IMPLEMENTATION OF LIST

EXERCISE 1.1 [List ADT Using Array]

Program code :

```
#include<stdio.h>
#include <stdlib.h>
#include<conio.h>
void create();
void insert();
void deletion();
void search();
void display();
int *A , n;
int flag = 1;
void main()
{
   int ch;

   printf("Create Your array !\n\n");
   create();
   while(flag)
   {
      printf(" 1.INSERT \n 2.DELETE \n 3.SEARCH \n 4.DISPLAY \n 5.EXIT \n");
      printf("\n ENTER THE CHOICE :");

      scanf("%d",&ch);
      switch(ch)
      {
         case 1 : insert();
            break ;
```

```
        case 2 : delete ();
           break ;
        case 3 : search();
           break ;
        case 4 : display();
           break ;
        case 5 :
           flag=0;
           printf("Thank you !");
           break;
        default : printf("IT IS INVALID NUMBER :");
        break ;
      }
   }
}

void create()
{
     printf(" ------>  CREATE  <------");
     printf("\nEnter the no. of elements:");
     scanf("%d", &n);
     A = (int*)malloc(n*sizeof(int));
    for (int i=0;i<n;i++)
     {
        printf("\n Enter the value :");
        scanf(" %d",&A[i]);
     }

     printf("\n");
```

```
    printf("\n ---------------------------------- \n");
}
void display()
{
      printf("------> DISPLAY <-------\n");
      printf("The original array elements are :\n");
      for(int i = 0; i<n; i++)
      {
         printf("A[%d] = %d \n", i, A[i]);
      }
      printf("\n");
      printf("\n ---------------------------------- \n");
}
void insert()
{
    printf(" -----> INSERT <-------");

   printf("\n Enter the position to insert :");
   int p;
   scanf("%d",&p);
   if(p>n)
   {
     printf("\n Invalid position");
   }
   else
   {
       printf("\n Enter the value :");
       int value;
       scanf(" %d",&value);
```

```
        for(int i = n-1; i >= p; i--)
          {
            A[i+1] = A[i];
          }
        A[p] = value;
        n++;
    }
    printf("\n THE LIST AFTER INSERT :\n");
    for(int i=0 ; i<n ; i++)
    {
        printf("\t%d", A[i]);
    }
    printf("\n");
    printf("\n ---------------------------------- \n");
}
void delete()
{
    printf("-----> DELETE <------\n");
    printf("\n Enter the position to Delete :");
    int p;
    scanf("%d",&p);
    if(p>n)
    {
      printf("\n Invalid position");
    }
    else
    {
    for(int i = p+1; i <n; i++)
        {
```

```
        A[i-1] = A[i];
      }
    n--;
    }
    printf("\n THE ELEMENT AFTER DELETE :\n");
    for(int i=0 ; i<n ; i++)
    {
       printf("\t%d", A[i]);
    }
   printf("\n");
   printf("\n ---------------------------------- \n");
}

void search()
{
   printf("-----> SEARCH <------");
   printf("\n Enter the value to search :");
   int value;
   scanf("%d",&value);
   int fg=1;
   for(int i=0;i<n;i++)
    {
       if(A[i] == value )
       {
          printf(" Found element %d at index of %d \n", value, i);
          fg=0;
          break;
       }
   }
```

```
    if (fg)
    {
      printf("\n Element not found!");
    }
    printf("\n");
    printf("\n ---------------------------------- \n");
}
```

Program Output :

```
Create Your array !

 ------>  CREATE  <------
Enter the no. of elements:5

 Enter the value :1

 Enter the value :2

 Enter the value :3

 Enter the value :4

 Enter the value :5


 ----------------------------------
 1.INSERT
 2.DELETE
 3.SEARCH
 4.DISPLAY
 5.EXIT

 ENTER THE CHOICE :1
 -----> INSERT <-------
 Enter the position to insert :4

 Enter the value :9

 THE LIST AFTER INSERT :
        1       2       3       4       9       5

 ----------------------------------
 1.INSERT
 2.DELETE
 3.SEARCH
 4.DISPLAY
 5.EXIT

 ENTER THE CHOICE :2
-----> DELETE <------

 Enter the position to Delete :5

 THE ELEMENT AFTER DELETE :
        1       2       3       4       9
```

```
 ---------------------------------
 1.INSERT
 2.DELETE
 3.SEARCH
 4.DISPLAY
 5.EXIT

 ENTER THE CHOICE :4
------> DISPLAY <-------
The original array elements are :
A[0] = 1
A[1] = 2
A[2] = 3
A[3] = 4
A[4] = 9


 ---------------------------------
 1.INSERT
 2.DELETE
 3.SEARCH
 4.DISPLAY
 5.EXIT

 ENTER THE CHOICE :5
Thank you !
Process returned 0 (0x0)   execution time : 159.972 s
Press any key to continue.
```

EXERCISE 1.2 [Singly Linked List]

