdio.h>
#include<conio.h>
struct Node
{
   int data;
   struct Node *next;
}*top = NULL;
void push(int);
void pop();
void display();
void main()
{
   int choice, value;
   clrscr();
   printf("\n<---Stack using Linked List --->>\n");
   while(1){
      printf("\n****** MENU ******\n");
      printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");
      printf("Enter your choice: ");
      scanf("%d",&choice);
      switch(choice){
             case 1: printf("Enter the value to be insert: ");
                     scanf("%d", &value);
                     push(value);
                     break;
             case 2: pop(); break;
             case 3: display(); break;
             case 4: exit(0);
             default: printf("\nWrong selection!!! Please try again!!!\n");
      }
   }
}
void push(int value)
{
   struct Node *newNode;
   newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   if(top == NULL)
      newNode->next = NULL;
   else
      newNode->next = top;
   top = newNode;
```

```
  printf("\nInsertion is Success!!!\n");
}
void pop()
{
  if(top == NULL)
    printf("\nStack is Empty!!!\n");
  else{
    struct Node *temp = top;
    printf("\nDeleted element: %d", temp->data);
    top = temp->next;
    free(temp);
  }
}
void display()
{
  if(top == NULL)
    printf("\nStack is Empty!!!\n");
  else{
    struct Node *temp = top;
    while(temp->next != NULL){
         printf("%d--->",temp->data);
         temp = temp -> next;
    }
    printf("%d--->NULL",temp->data);
  }
}
```

OUTPUT:



```
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 55

Insertion success!!!

***** MENU *****
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3

Stack elements are:
55


***** MENU *****
1. Push
2. Pop
3. Display
4. Exit
Enter your choice:
```

5. **WAP for Queue implementation using**
a) Arrays b) LIST

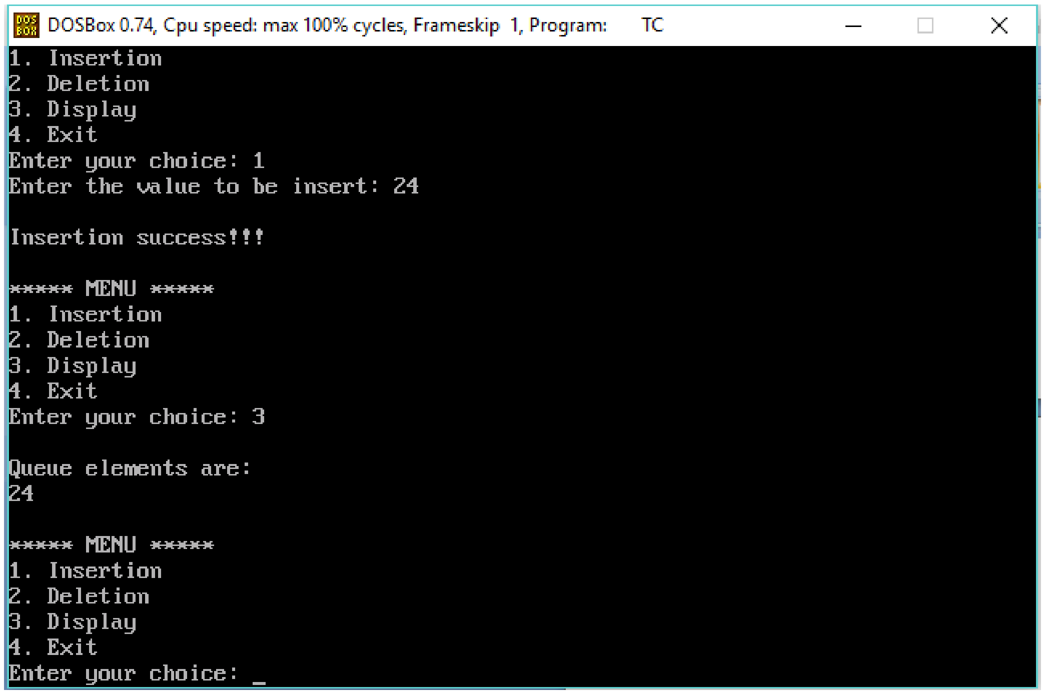
a. Queue Implementation of Array

Program:

```
#include<stdio.h>
#include<conio.h>
#define SIZE 10
void enQueue(int);
void deQueue();
void display();
int queue[SIZE], front = -1, rear = -1;
void main()
{
  int value, choice;
  clrscr();
  while(1){
    printf("\n\n***** MENU *****\n");
    printf("1. Insertion\n2. Deletion\n3. Display\n4. Exit");
    printf("\nEnter your choice: ");
    scanf("%d",&choice);
    switch(choice){
         case 1: printf("Enter the value to be insert: ");
                 scanf("%d",&value);
                 enQueue(value);
                 break;
         case 2: deQueue();
                 break;
         case 3: display();
                 break;
         case 4: exit(0);
         default: printf("\nWrong selection!!! Try again!!!");
    }
  }
}

void enQueue(int value){
  if(rear == SIZE-1)
    printf("\nQueue is Full!!! Insertion is not possible!!!");
  else{
    if(front == -1)
         front = 0;
    rear++;
    queue[rear] = value;
```

```
        printf("\nInsertion success!!!");
    }
}
void deQueue(){
    if(front == rear)
        printf("\nQueue is Empty!!! Deletion is not possible!!!");
    else{
        printf("\nDeleted : %d", queue[front]);
        front++;
        if(front == rear)
            front = rear = -1;
    }
}
void display(){
    if(rear == -1)
        printf("\nQueue is Empty!!!");
    else{
        int i;
        printf("\nQueue elements are:\n");
        for(i=front; i<=rear; i++)
            printf("%d\t",queue[i]);
    }
}
```

OUTPUT:



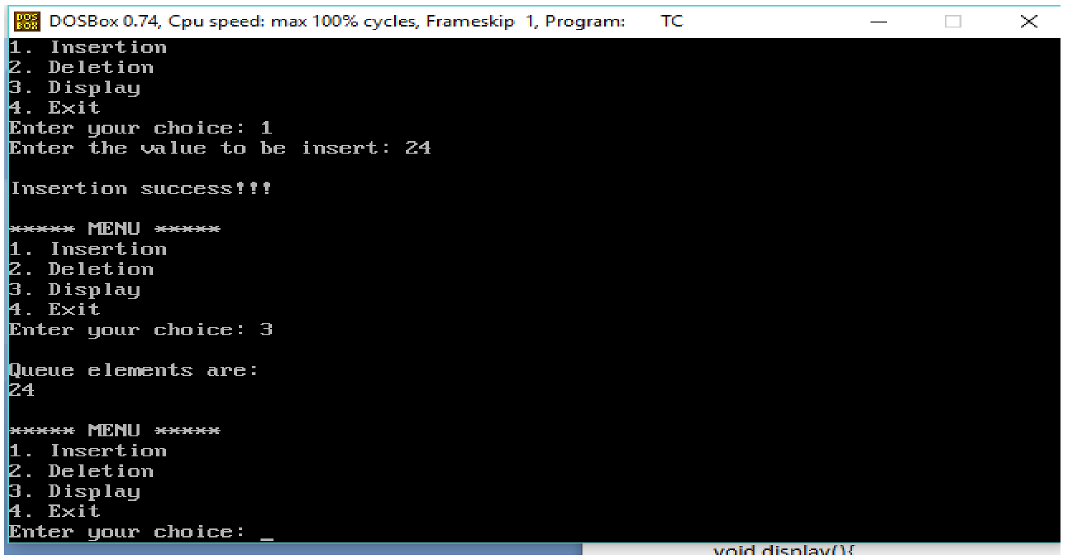
```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 1, Program: TC
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 24

Insertion success!!!