Program code:-

```
#include <stdio.h>
#include <stdlib.h>
int size=0;

struct node {
  int data;
  struct node* next;
}*head;
void SEARCH(){
    int item;
    printf("Enter the value to be search: ");
    scanf("%d",&item);
    struct node* ptr = head;
    int i =0 , flag=0;
    if(ptr==NULL){
        printf("\nEmpty List\n");
    }
    else{
        while(ptr!=NULL){
            if(ptr->data==item){
                printf("Item found at location %d",i+1);
                flag=0;
                break;
            }
            else{
                flag=1;
            }
```

```
            i++;
            ptr=ptr->next;
        }
        if(flag){
            printf("Item not found\n");
        }
    }
}
void del_b(){
    struct node* ptr=head;
    head =ptr->next;
    size--;
}
void del_e(){
    struct node * pos = head,* temp;

    while (pos->next != NULL) {
        temp = pos;
        pos = pos->next;
  }
  temp->next=NULL;
  size--;

}
void del_m(){
    struct node* ptr=head;
    int i,n=0;
    printf("\nEnter the position where to delete from the range (1 - %d) : ",size);
    scanf(" %d",&n);
```

```
   if  ((n<2) ||(n>size)){
   printf("\nPlease enter the value in the range (1 - %d) : ",size);
   del_m();
   }
   else{
      int t=0;
   struct node * pos = head,*temp;
   while ( t<n-1 ) {
      temp=pos;
      pos = pos->next;
      t++;
      if(ptr==NULL)
         return;
  }
  temp->next=pos->next;

}
size--;
}
void in_b(){
   int i;
   struct node* ptr = (struct node*)malloc(sizeof(struct node));
   printf("\n\nEnter item is to be inserted at the beginning : ");
   scanf(" %d",&i);
   ptr->data = i;
   ptr->next = head;
   head=ptr;
   size++;
```

```
}

void in_e(){
   int ine;
   struct node* ptre = (struct node*)malloc(sizeof(struct node));

   printf("\n\nEnter item is to be inserted at the end : ");
   scanf(" %d",&ine);
   ptre->data = ine;
   ptre->next=NULL;
   struct node * pos = head;

   if (head== NULL) {
  head = ptre;
  return;
  }
   while (pos->next != NULL) {
      pos = pos->next;
  }
  pos->next=ptre;
  size++;

  return;

}

void in_m(){
   int i,n=0;
   struct node* ptr = (struct node*)malloc(sizeof(struct node));
```

```
   printf("\n\nEnter item is to be inserted at the middle : ");
   scanf(" %d",&i);
   ptr->data = i;
   printf("\nEnter the position where to insert from the range (1 - %d) : ",size);
   scanf(" %d",&n);
   if  ((n<2) ||(n>size)){
   printf("\nPlease enter the value in the range (1 - %d) : ",size-1);
   in_m();
   }
   else{
      int t=1;
   struct node * pos = head;
   while ( t<n-1 ) {
      pos = pos->next;
      t++;
  }

  ptr->next=pos->next;
  pos->next=ptr;
   }
   size++;
}

void INSERT(){
   int item,choice;
   if(head==NULL){
      puts("\n\tYou can only insert at the beginning");
      struct node* ptr = (struct node*)malloc(sizeof(struct node));
```

```
    printf("\nEnter item is to be inserted : ");
    scanf(" %d",&item);
    ptr->data=item;
    ptr->next=head;
    head=ptr;
    size++;
}
else{
    e1:
        printf("\nTypes of Insertion Operations :\n");
        printf("\n\t1.Inserting - Beginning \n\t2.Inserting - Between \n\t3.Insert - Last\n");
        printf("Enter your choise: ");
        scanf(" %d",&choice);
        switch (choice){
        case 1:
            in_b();
            break;
        case 2:
            if(size<2){
                puts("\nYou cannot insert element in the middle since the size is very low");
                break;
            }
            else{
            in_m();
            break;
            }
        case 3:
            in_e();
            break;
```

```
            default:
               puts("Enter the valid choice");
               goto e1;
               break;
         }
      }
   }
   void DELETE(){
      int item,choice;
      if(head==NULL){
         puts("THERE IS NO NODE IS PRESENT TO DELETE");
      }
      else if(size==1){
         printf("There is only one node in the list ");
         head=NULL;

      }
      else{
         e2:
            printf("\nTypes of Insertion Operations :\n");
            printf("\n\t1.Delete - Beginning \n\t2.Delete - Between \n\t3.Delete - Last\n");
            printf("Enter your choise: ");
            scanf(" %d",&choice);
            switch (choice){
            case 1:
               del_b();
               break;
            case 2:
```

```
            del_m();
            break;

        case 3:
            del_e();
            break;
        default:
            puts("Enter the valid choice");
            goto e2;
            break;
        }
    }
}
void DISPLAY(){
    struct node * pos = head;
    printf("\n [%d]",head);
    while (pos != NULL) {
        printf(" -> [%d | %d]", pos->data,pos->next);
        pos = pos->next;
  }
}

int main(){

    head = NULL;
    int op =1,i=0;

    while(op){
        if (size ==0)
```

```
        head=NULL;
      int choose;
      m1:

puts("\n**********************************************************************
***********");
        puts("\n\t\t\t\tMENU: \n\n\t\t1.INSERT \n\t\t2.DELETE \n\t\t3.SEARCH
\n\t\t4.DISPLAY \n\t\t5.EXIT");

puts("\n**********************************************************************
***********");
        printf("\n\tSELECT YOUR OPTION FROM MENU :");
        scanf("%d",&choose);
        switch(choose){
        case 1:
          INSERT();
          break;
        case 2:
          DELETE();
          break;
        case 3:
          SEARCH();
          break;
        case 4:
          DISPLAY();
          break;
        case 5:
          op=0;
          break;
        default:
```

```
            puts("Please Enter the valid choice : ");
            goto m1;
            break;
        }
    }
}
```

Program Output :

```
********************************************************************************

                                MENU:

                1.INSERT
                2.DELETE
                3.SEARCH
                4.DISPLAY
                5.EXIT

********************************************************************************

        SELECT YOUR OPTION FROM MENU :1

        You can only insert at the beginning

Enter item is to be inserted : 3

********************************************************************************

                                MENU:

                1.INSERT
                2.DELETE
                3.SEARCH
                4.DISPLAY
                5.EXIT

********************************************************************************

        SELECT YOUR OPTION FROM MENU :1

Types of Insertion Operations :

        1.Inserting - Beginning
        2.Inserting - Between
        3.Insert - Last
Enter your choise: 1


Enter item is to be inserted at the beginning : 2

********************************************************************************

                                MENU:

                1.INSERT
                2.DELETE
                3.SEARCH
                4.DISPLAY
                5.EXIT
```

```
       SELECT YOUR OPTION FROM MENU :1

Types of Insertion Operations :

       1.Inserting - Beginning
       2.Inserting - Between
       3.Insert - Last
Enter your choise: 1


Enter item is to be inserted at the beginning : 1

********************************************************************************

                              MENU:

               1.INSERT
               2.DELETE
               3.SEARCH
               4.DISPLAY
               5.EXIT

********************************************************************************

       SELECT YOUR OPTION FROM MENU :4

 [7629728] -> [1 | 7629696] -> [2 | 7629664] -> [3 | 0]
********************************************************************************

                              MENU:

               1.INSERT
               2.DELETE
               3.SEARCH
               4.DISPLAY
               5.EXIT

********************************************************************************

       SELECT YOUR OPTION FROM MENU :2

Types of Insertion Operations :

       1.Delete - Beginning
       2.Delete - Between
       3.Delete - Last
Enter your choise: 3

********************************************************************************

                              MENU:
```

```
                                MENU:

                1.INSERT
                2.DELETE
                3.SEARCH
                4.DISPLAY
                5.EXIT

*********************************************************************************

        SELECT YOUR OPTION FROM MENU :1

Types of Insertion Operations :

        1.Inserting - Beginning
        2.Inserting - Between
        3.Insert - Last
Enter your choise: 3


Enter item is to be inserted at the end : 5

*********************************************************************************

                                MENU:

                1.INSERT
                2.DELETE
                3.SEARCH
                4.DISPLAY
                5.EXIT

*********************************************************************************

        SELECT YOUR OPTION FROM MENU :2

Types of Insertion Operations :

        1.Delete - Beginning
        2.Delete - Between
        3.Delete - Last
Enter your choise: 1

*********************************************************************************

                                MENU:

                1.INSERT
                2.DELETE
                3.SEARCH
                4.DISPLAY
                5.EXIT
```

```
                                MENU:

                1.INSERT
                2.DELETE
                3.SEARCH
                4.DISPLAY
                5.EXIT

********************************************************************************

        SELECT YOUR OPTION FROM MENU :4

 [7629696] -> [2 | 7629760] -> [5 | 0]
********************************************************************************

                                MENU:

                1.INSERT
                2.DELETE
                3.SEARCH
                4.DISPLAY
                5.EXIT

********************************************************************************

        SELECT YOUR OPTION FROM MENU :3
Enter the value to be search: 5
Item found at location 2
********************************************************************************

                                MENU:

                1.INSERT
                2.DELETE
                3.SEARCH
                4.DISPLAY
                5.EXIT

********************************************************************************

        SELECT YOUR OPTION FROM MENU :5

Process returned 0 (0x0)   execution time : 82.906 s
Press any key to continue.
```

EXERCISE 1.3 [Doubly Linked List]

Program Code:

```
#include <stdio.h>
#include <stdlib.h>
int size=0;

struct node {
  struct node* prev;
  int data;
  struct node* next;
}*head;
void SEARCH(){
    int item;
    printf("Enter the value to be search: ");
    scanf("%d",&item);
    struct node* ptr = head;
    int i =0 , flag=0;
    if(ptr==NULL){
        printf("\nEmpty List\n");
    }
    else{
        while(ptr!=NULL){
            if(ptr->data==item){
                printf("Item found at location %d",i+1);
                flag=0;
                break;
            }
            else{
                flag=1;
```

```
            }
            i++;
            ptr=ptr->next;
        }
        if(flag){
            printf("Item not found\n");
        }
    }
}
void in_b(){
    int i;
    struct node* ptr = (struct node*)malloc(sizeof(struct node));
    struct node* temp=head;
    printf("\n\nEnter item is to be inserted at the beginning : ");
    scanf(" %d",&i);
    ptr->prev=NULL;
    ptr->data = i;
    ptr->next = head;
    head=ptr;
    temp->prev=ptr;
    size++;

}
void in_e(){
    int ine;
    struct node* ptre = (struct node*)malloc(sizeof(struct node));

    printf("\n\nEnter item is to be inserted at the end : ");
    scanf(" %d",&ine);
```

```
   ptre->data = ine;
   ptre->next=NULL;
   struct node * pos = head;

   if (head== NULL) {
  head = ptre;
  return;
  }
   while (pos->next != NULL) {
       pos = pos->next;
  }
  pos->next=ptre;
  ptre->prev=pos;
  size++;

  return;

}
void in_m(){
   int i,n=0;
   struct node* ptr = (struct node*)malloc(sizeof(struct node));

   printf("\n\nEnter item is to be inserted at the middle : ");
   scanf(" %d",&i);
   ptr->data = i;
   printf("\nEnter the position where to insert from the range (1 - %d) : ",size);
   scanf(" %d",&n);
   if  ((n<2) ||(n>size)){
   printf("\nPlease enter the value in the range (1 - %d) : ",size-1);
```

```
    in_m();
    }
    else{
        int t=1;
    struct node * pos = head;
    while ( t<n-1 ) {
        pos = pos->next;
        t++;
   }

   ptr->next=pos->next;
   pos->next=ptr;
   ptr->prev=pos;
   ptr=ptr->next;
   ptr->prev=pos->next;
    }
    size++;
}
void del_b(){
    struct node* ptr=head;
    head =ptr->next;
    struct node* tmp=head;
    tmp->prev=NULL;
    size--;
}
void del_e(){
    struct node * pos = head,* temp;

    while (pos->next != NULL) {
```

```
        temp = pos;
        pos = pos->next;
    }
    temp->next=NULL;
    size--;

}
void del_m(){
    struct node* ptr=head;
    int i,n=0;
    printf("\nEnter the position where to delete from the range (1 - %d) : ",size);
    scanf(" %d",&n);
    if  ((n<2) ||(n>size)){
    printf("\nPlease enter the value in the range (1 - %d) : ",size);
    del_m();
    }
    else{
        int t=0;
    struct node * pos = head,*temp;
    while ( t<n-1 ) {
        temp=pos;
        pos = pos->next;
        t++;
        if(ptr==NULL)
            return;
    }
    temp->next=pos->next;
    pos=pos->next;
    pos->prev=temp;
```

```
}
size--;
}
void DELETE(){
   int item,choice;
   if(head==NULL){
      puts("THERE IS NO NODE IS PRESENT TO DELETE");
   }
   else if(size==1){
      printf("There is only one node in the list ");
      head=NULL;

   }
   else{
      e2:
         printf("\nTypes of Deletion Operations :\n");
         printf("\n\t1.Delete - Beginning \n\t2.Delete - Between \n\t3.Delete - Last\n");
         printf("Enter your choise: ");
         scanf(" %d",&choice);
         switch (choice){
         case 1:
            del_b();
            break;
         case 2:

            del_m();
            break;
```

```
            case 3:
               del_e();
               break;
            default:
               puts("Enter the valid choice");
               goto e2;
               break;
            }
      }
}
void INSERT(){
   int item,choice;
   if(head==NULL){
      puts("\n\tYou can only insert at the beginning");
      struct node* ptr = (struct node*)malloc(sizeof(struct node));
      printf("\nEnter item is to be inserted : ");
      scanf(" %d",&item);
      ptr->prev=NULL;
      ptr->data=item;
      ptr->next=head;
      head=ptr;
      size++;
   }
   else{
      e1:
         printf("\nTypes of Insertion Operations :\n");
         printf("\n\t1.Inserting - Beginning \n\t2.Inserting - Between \n\t3.Insert - Last\n");
         printf("Enter your choise: ");
         scanf(" %d",&choice);
```

```
            switch (choice){
            case 1:
               in_b();
               break;
            case 2:
               if(size<2){
                  puts("\nYou cannot insert element in the middle since the size is very low");
                  break;
               }
               else{
               in_m();
               break;
               }
            case 3:
               in_e();
               break;
            default:
               puts("Enter the valid choice");
               goto e1;
               break;
         }
      }
   }
void DISPLAY(){
   struct node * pos = head;
   printf("\n [%d]",head);
   while (pos != NULL) {
      printf(" -> [%d|[%d]|%d]",pos->prev, pos->data,pos->next);
      pos = pos->next;
```

```
 }
}
int main(){

   head = NULL;
   int op =1,i=0;

   while(op){
      if (size ==0)
         head=NULL;
      int choose;
      m1:

puts("\n*****************************************************************
***********");
         puts("\n\t\t\t\tMENU: \n\n\t\t1.INSERT \n\t\t2.DELETE \n\t\t3.SEARCH
\n\t\t4.DISPLAY \n\t\t5.EXIT");

puts("\n*****************************************************************
***********");
         printf("\n\tSELECT YOUR OPTION FROM MENU :");
         scanf("%d",&choose);
         switch(choose){
         case 1:
            INSERT();
            break;
         case 2:
            DELETE();
            break;
         case 3:
```

```
            SEARCH();
            break;
        case 4:
            DISPLAY();
            break;
        case 5:
            op=0;
            break;
        default:
            puts("Please Enter the valid choice : ");
            goto m1;
            break;
      }
   }
}
```