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 3

Queue elements are:
24

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: _
```

b. Queue Implementation using Linked List

```
#include<stdio.h>
#include<conio.h>
struct Node
{
 int data;
   struct Node *next;
}*front = NULL,*rear = NULL;
void insert(int);
void delete();
void display();
void main()
{
   int choice, value;
   clrscr();
   printf("\n:: Queue Implementation using Linked List ::\n");
   while(1){
      printf("\n****** MENU ******\n");
      printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
      printf("Enter your choice: ");
      scanf("%d",&choice);
      switch(choice){
         case 1: printf("Enter the value to be insert: ");
                 scanf("%d", &value);
                 insert(value);
                 break;
         case 2: delete(); break;
         case 3: display(); break;
         case 4: exit(0);
         default: printf("\nWrong selection!!! Please try again!!!\n");
      }
   }
}
void insert(int value)
{
   struct Node *newNode;
   newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode -> next = NULL;
   if(front == NULL)
      front = rear = newNode;
   else{
      rear -> next = newNode;
```

```
    rear = newNode;
  }
  printf("\nInsertion is Success!!!\n");
}
void delete()
{
  if(front == NULL)
    printf("\nQueue is Empty!!!\n");
  else{
    struct Node *temp = front;
    front = front -> next;
    printf("\nDeleted element: %d\n", temp->data);
    free(temp);
  }
}
void display()
{
  if(front == NULL)
    printf("\nQueue is Empty!!!\n");
  else{
    struct Node *temp = front;
    while(temp->next != NULL){
         printf("%d--->",temp->data);
         temp = temp -> next;
    }
    printf("%d--->NULL\n",temp->data);
  } }
```

OUTPUT:



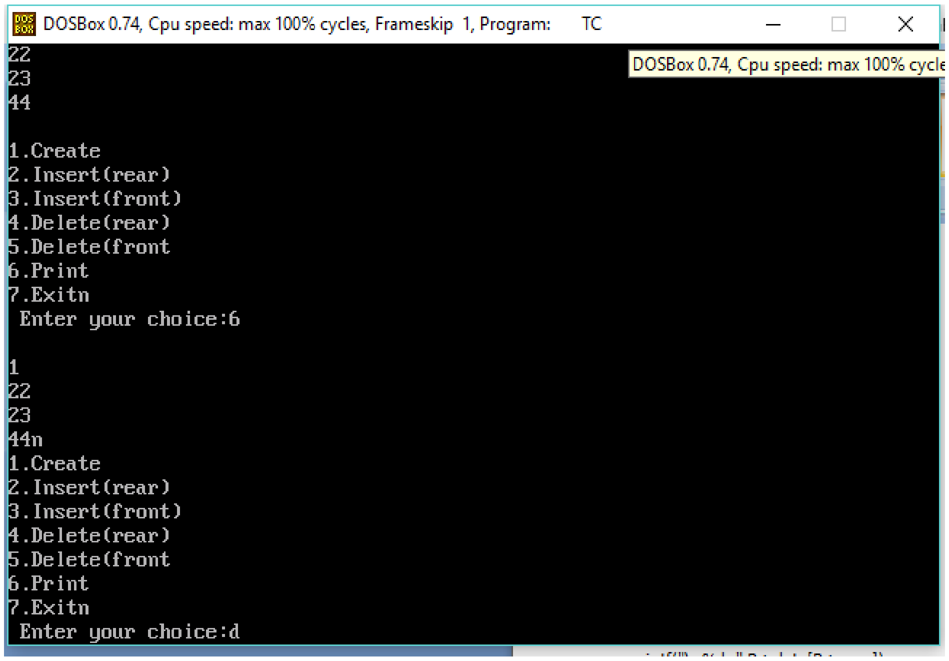
```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 1, Program:    TC
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 24

Insertion success!!!

****** MENU ******
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 3

Queue elements are:
24

****** MENU ******
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: _
```

6. Write C programs to implement a Double Ended Queue ADT

```
#include<stdio.h>
#include<process.h>
#define MAX 30
typedef struct dequeue
{
   int data[MAX];
   int rear,front;
}dequeue;
void initialize(dequeue *p);
int empty(dequeue *p);
int full(dequeue *p);
void enqueueR(dequeue *p,int x);
void enqueueF(dequeue *p,int x);
int dequeueF(dequeue *p);
int dequeueR(dequeue *p);
void print(dequeue *p);
void main()
{
   int i,x,op,n;
   dequeue q;

initialize(&q);
   do
   {
printf("\n1.Create\n2.Insert(rear)\n3.Insert(front)\n4.Delete(rear)\n5.Delete(front)");
            printf("\n6.Print\n7.Exitn\n Enter your choice:");

scanf("%d",&op);
switch(op)
            {
 case 1:
printf("/nEnter number of elements:");
scanf("%d",&n);
initialize(&q);
printf("\nEnter the data:");
for(i=0;i<n;i++)
  {
scanf("%d",&x);
if(full(&q))
{
printf("\nQueue is full!!");
exit(0);
}
enqueueR(&q,x);
   }
break;
case 2:
printf("\nEnter element to be inserted:");
scanf("%d",&x);
if(full(&q))
    {
printf("\nQueue is full!!");
exit(0);
    }
enqueueR(&q,x);
break;
case 3: printf("\nEnter the element to be inserted:");
scanf("%d",&x);
if(full(&q))
{
printf("\nQueue is full!!");
    exit(0);
}
enqueueF(&q,x);
break;
case 4: if(empty(&q))
{
printf("\nQueue is empty!!");
 exit(0);
}
x=dequeueR(&q);
printf("\nElement deleted is %dn",x);
break;
case 5: if(empty(&q))
{
printf("\nQueue isempty!!");
exit(0);

}
x=dequeueF(&q);
printf("\nElement deleted is %dn",x);
break;
case 6: print(&q);
break;
default: break;
            }
    }while(op!=7);
}
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,int 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,int 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;
    }
}
int dequeueF(dequeue *P)
{
    int x;
x=P->data[P->front];
if(P->rear==P->front)
//delete the last element
initialize(P);
    else
P->front=(P->front+1)%MAX;
    return(x);
}
int dequeueR(dequeue *P)
{
    int 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%d",P->data[i]);
i=(i+1)%MAX;
    }
    }
printf("\n%dn",P->data[P->rear]);
}
```

OUTPUT:

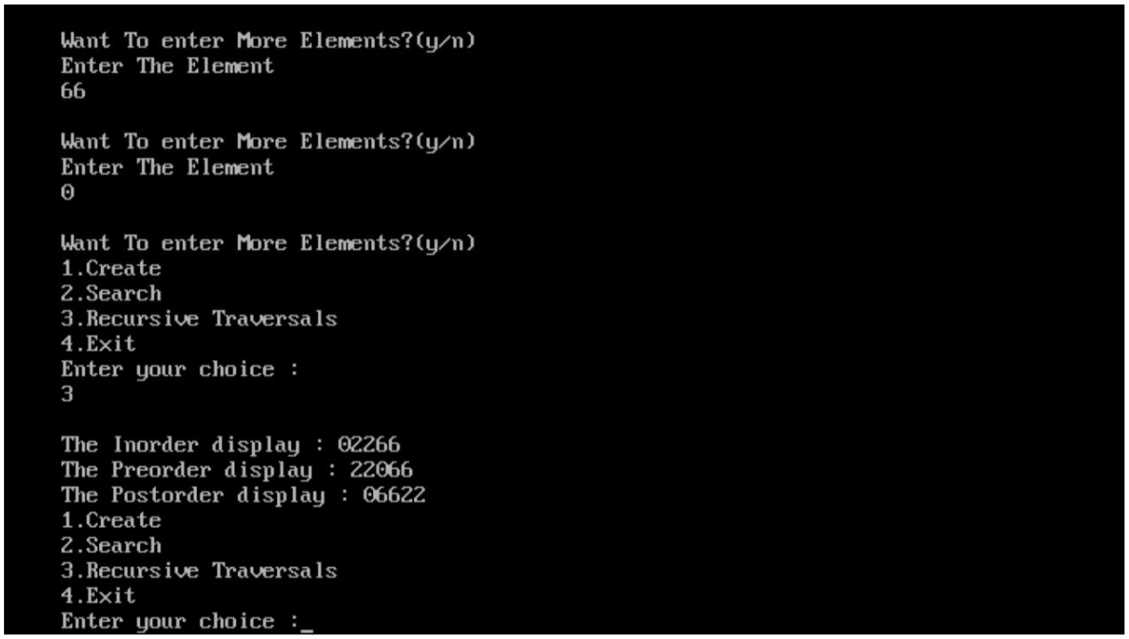


```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip  1, Program:    TC
22
23
44

1.Create
2.Insert(rear)
3.Insert(front)
4.Delete(rear)
5.Delete(front
6.Print
7.Exitn
 Enter your choice:6

1
22
23
44n
1.Create
2.Insert(rear)
3.Insert(front)
4.Delete(rear)
5.Delete(front
6.Print
7.Exitn
 Enter your choice:d
```

7. Write a program to implement all the functions of a dictionary using hashing.
Program:

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

typedef struct BST {
   int data;
   struct BST *lchild, *rchild;
} node;

void insert(node *, node *);
void inorder(node *);
void preorder(node *);
void postorder(node *);
node *search(node *, int, node **);

void main() {
   int choice;
   char ans = 'N';
   int key;
   node *new_node, *root, *tmp, *parent;
   node *get_node();
   root = NULL;
   clrscr();

   printf("\nProgram For Binary Search Tree ");
   do {
      printf("\n1.Create");
      printf("\n2.Search");
      printf("\n3.Recursive Traversals");
      printf("\n4.Exit");
      printf("\nEnter your choice :");
      scanf("%d", &choice);

      switch (choice) {
      case 1:
         do {
            new_node = get_node();
            printf("\nEnter The Element ");
            scanf("%d",  &new_node->data);

            if (root == NULL) /* Tree is not Created */
               root = new_node;
            else
               insert(root, new_node);

            printf("\nWant To enter More Elements?(y/n)");
            ans = getch();
      } while (ans == 'y');
      break;

    case 2:
      printf("\nEnter Element to be searched :");
      scanf("%d", &key);

      tmp = search(root, key, &parent);
      printf("\nParent of node %d is %d", tmp->data, parent->data);
      break;

    case 3:
      if (root == NULL)
        printf("Tree Is Not Created");
      else {
        printf("\nThe Inorder display : ");
        inorder(root);
        printf("\nThe Preorder display : ");
        preorder(root);
        printf("\nThe Postorder display : ");
        postorder(root);
      }
      break;
    }
  } while (choice != 4);
}
/*
 Get new Node
 */
node *get_node() {
  node *temp;
  temp = (node *) malloc(sizeof(node));
  temp->lchild = NULL;
  temp->rchild = NULL;
  return temp;
}
/*
 This function is for creating a binary search tree
 */
void insert(node *root, node *new_node) {
  if (new_node->data < root->data) {
    if (root->lchild == NULL)
      root->lchild = new_node;
    else
      insert(root->lchild, new_node);
  }

  if (new_node->data > root->data) {
    if (root->rchild == NULL)
      root->rchild = new_node;
    else
       insert(root->rchild, new_node);
  }
}
/*
 This function is for searching the node from
 binary Search Tree
 */
node *search(node *root, int key, node **parent) {
  node *temp;
  temp = root;
  while (temp != NULL) {
    if (temp->data == key) {
       printf("\nThe %d Element is Present", temp->data);
       return temp;
    }
    *parent = temp;

    if (temp->data > key)
       temp = temp->lchild;
    else
       temp = temp->rchild;
  }
  return NULL;
}
/*
 This function displays the tree in inorder fashion
 */
void inorder(node *temp) {
  if (temp != NULL) {
    inorder(temp->lchild);
    printf("%d", temp->data);
    inorder(temp->rchild);
  }
}
/*
 This function displays the tree in preorder fashion
 */
void preorder(node *temp) {
  if (temp != NULL) {
    printf("%d", temp->data);
    preorder(temp->lchild);
    preorder(temp->rchild);
  }
}

/*
 This function displays the tree in postorder fashion
 */
void postorder(node *temp) {
        if (temp != NULL) {
          postorder(temp->lchild);
          postorder(temp->rchild);
          printf("%d",  temp->data);
        }
      }
```

OUTPUT:



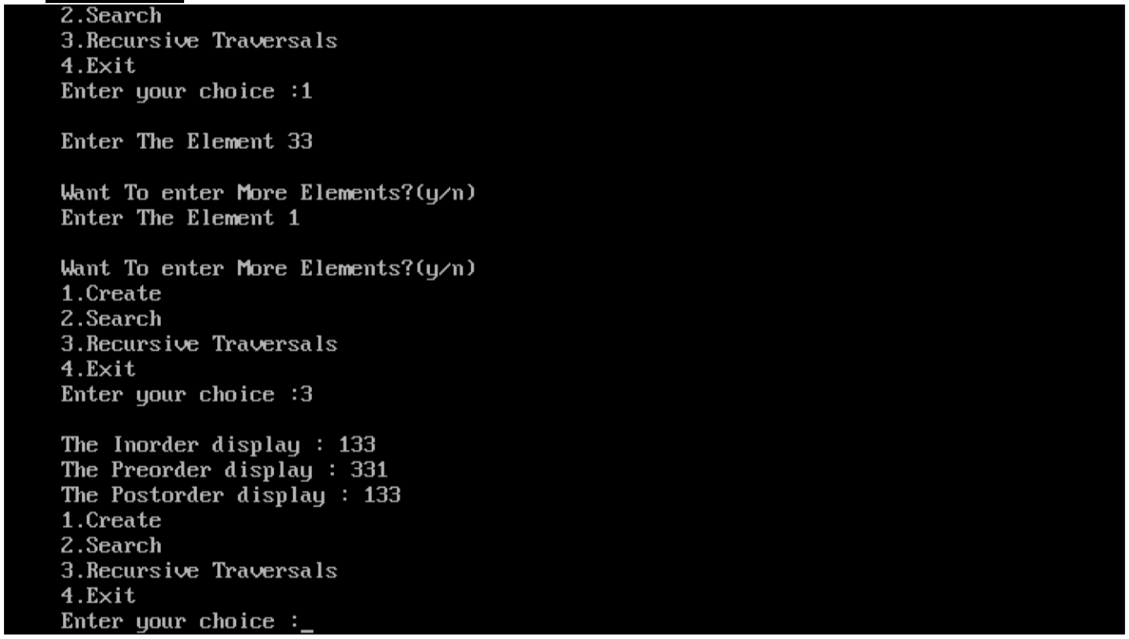
```
Want To enter More Elements?(y/n)
Enter The Element
66

Want To enter More Elements?(y/n)
Enter The Element
0

Want To enter More Elements?(y/n)
1.Create
2.Search
3.Recursive Traversals
4.Exit
Enter your choice :
3

The Inorder display : 02266
The Preorder display : 22066
The Postorder display : 06622
1.Create
2.Search
3.Recursive Traversals
4.Exit
Enter your choice :_
```