Program Output :

```
********************************************************************************

                                MENU:

                1.INSERT
                2.DELETE
                3.SEARCH
                4.DISPLAY
                5.EXIT

********************************************************************************

        SELECT YOUR OPTION FROM MENU :1

        You can only insert at the beginning

Enter item is to be inserted : 3

********************************************************************************

                                MENU:

                1.INSERT
                2.DELETE
                3.SEARCH
                4.DISPLAY
                5.EXIT

********************************************************************************

        SELECT YOUR OPTION FROM MENU :1

Types of Insertion Operations :

        1.Inserting - Beginning
        2.Inserting - Between
        3.Insert - Last
Enter your choise: 1


Enter item is to be inserted at the beginning : 1
```

```
********************************************************************************

                                MENU:

                1.INSERT
                2.DELETE
                3.SEARCH
                4.DISPLAY
                5.EXIT

********************************************************************************

        SELECT YOUR OPTION FROM MENU :1

Types of Insertion Operations :

        1.Inserting - Beginning
        2.Inserting - Between
        3.Insert - Last
Enter your choise: 2


Enter item is to be inserted at the middle : 2

Enter the position where to insert from the range (1 - 2) : 2

********************************************************************************

                                MENU:

                1.INSERT
                2.DELETE
                3.SEARCH
                4.DISPLAY
                5.EXIT

********************************************************************************

        SELECT YOUR OPTION FROM MENU :1

Types of Insertion Operations :

        1.Inserting - Beginning
        2.Inserting - Between
        3.Insert - Last
Enter your choise: 3


Enter item is to be inserted at the end : 4

********************************************************************************
```

```
********************************************************************************

                                MENU:

                1.INSERT
                2.DELETE
                3.SEARCH
                4.DISPLAY
                5.EXIT

********************************************************************************

        SELECT YOUR OPTION FROM MENU :4

 [6975072] -> [0|[1]|6975104] -> [6975072|[2]|6975040] -> [6975104|[3]|6975136] -> [6975040|[4]|0]
********************************************************************************

                                MENU:

                1.INSERT
                2.DELETE
                3.SEARCH
                4.DISPLAY
                5.EXIT

********************************************************************************

        SELECT YOUR OPTION FROM MENU :2

Types of Deletion Operations :

        1.Delete - Beginning
        2.Delete - Between
        3.Delete - Last
Enter your choise: 1

********************************************************************************

                                MENU:

                1.INSERT
                2.DELETE
                3.SEARCH
                4.DISPLAY
                5.EXIT

********************************************************************************
```

```
********************************************************************************

        SELECT YOUR OPTION FROM MENU :2

Types of Deletion Operations :

        1.Delete - Beginning
        2.Delete - Between
        3.Delete - Last
Enter your choise: 3

********************************************************************************

                                MENU:

                1.INSERT
                2.DELETE
                3.SEARCH
                4.DISPLAY
                5.EXIT

********************************************************************************

        SELECT YOUR OPTION FROM MENU :4

 [6975104] -> [0|[2]|6975040] -> [6975104|[3]|0]
********************************************************************************

                                MENU:

                1.INSERT
                2.DELETE
                3.SEARCH
                4.DISPLAY
                5.EXIT

********************************************************************************

        SELECT YOUR OPTION FROM MENU :3
Enter the value to be search: 3
Item found at location 2
********************************************************************************
```

```
********************************************************************************

                                MENU:

                1.INSERT
                2.DELETE
                3.SEARCH
                4.DISPLAY
                5.EXIT

********************************************************************************

        SELECT YOUR OPTION FROM MENU :5

Process returned 0 (0x0)   execution time : 109.476 s
Press any key to continue.
```

EXERCISE 1.4 [Circular Linked List]