8. Write a program that implement Binary Search Trees to perform the following operations i) Creation ii) Insertion iii) Deletion iv) Traversal

```
# include <stdio.h>
# include <conio.h>
# include <stdlib.h>
 typedef struct BST {
   int data;
   struct BST *lchild, *rchild;
} node;
 void insert(node *, node *);
void inorder(node *);
void preorder(node *);
void postorder(node *);
node *search(node *, int, node **);
 void main() {
   int choice;
   char ans = 'N';
   int key;
   node *new_node, *root, *tmp, *parent;
   node *get_node();
   root = NULL;
   clrscr();
    printf("\nProgram For Binary Search Tree ");
   do {
      printf("\n1.Create");
      printf("\n2.Search");
      printf("\n3.Recursive Traversals");
      printf("\n4.Exit");
      printf("\nEnter your choice :");
      scanf("%d", &choice);
       switch (choice) {
      case 1:
         do {
            new_node = get_node();
            printf("\nEnter The Element ");
            scanf("%d",  &new_node->data);
             if (root == NULL) /* Tree is not Created */
               root = new_node;
            else
               insert(root, new_node);
             printf("\nWant To enter More Elements?(y/n)");
            ans = getch();
         } while (ans == 'y');
         break;

      case 2:
         printf("\nEnter Element to be searched :");
         scanf("%d", &key);
          tmp = search(root, key, &parent);
         printf("\nParent of node %d is %d", tmp->data, parent->data);
         break;
        case 3:
         if (root == NULL)
            printf("Tree Is Not Created");
         else {
            printf("\nThe Inorder display : ");
            inorder(root);
            printf("\nThe Preorder display : ");
            preorder(root);
            printf("\nThe Postorder display : ");
            postorder(root);
         }
         break;
      }
   } while (choice != 4);
}
/*  Get new Node  */
node *get_node() {
   node *temp;
   temp = (node *) malloc(sizeof(node));
   temp->lchild = NULL;
   temp->rchild = NULL;
   return temp;
}
/*  This function is for creating a binary search tree */
void insert(node *root, node *new_node) {
   if (new_node->data < root->data) {
      if (root->lchild == NULL)
         root->lchild = new_node;
      else
         insert(root->lchild, new_node);
   }
   if (new_node->data > root->data) {
      if (root->rchild == NULL)
         root->rchild = new_node;
      else
         insert(root->rchild, new_node);
   }        }
/* This function is for searching the node from binary Search Tree */
node *search(node *root, int key, node **parent) {
   node *temp;
   temp = root;
   while (temp != NULL) {
      if (temp->data == key) {
         printf("\nThe %d Element is Present", temp->data);
         return temp;
      }
      *parent = temp;
       if (temp->data > key)
```

```
            temp = temp->lchild;
        else
            temp = temp->rchild;
    }
    return NULL;
}

/* This function displays the tree in inorder fashion */
void inorder(node *temp) {
    if (temp != NULL) {
        inorder(temp->lchild);
        printf("%d", temp->data);
        inorder(temp->rchild);
    }
}
/*  This function displays the tree in preorder fashion  */
void preorder(node *temp) {
    if (temp != NULL) {
        printf("%d", temp->data);
        preorder(temp->lchild);
        preorder(temp->rchild);
    }
}
 /*  This function displays the tree in postorder fashion  */
void postorder(node *temp) {
    if (temp != NULL) {
        postorder(temp->lchild);
        postorder(temp->rchild);
        printf("%d",  temp->data);
    }
}
```

OUTPUT:



```
2.Search
3.Recursive Traversals
4.Exit
Enter your choice :1

Enter The Element 33

Want To enter More Elements?(y/n)
Enter The Element 1

Want To enter More Elements?(y/n)
1.Create
2.Search
3.Recursive Traversals
4.Exit
Enter your choice :3

The Inorder display : 133
The Preorder display : 331
The Postorder display : 133
1.Create
2.Search
3.Recursive Traversals
4.Exit
Enter your choice :_
```

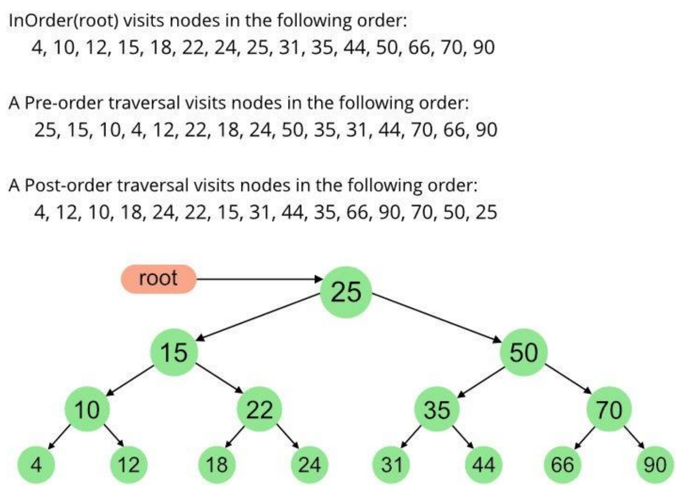
9. Write a program to implement the tree traversal methods using recursion.

/* C Program to Traverse the Tree Recursively */

```
#include <stdio.h>
#include <stdlib.h>
/* A binary tree node has data, pointer to left child and a pointer to right child */
struct node {
        int data;
        struct node* left;
        struct node* right;
};
/* Helper function that allocates a new node with the given data and NULL left and right
pointers. */
struct node* newNode(int data)
{
        struct node* node
                = (struct node*)malloc(sizeof(struct node));
        node->data = data;
        node->left = NULL;
        node->right = NULL;
        return (node);
}
/* Given a binary tree, print its nodes according to the "bottom-up" postorder traversal. */
void printPostorder(struct node* node)
{
        if (node == NULL)
                return;
        // first recur on left subtree
        printPostorder(node->left);
        // then recur on right subtree
        printPostorder(node->right);
        // now deal with the node
        printf("%d ", node->data);
}
/* Given a binary tree, print its nodes in inorder*/
void printInorder(struct node* node)
{
        if (node == NULL)
                return;
        /* first recur on left child */
        printInorder(node->left);
        /* then print the data of node */
        printf("%d ", node->data);
        /* now recur on right child */
        printInorder(node->right);
}
/* Given a binary tree, print its nodes in preorder*/
void printPreorder(struct node* node)
{
        if (node == NULL)
                return;
        /* first print data of node */
        printf("%d ", node->data);
        /* then recur on left sutree */
        printPreorder(node->left);
        /* now recur on right subtree */
        printPreorder(node->right);
}
int main()
{
        struct node* root = newNode(1);
        root->left = newNode(2);
        root->right = newNode(3);
        root->left->left = newNode(4);
        root->left->right = newNode(5);
        printf("\nPreorder traversal of binary tree is \n");
        printPreorder(root);
        printf("\nInorder traversal of binary tree is \n");
        printInorder(root);
        printf("\nPostorder traversal of binary tree is \n");
        printPostorder(root);
        getchar();
        return 0;
}
```

OUTPUT:

```
Preorder traversal of binary tree is
1 2 4 5 3
Inorder traversal of binary tree is
4 2 5 1 3
Postorder traversal of binary tree is
4 5 2 3 1
```



InOrder(root) visits nodes in the following order:
4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 66, 70, 90

A Pre-order traversal visits nodes in the following order:
25, 15, 10, 4, 12, 22, 18, 24, 50, 35, 31, 44, 70, 66, 90

A Post-order traversal visits nodes in the following order:
4, 12, 10, 18, 24, 22, 15, 31, 44, 35, 66, 90, 70, 50, 25

10. Write a program that implements the following sorting methods to sort a given list of integers in ascending order

I) Heap Sort ii) Merge Sort

I) <u>Heap Sort</u>

```
#include<stdio.h>
int temp;

void heapify(int arr[], int size, int i)
{
int largest = i;
int left = 2*i + 1;
int right = 2*i + 2;

if (left < size && arr[left] >arr[largest])
largest = left;

if (right < size && arr[right] > arr[largest])
largest = right;

if (largest != i)
{
temp = arr[i];
   arr[i]= arr[largest];
    arr[largest] = temp;
heapify(arr, size, largest);
}
}

void heapSort(int arr[], int size)
{
int i;
for (i = size / 2 - 1; i >= 0; i--)
heapify(arr, size, i);
for (i=size-1; i>=0; i--)
{
temp = arr[0];
   arr[0]= arr[i];
   arr[i] = temp;
heapify(arr, i, 0);
}
}

void main()
{
int arr[] = {1, 10, 2, 3, 4, 1, 2, 100,23, 2};
int i;
int size = sizeof(arr)/sizeof(arr[0]);
heapSort(arr, size);

printf("printing sorted elements\n");
for (i=0; i<size; ++i)
printf("%d\n",arr[i]);
}
```

OUTPUT:

```
Printing sorted elements

1
1
2
2
2
3
4
10
23
100
```

II. Merge Sort

/* C program for Merge Sort */

```
#include <stdio.h>
#include <stdlib.h>
// Merges two subarrays of arr[].
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
void merge(int arr[], int l, int m, int r)
{
        int i, j, k;
        int n1 = m - l + 1;
        int n2 = r - m;
        /* create temp arrays */
        int L[n1], R[n2];
        /* Copy data to temp arrays L[] and R[] */
        for (i = 0; i < n1; i++)
                L[i] = arr[l + i];
        for (j = 0; j < n2; j++)
                R[j] = arr[m + 1 + j];
        /* Merge the temp arrays back into arr[l..r]*/
        i = 0; // Initial index of first subarray
        j = 0; // Initial index of second subarray
        k = l; // Initial index of merged subarray
        while (i < n1 && j < n2) {
                if (L[i] <= R[j]) {
                        arr[k] = L[i];
                        i++;
                }
                else {
                        arr[k] = R[j];
                        j++;
                }
                k++;
        }
        /* Copy the remaining elements of L[], if there
        are any */
        while (i < n1) {
                arr[k] = L[i];
                i++;
                k++;
        }
        /* Copy the remaining elements of R[], if there
        are any */
        while (j < n2) {
                arr[k] = R[j];
                j++;
                k++;
        }
}
```

```
/* l is for left index and r is right index of the
sub-array of arr to be sorted */
void mergeSort(int arr[], int l, int r)
{
        if (l < r) {
                // Same as (l+r)/2, but avoids overflow for
                // large l and h
                int m = l + (r - l) / 2;
                // Sort first and second halves
                mergeSort(arr, l, m);
                mergeSort(arr, m + 1, r);
                merge(arr, l, m, r);
        }
}

/* UTILITY FUNCTIONS */
/* Function to print an array */
void printArray(int A[], int size)
{
        int i;
        for (i = 0; i < size; i++)
                printf("%d ", A[i]);
        printf("\n");
}
int main()
{
        int arr[] = { 12, 11, 13, 5, 6, 7 };
        int arr_size = sizeof(arr) / sizeof(arr[0]);
        printf("Given array is \n");
        printArray(arr, arr_size);
        mergeSort(arr, 0, arr_size - 1);
        printf("\nSorted array is \n");
        printArray(arr, arr_size);
        return 0;
}
```

OUTPUT:

```
Given array is
12 11 13 5 6 7
Sorted array is
5 6 7 11 12 13
```

11. Write a program to implement the graph traversal methods

A) Depth First Traversal

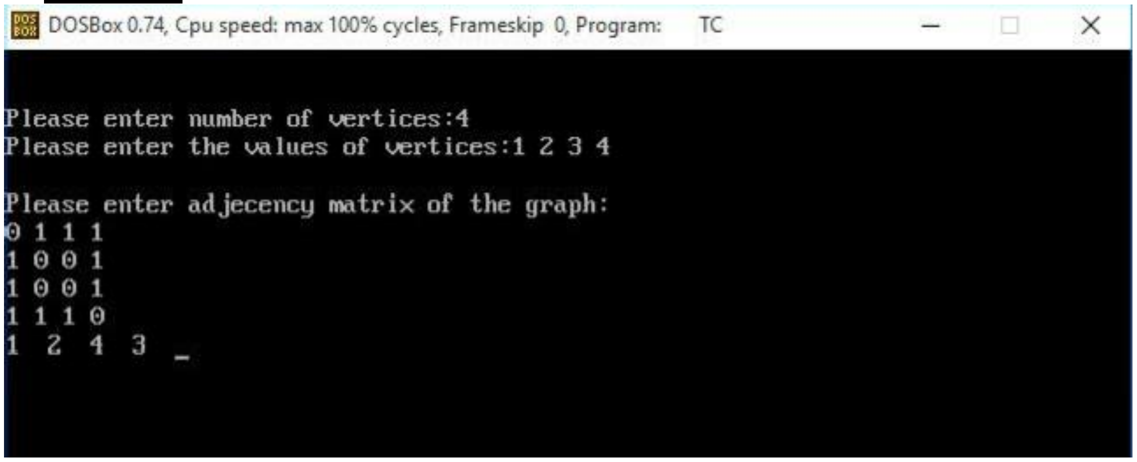
B) Breadth First Traversal

Program:

A) Depth First Traversal

```
#include  <stdio.h>
#include<conio.h>
void dfs(int);
int g[10][10],visited[10],n,vertex[10];
void main() {
int i,j;
clrscr();
printf("\n\nPlease enter number of vertices:");
scanf("%d",&n);
printf("Please enter the values of vertices:");
for(i=0;i<n;i++)
scanf("%d",&vertex[i]);
printf("\nPlease enter adjecency matrix of the graph:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&g[i][j]);
for(i=0;i<n;i++)
visited[i]=0;
dfs(0);
getch();
}
void dfs(int i) {
int j;
printf("%d ",vertex[i]);
visited[i]=1;
for(j=0;j<n;j++)
if(!visited[j]&&g[i][j]==1)
dfs(j);
}
```