Program Code:

```
//Circular linked list
#include <stdio.h>
#include <stdlib.h>
int size=0;

struct node {
  int data;
  struct node* next;
}*head;
void SEARCH(){
    int item;
    printf("Enter the value to be search: ");
    scanf("%d",&item);
    struct node* ptr = head;
    int i =0 , flag=0;
    if(ptr==NULL){
        printf("\nEmpty List\n");
    }
    else{
        while(ptr!=NULL){
            if(ptr->data==item){
                printf("Item found at location %d",i+1);
                flag=0;
                break;
            }
            else{
                flag=1;
```

```
            }
            i++;
            ptr=ptr->next;
        }
        if(flag){
            printf("Item not found\n");
        }
    }
}
void in_b(){
    int i;
    struct node* ptr = (struct node*)malloc(sizeof(struct node));
    struct node* temp=head;
    printf("\n\nEnter item is to be inserted at the beginning : ");
    scanf(" %d",&i);
    ptr->data = i;
    ptr->next = head;

    struct node * pos = head;
    while (pos->next != head) {
        pos = pos->next;
  }
  head=ptr;
  pos->next=head;
    size++;

}
void in_e(){
    int ine;
```

```
   struct node* ptre = (struct node*)malloc(sizeof(struct node));

   printf("\n\nEnter item is to be inserted at the end : ");
   scanf(" %d",&ine);
   ptre->data = ine;
   ptre->next=head;
   struct node * pos = head;

   if (head== NULL) {
  head = ptre;
  return;
  }
   while (pos->next != head) {
       pos = pos->next;
  }
  pos->next=ptre;
  size++;

  return;

}
void in_m(){
   int i,n=0;
   struct node* ptr = (struct node*)malloc(sizeof(struct node));

   printf("\n\nEnter item is to be inserted at the middle : ");
   scanf(" %d",&i);
   ptr->data = i;
   printf("\nEnter the position where to insert from the range (1 - %d) : ",size);
```

```
   scanf(" %d",&n);
   if  ((n<2) ||(n>size)){
   printf("\nPlease enter the value in the range (1 - %d) : ",size-1);
   in_m();
   }
   else{
      int t=1;
   struct node * pos = head;
   while ( t<n-1 ) {
      pos = pos->next;
      t++;
  }

  ptr->next=pos->next;
  pos->next=ptr;
  ptr=ptr->next;
   }
   size++;
}
void del_b(){
   struct node* ptr=head;

    struct node * pos = head;

   while (pos->next != head) {
      pos = pos->next;
  }
  head =ptr->next;
  pos->next=head;
```

```
    size--;
}
void del_e(){
    struct node * pos = head,* temp;

    while (pos->next != head) {
        temp = pos;
        pos = pos->next;
  }
  temp->next=head;
  size--;

}
void del_m(){
    struct node* ptr=head;
    int i,n=0;
    printf("\nEnter the position where to delete from the range (1 - %d) : ",size);
    scanf(" %d",&n);
    if  ((n<2) ||(n>size)){
    printf("\nPlease enter the value in the range (1 - %d) : ",size);
    del_m();
    }
    else{
        int t=0;
    struct node * pos = head,*temp;
    while ( t<n-1 ) {
        temp=pos;
        pos = pos->next;
        t++;
```

```
        if(ptr==NULL)
            return;
    }
    temp->next=pos->next;
    pos=pos->next;

}
size--;
}
void DELETE(){
    int item,choice;
    if(head==NULL){
        puts("THERE IS NO NODE IS PRESENT TO DELETE");
    }
    else if(size==1){
        printf("There is only one node in the list ");
        head=NULL;

    }
    else{
        e2:
            printf("\nTypes of Deletion Operations :\n");
            printf("\n\t1.Delete - Beginning \n\t2.Delete - Between \n\t3.Delete - Last\n");
            printf("Enter your choise: ");
            scanf(" %d",&choice);
            switch (choice){
            case 1:
                del_b();
                break;
```

```
            case 2:

               del_m();
               break;

            case 3:
               del_e();
               break;
            default:
               puts("Enter the valid choice");
               goto e2;
               break;
            }
      }
}
void INSERT(){
   int item,choice;
   if(head==NULL){
      puts("\n\tYou can only insert at the beginning");
      struct node* ptr = (struct node*)malloc(sizeof(struct node));
      printf("\nEnter item is to be inserted : ");
      scanf(" %d",&item);
      ptr->data=item;
      ptr->next=ptr;
      head=ptr;
      size++;
   }
   else{
      e1:
```

```
        printf("\nTypes of Insertion Operations :\n");
        printf("\n\t1.Inserting - Beginning \n\t2.Inserting - Between \n\t3.Insert - Last\n");
        printf("Enter your choise: ");
        scanf(" %d",&choice);
        switch (choice){
        case 1:
           in_b();
           break;
        case 2:
           if(size<2){
              puts("\nYou cannot insert element in the middle since the size is very low");
              break;
           }
           else{
           in_m();
           break;
           }
        case 3:
           in_e();
           break;
        default:
           puts("Enter the valid choice");
           goto e1;
           break;
      }
   }
}
void DISPLAY(){
   struct node * pos = head;
```

```
    printf("\n [%d]",head);
    printf(" -> [%d|%d]", pos->data,pos->next);
    while (pos->next != head) {
        pos = pos->next;
        printf(" -> [%d|%d]", pos->data,pos->next);

  }
}
int main(){

    head = NULL;
    int op =1,i=0;

    while(op){
        if (size ==0)
            head=NULL;
        int choose;
        m1:

            puts("\n");
            puts("\n\t\t\t\tMENU: \n\n\t\t1.INSERT \n\t\t2.DELETE \n\t\t3.SEARCH
\n\t\t4.DISPLAY \n\t\t5.EXIT");
            puts("\n");
            printf("\n\tSELECT YOUR OPTION FROM MENU :");
            scanf("%d",&choose);
            switch(choose){
            case 1:
               INSERT();
               break;
            case 2:
```

```
            DELETE();
            break;
        case 3:
            SEARCH();
            break;
        case 4:
            DISPLAY();
            break;
        case 5:
            op=0;
            break;
        default:
            puts("Please Enter the valid choice : ");
            goto m1;
            break;
      }
   }
}
```

Program Output :

```
                              MENU:

               1.INSERT
               2.DELETE
               3.SEARCH
               4.DISPLAY
               5.EXIT



       SELECT YOUR OPTION FROM MENU :1

       You can only insert at the beginning

Enter item is to be inserted : 3



                              MENU:

               1.INSERT
               2.DELETE
               3.SEARCH
               4.DISPLAY
               5.EXIT



       SELECT YOUR OPTION FROM MENU :1

Types of Insertion Operations :

       1.Inserting - Beginning
       2.Inserting - Between
       3.Insert - Last
Enter your choise: 1


Enter item is to be inserted at the beginning : 1



                              MENU:

               1.INSERT
               2.DELETE
               3.SEARCH
               4.DISPLAY
               5.EXIT
```

```
        SELECT YOUR OPTION FROM MENU :1

Types of Insertion Operations :

        1.Inserting - Beginning
        2.Inserting - Between
        3.Insert - Last
Enter your choise: 2


Enter item is to be inserted at the middle : 2

Enter the position where to insert from the range (1 - 2) : 2



                                MENU:

                1.INSERT
                2.DELETE
                3.SEARCH
                4.DISPLAY
                5.EXIT



        SELECT YOUR OPTION FROM MENU :1

Types of Insertion Operations :

        1.Inserting - Beginning
        2.Inserting - Between
        3.Insert - Last
Enter your choise: 3


Enter item is to be inserted at the end : 4
```

```
                                MENU:

                1.INSERT
                2.DELETE
                3.SEARCH
                4.DISPLAY
                5.EXIT



        SELECT YOUR OPTION FROM MENU :4

 [11890272] -> [1|11890304] -> [2|11890240] -> [3|11890336] -> [4|11890272]


                                MENU:

                1.INSERT
                2.DELETE
                3.SEARCH
                4.DISPLAY
                5.EXIT



        SELECT YOUR OPTION FROM MENU :2

Types of Deletion Operations :

        1.Delete - Beginning
        2.Delete - Between
        3.Delete - Last
Enter your choise: 1



                                MENU:

                1.INSERT
                2.DELETE
                3.SEARCH
                4.DISPLAY
                5.EXIT



        SELECT YOUR OPTION FROM MENU :4

 [11890304] -> [2|11890240] -> [3|11890336] -> [4|11890304]
```

```
                              MENU:

               1.INSERT
               2.DELETE
               3.SEARCH
               4.DISPLAY
               5.EXIT



       SELECT YOUR OPTION FROM MENU :2

Types of Deletion Operations :

       1.Delete - Beginning
       2.Delete - Between
       3.Delete - Last
Enter your choise: 3



                              MENU:

               1.INSERT
               2.DELETE
               3.SEARCH
               4.DISPLAY
               5.EXIT



       SELECT YOUR OPTION FROM MENU :4

 [11890304] -> [2|11890240] -> [3|11890304]
```

```
                                MENU:

                1.INSERT
                2.DELETE
                3.SEARCH
                4.DISPLAY
                5.EXIT



        SELECT YOUR OPTION FROM MENU :1

Types of Insertion Operations :

        1.Inserting - Beginning
        2.Inserting - Between
        3.Insert - Last
Enter your choise: 3


Enter item is to be inserted at the end : 5



                                MENU:

                1.INSERT
                2.DELETE
                3.SEARCH
                4.DISPLAY
                5.EXIT



        SELECT YOUR OPTION FROM MENU :2

Types of Deletion Operations :

        1.Delete - Beginning
        2.Delete - Between
        3.Delete - Last
Enter your choise: 2

Enter the position where to delete from the range (1 - 3) : 2
```

```
                                MENU:

                1.INSERT
                2.DELETE
                3.SEARCH
                4.DISPLAY
                5.EXIT



        SELECT YOUR OPTION FROM MENU :4

 [11890304] -> [2|11890368] -> [5|11890304]


                                MENU:

                1.INSERT
                2.DELETE
                3.SEARCH
                4.DISPLAY
                5.EXIT



        SELECT YOUR OPTION FROM MENU :3
Enter the value to be search: 5
Item found at location 2


                                MENU:

                1.INSERT
                2.DELETE
                3.SEARCH
                4.DISPLAY
                5.EXIT



        SELECT YOUR OPTION FROM MENU :5

Process returned 0 (0x0)   execution time : 122.228 s
Press any key to continue.
```