OUTPUT:

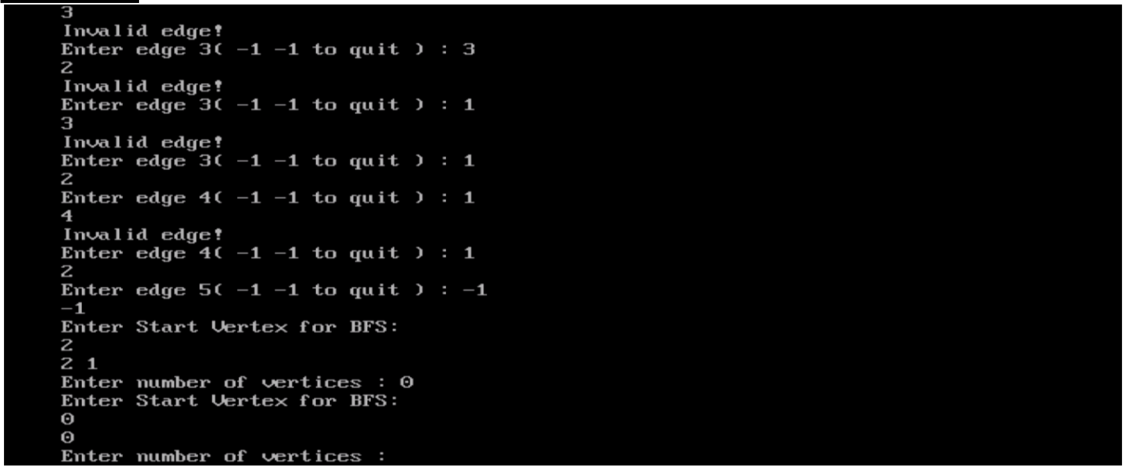

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Please enter number of vertices:4
Please enter the values of vertices:1 2 3 4

Please enter adjecency matrix of the graph:
0 1 1 1
1 0 0 1
1 0 0 1
1 1 1 0
1  2  4  3 _
```

B) Breadth First Traversal

```
#include<stdio.h>
#include<conio.h>
int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;
void bfs(int v) {
        for (i=1;i<=n;i++)
          if(a[v][i] && !visited[i])
          q[++r]=i;
        if(f<=r) {
                visited[q[f]]=1;
                bfs(q[f++]);
        }
}
void main() {
        int v;
        clrscr();
        printf("\n Enter the number of vertices:");
        scanf("%d",&n);
        for (i=1;i<=n;i++) {
                q[i]=0;
                visited[i]=0;
        }
        printf("\n Enter graph data in matrix form:\n");
        for (i=1;i<=n;i++)
          for (j=1;j<=n;j++)
           scanf("%d",&a[i][j]);
        printf("\n Enter the starting vertex:");
        scanf("%d",&v);
        bfs(v);
        printf("\n The node which are reachable are:\n");
        for (i=1;i<=n;i++)
          if(visited[i])
           printf("%d\t",i);  else
           printf("\n Bfs is not possible");
        getch();
} }
```

OUTPUT:

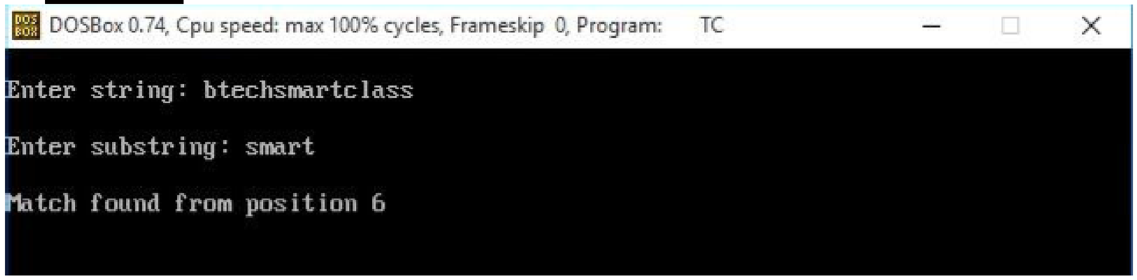


```
3
Invalid edge!
Enter edge 3( -1 -1 to quit ) : 3
2
Invalid edge!
Enter edge 3( -1 -1 to quit ) : 1
3
Invalid edge!
Enter edge 3( -1 -1 to quit ) : 1
2
Enter edge 4( -1 -1 to quit ) : 1
4
Invalid edge!
Enter edge 4( -1 -1 to quit ) : 1
2
Enter edge 5( -1 -1 to quit ) : -1
-1
Enter Start Vertex for BFS:
2
2 1
Enter number of vertices : 0
Enter Start Vertex for BFS:
0
0
Enter number of vertices :
```

12. Write Program for implementing Knuth-Morris-Pratt Pattern matching algorithm.

```
#include  <stdio.h>
#include  <conio.h>
#include  <string.h>
#include  <ctype.h>
void main()
{
char string[100], matchcase[20], c;
int i = 0, j = 0, index;
clrscr();
/*Reading string*/
printf("\nEnter string: ");
scanf("%s",string);
i = strlen(string);
string[i - 1] = '\0';
/* Reading pattern to be search*/
printf("\nEnter substring: ");
scanf("%s",matchcase);
i = strlen(matchcase);
matchcase[i - 1] = '\0';
for (i = 0; i < strlen(string) - strlen(matchcase) + 1; i++)
{
index = i;
if (string[i] == matchcase[j])
{
do
{
i++;        j++;
} while(j != strlen(matchcase) && string[i] == matchcase[j]);
if (j == strlen(matchcase))
{
printf("\nMatch found from position %d\n", index + 1);
goto end;
}        else
{
i = index + 1;
j = 0;
}  }    }
printf("\nNo substring match found in the string.\n");
end: getch();
}
```

OUTPUT:


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

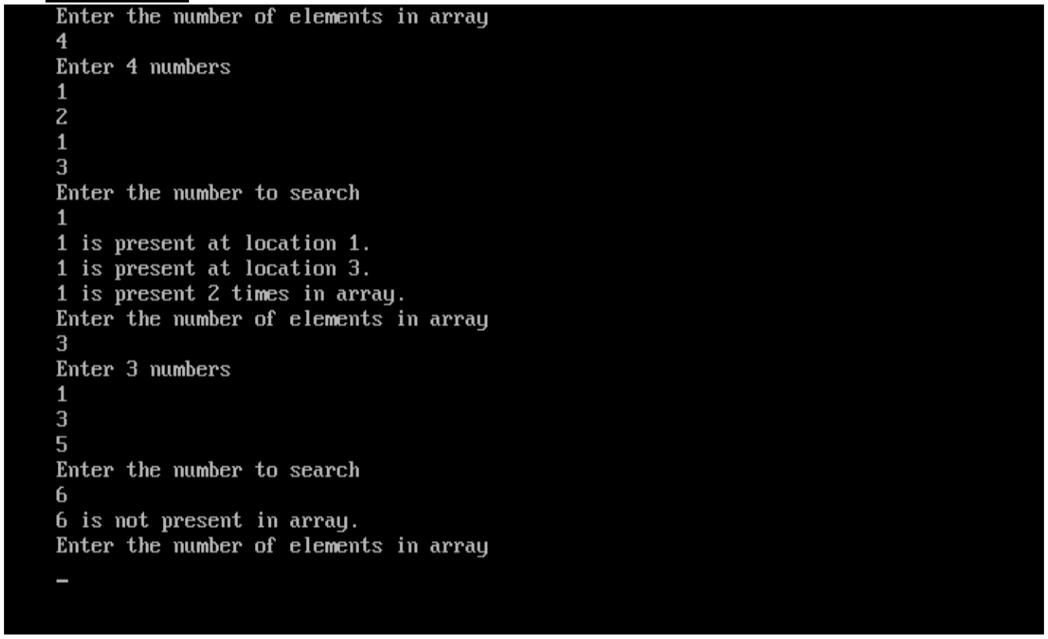
```
Enter string: btechsmartclass

Enter substring: smart

Match found from position 6
```

Additional Programs

a) Linear Search

```
#include <stdio.h>
 int main()
{
   int array[100], search, c, n, count = 0;
   printf("Enter the number of elements in array\n");
   scanf("%d", &n);
    printf("Enter %d numbers\n", n);
    for ( c = 0 ; c < n ; c++ )
    scanf("%d", &array[c]);
    printf("Enter the number to search\n");
   scanf("%d", &search);
    for (c = 0; c < n; c++) {
      if (array[c] == search) {
         printf("%d is present at location %d.\n", search, c+1);
             count++;
      }
   }
   if (count == 0)
      printf("%d is not present in array.\n", search);
   else
      printf("%d is present %d times in array.\n", search, count);

   return 0;
}
```

<u>OUTPUT:</u> —

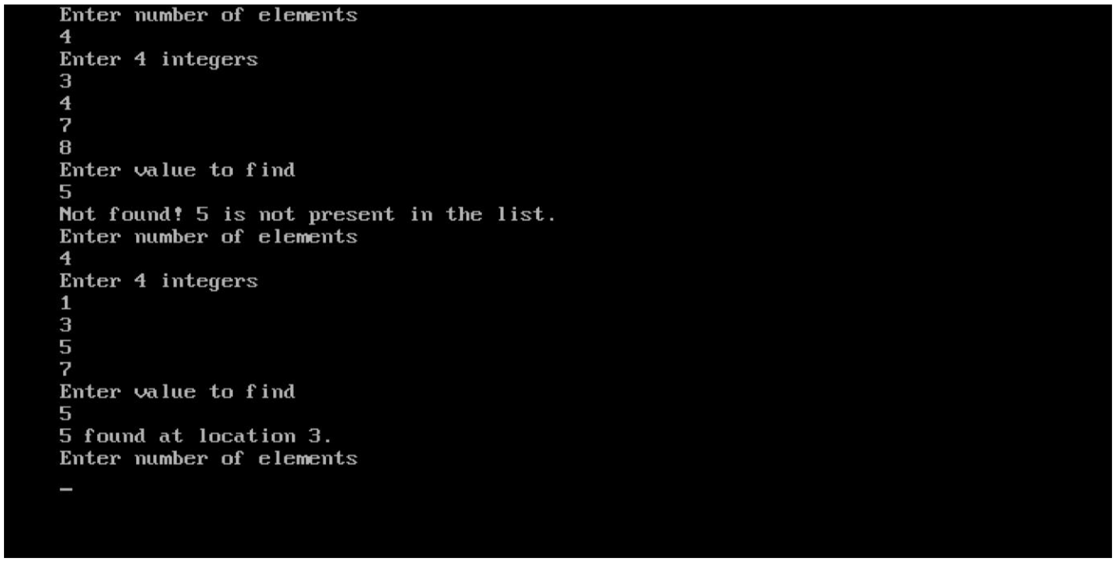


```
Enter the number of elements in array
4
Enter 4 numbers
1
2
1
3
Enter the number to search
1
1 is present at location 1.
1 is present at location 3.
1 is present 2 times in array.
Enter the number of elements in array
3
Enter 3 numbers
1
3
5
Enter the number to search
6
6 is not present in array.
Enter the number of elements in array
_
```

B) Binary Search

```
#include <stdio.h>
 int main()
{
   int c, first, last, middle, n, search, array[100];
   printf("Enter number of elements\n");
   scanf("%d",&n);
    printf("Enter %d integers\n", n);
   for (c = 0; c < n; c++)
   scanf("%d",&array[c]);
   printf("Enter value to find\n");
   scanf("%d", &search);
    first = 0;
   last = n - 1;
   middle = (first+last)/2;
    while (first <= last) {
      if (array[middle] < search)
         first = middle + 1;
      else if (array[middle] == search) {
         printf("%d found at location %d.\n", search, middle+1);
         break;
      }
      else
         last = middle - 1;
       middle = (first + last)/2;
   }
   if (first > last)
      printf("Not found! %d is not present in the list.\n", search);
    return 0;
}
```

<u>OUTPUT:</u>

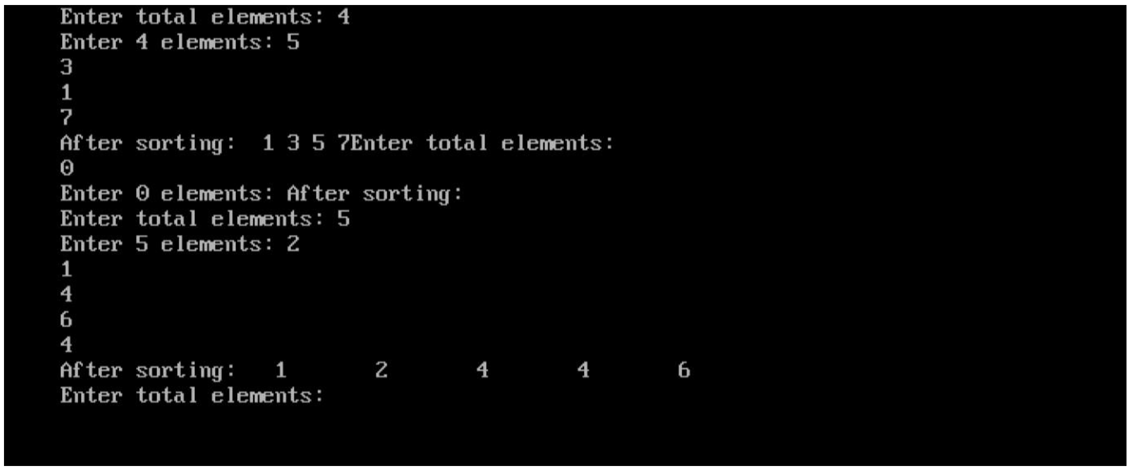


```
Enter number of elements
4
Enter 4 integers
3
4
7
8
Enter value to find
5
Not found! 5 is not present in the list.
Enter number of elements
4
Enter 4 integers
1
3
5
7
Enter value to find
5
5 found at location 3.
Enter number of elements
_
```

A) Insertion Sort