EXERCISE 1.5 [STACK]

Program Code:

```
#include <stdio.h>
#include <stdlib.h>
int *arr;
int top,n;
void push(){
   if(top>=n-1){
      printf("STACK OVERFLOW!\n");
   }
   else{
      int item;
      printf("Enter the item is to be inserted :");
      scanf(" %d",&item);
      top++;
      arr[top]=item;
   }
}
void pop(){
   if(top<=-1){
      printf("STACK UNDERFLOW!\n");

   }
   else{
      int item;
      item=arr[top];
      top--;

   }
```

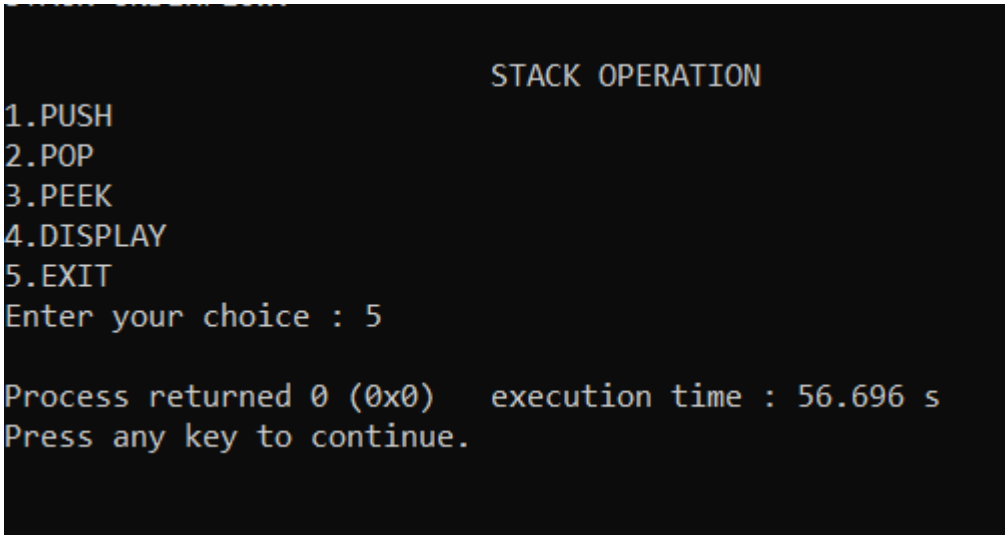
```
}
void peek(){
   printf("\nTHE TOP ELEMENT OF THE STACK IS -> %d\n",arr[top]);
}
void display(){
   for(int w=top;w>=0;w--){
      printf("| %d |",arr[w]);
      if(w==top){
         printf("<-top\n");
      }
      else{
         printf("\n");
      }
   }
}
int main(){
   top=-1;
   printf("\n\t\t\t   STACK OPERATION \n");
   printf("size of the array :");
   scanf("%d",&n);
   arr=(int*)malloc(n*sizeof(int));
   int f=1;
   int choose;
   while(f){
      l1:

         printf("\n\t\t\t   STACK OPERATION \n");
         printf("1.PUSH\n2.POP\n3.PEEK\n4.DISPLAY\n5.EXIT\n");
         printf("Enter your choice : ");
```

```
        scanf("%d",&choose);
        switch(choose){
           case 1:
              push();
              break;
           case 2:
              pop();
              break;
           case 3:
              peek();
              break;
           case 4:
              display();
              break;
           case 5:
              f=0;
              break;
           default:
              printf("### Enter valid choose ###");
              goto l1;
              break;

        }
    }
}
```

Program Output :

```
                          STACK OPERATION
size of the array :3

                          STACK OPERATION
1.PUSH
2.POP
3.PEEK
4.DISPLAY
5.EXIT
Enter your choice : 1
Enter the item is to be inserted :1

                          STACK OPERATION
1.PUSH
2.POP
3.PEEK
4.DISPLAY
5.EXIT
Enter your choice : 1
Enter the item is to be inserted :2

                          STACK OPERATION
1.PUSH
2.POP
3.PEEK
4.DISPLAY
5.EXIT
Enter your choice : 1
Enter the item is to be inserted :3

                          STACK OPERATION
1.PUSH
2.POP
3.PEEK
4.DISPLAY
5.EXIT
Enter your choice : 1
STACK OVERFLOW!

                          STACK OPERATION
1.PUSH
2.POP
3.PEEK
4.DISPLAY
5.EXIT
Enter your choice : 4
| 3 |<-top
| 2 |
| 1 |
```

```
                            STACK OPERATION
1.PUSH
2.POP
3.PEEK
4.DISPLAY
5.EXIT
Enter your choice : 2

                            STACK OPERATION
1.PUSH
2.POP
3.PEEK
4.DISPLAY
5.EXIT
Enter your choice : 4
| 2 |<-top
| 1 |

                            STACK OPERATION
1.PUSH
2.POP
3.PEEK
4.DISPLAY
5.EXIT
Enter your choice : 3

THE TOP ELEMENT OF THE STACK IS -> 2

                            STACK OPERATION
1.PUSH
2.POP
3.PEEK
4.DISPLAY
5.EXIT
Enter your choice : 2

                            STACK OPERATION
1.PUSH
2.POP
3.PEEK
4.DISPLAY
5.EXIT
Enter your choice : 2

                            STACK OPERATION
1.PUSH
2.POP
3.PEEK
4.DISPLAY
5.EXIT
Enter your choice : 2
STACK UNDERFLOW!
```