```
#include<stdio.h>
int main(){

  int i,j,s,temp,a[20];

  printf("Enter total 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("After sorting: ");
  for(i=0;i<s;i++)
      printf(" %d",a[i]);

  return 0;
}
```

OUTPUT:

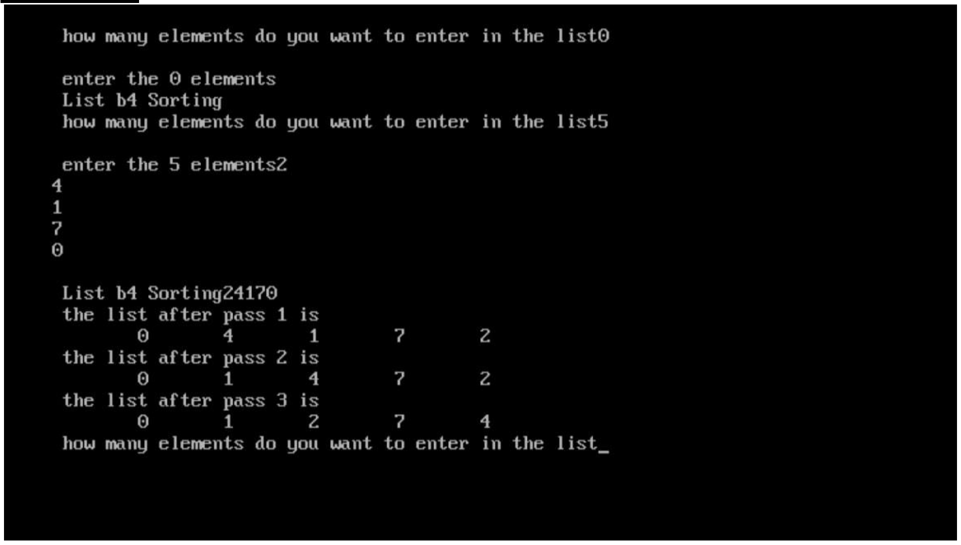


```
Enter total elements: 4
Enter 4 elements: 5
3
1
7
After sorting:  1 3 5 7Enter total elements:
0
Enter 0 elements: After sorting:
Enter total elements: 5
Enter 5 elements: 2
1
4
6
4
After sorting:   1       2       4       4       6
Enter total elements:
```

B) Selection Sort

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,A[10],n;
int j,min,temp,k;
//clrscr();
printf("\n how many elements do you want to enter in the list");
scanf("%d",&n);
printf("\n enter the %d elements",n);
for(i=0;i<n;i++)
scanf("%d",&A[i]);
printf("\n List b4 Sorting");
for(i=0;i<n;i++)
printf("%d",A[i]);
for(i=0;i<n-2;i++)
{
min=i;
for(j=i+1;j<=n-1;j++)
{
if(A[j]<A[min])
min=j;
}
temp=A[i];
A[i]=A[min];
A[min]=temp;
printf("\n the list after pass %d is \n",i+1);
for(k=0;k<n;k++)
printf(" \n%d-",A[k]);
}
}
```

OUTPUT:



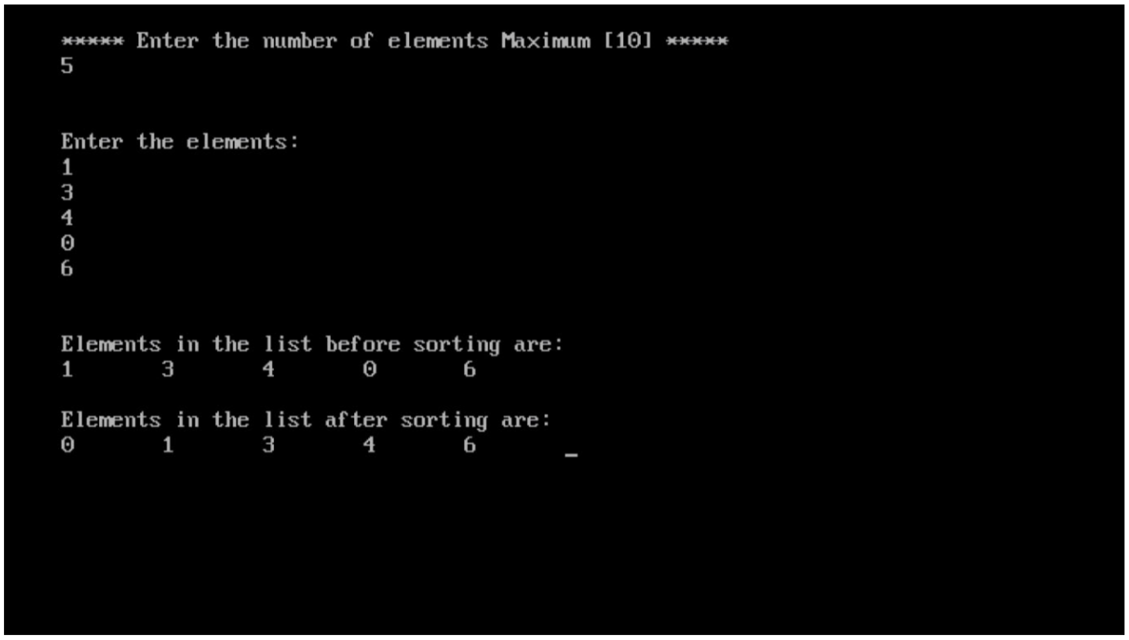
```
 how many elements do you want to enter in the list0

 enter the 0 elements
 List b4 Sorting
 how many elements do you want to enter in the list5

 enter the 5 elements2
4
1
7
0

 List b4 Sorting24170
 the list after pass 1 is
        0       4       1       7       2
 the list after pass 2 is
        0       1       4       7       2
 the list after pass 3 is
        0       1       2       7       4
 how many elements do you want to enter in the list_
```

a) Quick Sort

```
#include <stdio.h>
#define MAX 10
void swap(int *m,int *n)
{
  int temp;
  temp = *m;
  *m = *n;
  *n = temp;
}
int get_key_position(int x,int y )
{
  return((x+y) /2);
}
// Function for Quick Sort
void quicksort(int list[],int m,int n)
{
  int key,i,j,k;
  if( m < n)
  {
    k = get_key_position(m,n);
    swap(&list[m],&list[k]);
    key = list[m];
    i = m+1;
    j = n;
    while(i <= j)
    {
      while((i <= n) && (list[i] <= key))
           i++;
      while((j >= m) && (list[j] > key))
                j--;
         if( i < j)
                swap(&list[i],&list[j]);
    }
    swap(&list[m],&list[j]);
    quicksort(list,m,j-1);
    quicksort(list,j+1,n);
  }
}
// Function to read the data
void read_data(int list[],int n)
{
  int j;
  printf("\n\nEnter the elements:\n");
  for(j=0;j<n;j++)
     scanf("%d",&list[j]);
}
// Function to print the data
```

```
void print_data(int list[],int n)
{
  int j;
  for(j=0;j<n;j++)
      printf("%d\t",list[j]);
}
void main()
{
  int list[MAX], num;
  clrscr();
  printf("\n***** Enter the number of elements Maximum [10] *****\n");
  scanf("%d",&num);
  read_data(list,num);
  printf("\n\nElements in the list before sorting are:\n");
  print_data(list,num);
  quicksort(list,0,num-1);
  printf("\n\nElements in the list after sorting are:\n");
  print_data(list,num);
  getch();
}
```

OUTPUT:



```
***** Enter the number of elements Maximum [10] *****
5


Enter the elements:
1
3
4
0
6


Elements in the list before sorting are:
1       3       4       0       6

Elements in the list after sorting are:
0       1       3       4       6       _
```