STACK OPERATION
1.PUSH
2.POP
3.PEEK
4.DISPLAY
5.EXIT
Enter your choice : 5
Process returned 0 (0x0) execution time : 56.696 s
Press any key to continue.

EXERCISE 1.6 [QUEUE]

Program Code:

```
#include <stdio.h>
#include<stdlib.h>
int *a;
int front , rear ,size;
int Isfull(){
   if (rear==size-1){
      return 1;
   }
   else{
      return 0;
   }
}
int Isempty(){
   if (rear<front){
      return 1;
   }
   else if ((rear==-1)&&(front==-1))
   {
      return 1;
   }
   else{
      return 0;
   }
}
```

```
void enqueue(){
   if (Isfull()){
      printf("You cannot insert element the queue is full \n");
   }
   else{
      if (front == -1){
         front ++;
      }
      int i;
      printf("Enter your element need to be stored: ");
      scanf(" %d",&i);
      rear++;
      a[rear]=i;
   }

}
void dequeue(){
   if (Isempty()){
      printf("You cannot delete element the queue is empty \n");
   }
   else{
      a[front]=0;
      front++;
   }

}
```

```
void Display(){
   if (Isempty()){
      printf("Your Queue is empty \n");
   }
   else{
      for(int j=front ; j<=rear ; j++){
         printf("%d ",a[j]);
      }
      printf("\n");
   }
}
int main(){
   puts("\t\t QUEUE OPERATIONS");
   puts("INITIALIZE YOUR QUEUE");
   printf("Enter size of your queue: ");
   scanf(" %d",&size);
   a=(int *)malloc(sizeof(int)*size);
   front = rear =-1;
   puts("Starting QUEUE operations ....");
   int flag=1;
   int choose;
   while (flag){
      l1:
         puts("\nSELECT FROM OPTION :");

printf("\t1.enqueue()\n\t2.dequeue()\n\t3.Isempty()\n\t4.Isfull()\n\t5.Display(
)\n\t6.Exit()\n\n");
```

```
        printf("Enter your choise(1-6): ");
        scanf(" %d",&choose);
        switch(choose){
           case 1:
              enqueue();
              break;
           case 2:
              dequeue();
              break;
           case 3:
              (Isempty())?printf("Your queue is empty\n"):printf("Your queue is
not empty\n");
              break;
           case 4:
              (Isfull())?printf("Your queue is full\n"):printf("Your queue is not
full\n");
              break;
           case 5:
              Display();
              break;
           case 6:
              flag=0;
              break;
           default:
              printf("Enter the valid choise...\n");
              goto l1;
              break;
```

```
        }


    }
}
```

Program Output :

```
                QUEUE OPERATIONS
INITIALIZE YOUR QUEUE
Enter size of your queue: 4
Starting QUEUE operations ....

SELECT FROM OPTION :
        1.enqueue()
        2.dequeue()
        3.Isempty()
        4.Isfull()
        5.Display()
        6.Exit()

Enter your choise(1-6): 1
Enter your element need to be stored: 1

SELECT FROM OPTION :
        1.enqueue()
        2.dequeue()
        3.Isempty()
        4.Isfull()
        5.Display()
        6.Exit()

Enter your choise(1-6): 1
Enter your element need to be stored: 2

SELECT FROM OPTION :
        1.enqueue()
        2.dequeue()
        3.Isempty()
        4.Isfull()
        5.Display()
        6.Exit()

Enter your choise(1-6): 1
Enter your element need to be stored: 3

SELECT FROM OPTION :
        1.enqueue()
        2.dequeue()
        3.Isempty()
        4.Isfull()
        5.Display()
        6.Exit()

Enter your choise(1-6): 1
Enter your element need to be stored: 4

SELECT FROM OPTION :
        1.enqueue()
        2.dequeue()
```

```
SELECT FROM OPTION :
        1.enqueue()
        2.dequeue()
        3.Isempty()
        4.Isfull()
        5.Display()
        6.Exit()

Enter your choise(1-6): 1
You cannot insert element the queue is full

SELECT FROM OPTION :
        1.enqueue()
        2.dequeue()
        3.Isempty()
        4.Isfull()
        5.Display()
        6.Exit()

Enter your choise(1-6): 4
Your queue is full

SELECT FROM OPTION :
        1.enqueue()
        2.dequeue()
        3.Isempty()
        4.Isfull()
        5.Display()
        6.Exit()

Enter your choise(1-6): 5
1 2 3 4

SELECT FROM OPTION :
        1.enqueue()
        2.dequeue()
        3.Isempty()
        4.Isfull()
        5.Display()
        6.Exit()

Enter your choise(1-6): 2

SELECT FROM OPTION :
        1.enqueue()
        2.dequeue()
        3.Isempty()
        4.Isfull()
        5.Display()
        6.Exit()
```

```
        4.Isfull()
        5.Display()
        6.Exit()

Enter your choise(1-6): 2

SELECT FROM OPTION :
        1.enqueue()
        2.dequeue()
        3.Isempty()
        4.Isfull()
        5.Display()
        6.Exit()

Enter your choise(1-6): 2

SELECT FROM OPTION :
        1.enqueue()
        2.dequeue()
        3.Isempty()
        4.Isfull()
        5.Display()
        6.Exit()

Enter your choise(1-6): 5
4

SELECT FROM OPTION :
        1.enqueue()
        2.dequeue()
        3.Isempty()
        4.Isfull()
        5.Display()
        6.Exit()

Enter your choise(1-6): 2

SELECT FROM OPTION :
        1.enqueue()
        2.dequeue()
        3.Isempty()
        4.Isfull()
        5.Display()
        6.Exit()

Enter your choise(1-6): 2
You cannot delete element the queue is empty

SELECT FROM OPTION :
        1.enqueue()
        2.dequeue()
        3.Isempty()
```

```
Enter your choise(1-6): 2
You cannot delete element the queue is empty

SELECT FROM OPTION :
        1.enqueue()
        2.dequeue()
        3.Isempty()
        4.Isfull()
        5.Display()
        6.Exit()

Enter your choise(1-6): 3
Your queue is empty

SELECT FROM OPTION :
        1.enqueue()
        2.dequeue()
        3.Isempty()
        4.Isfull()
        5.Display()
        6.Exit()

Enter your choise(1-6): 6

Process returned 0 (0x0)   execution time : 69.359 s
Press any key to continue.
```

EXERCISE 2 [Binary Tree**]**

Program Code:

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

//Represent a node of binary tree
struct node {
   int data;
   struct node *left;
   struct node *right;
};

//Represent the root of binary tree
struct node *root = NULL;

//createNode() will create a new node
struct node* createNode(int data) {
   //Create a new node
   struct node *newNode = (struct node*)malloc(sizeof(struct node));
   //Assign data to newNode, set left and right child to NULL
   newNode->data = data;
   newNode->left = NULL;
   newNode->right = NULL;

   return newNode;
}

//Represent a queue
```

```
struct queue
{
   int front, rear, size;
   struct node* *arr;
};

//createQueue() will create a queue
struct queue* createQueue()
{
   struct queue* newQueue = (struct queue*) malloc(sizeof( struct queue ));

   newQueue->front = -1;
   newQueue->rear = 0;
   newQueue->size = 0;

   newQueue->arr = (struct node*) malloc(100 * sizeof( struct node ));

   return newQueue;
}

//Adds a node to queue
void enqueue(struct queue* queue, struct node *temp) {
   queue->arr[queue->rear++] = temp;
   queue->size++;
}

//Deletes a node from queue
struct node *dequeue(struct queue *queue) {
   queue->size--;
```

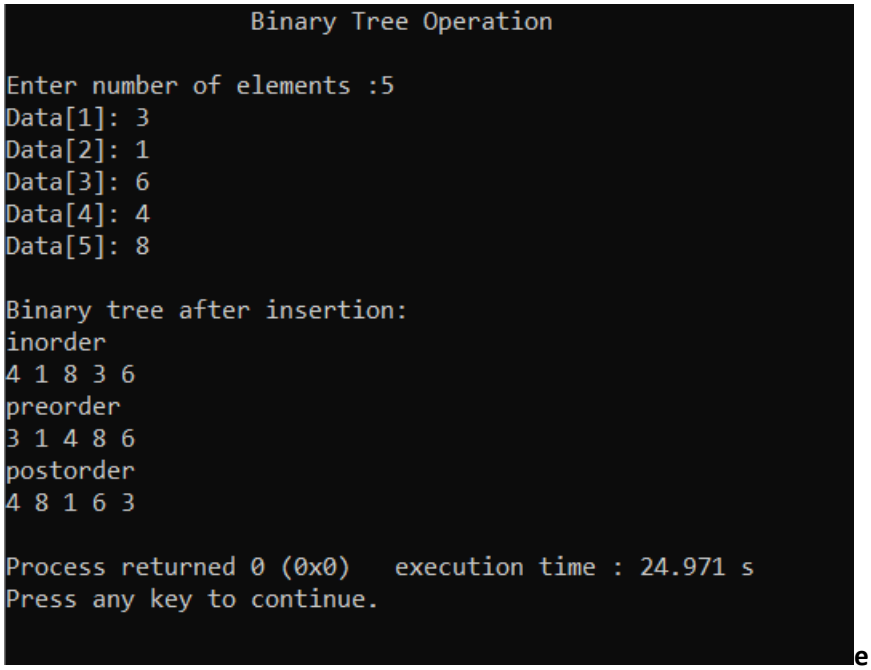
```
   return queue->arr[++queue->front];
}

//insertNode() will add new node to the binary tree
void insertNode(int data) {
   //Create a new node
   struct node *newNode = createNode(data);
   //Check whether tree is empty
   if(root == NULL) {
      root = newNode;
      return;
   }
   else {
      //Queue will be used to keep track of nodes of tree level-wise
      struct queue* queue = createQueue();
      //Add root to the queue
      enqueue(queue, root);

      while(true) {
         struct node *node = dequeue(queue);
         //If node has both left and right child, add both the child to queue
         if(node->left != NULL && node->right != NULL) {
            enqueue(queue, node->left);
            enqueue(queue, node->right);
         }
         else {
            //If node has no left child, make newNode as left child
            if(node->left == NULL) {
               node->left = newNode;
```

```
            enqueue(queue, node->left);
          }
          //If node has left child but no right child, make newNode as right child
          else {
            node->right = newNode;
            enqueue(queue, node->right);
          }
          break;
        }
      }
    }
}
//inorder() will perform inorder traversal on binary search tree
void inorderTraversal(struct node *node) {
   //Check whether tree is empty

      if(node->left != NULL)
        inorderTraversal(node->left);
      printf("%d ", node->data);
      if(node->right != NULL)
        inorderTraversal(node->right);
}
void preorderTraversal(struct node *node) {
   //Check whether tree is empty


      printf("%d ", node->data);
      if(node->left != NULL)
        preorderTraversal(node->left);
```

```
        if(node->right != NULL)
            preorderTraversal(node->right);
}
void postorderTraversal(struct node *node) {
   //Check whether tree is empty




        if(node->left != NULL)
            postorderTraversal(node->left);
        if(node->right != NULL)
            postorderTraversal(node->right);
        printf("%d ", node->data);
}
int main() {

   int n=0;
   printf("\t\t Binary Tree Operation\n\n");
   printf("Enter number of elements :");
   scanf(" %d",&n);
   for(int i=0;i<n;i++){
      int data;
      printf("Data[%d]: ",i+1);
      scanf(" %d",&data);
      insertNode(data);
   }
   printf("\nBinary tree after insertion: \n");
   printf("inorder\n");
   inorderTraversal(root);
```

```
    printf("\n");
    printf("preorder\n");
    preorderTraversal(root);
    printf("\n");
    printf("postorder\n");
    postorderTraversal(root);
    printf("\n");

    return 0;
}
```

Program Output:



e

EXERCISE 3 [Binary Search Tree]

Program code:

```
// Binary Search Tree operations in C

#include <stdio.h>

#include <stdlib.h>

struct node {

    int key;

    struct node *left, *right;

};

// Create a node

struct node *newNode(int item) {

    struct node *temp = (struct node *)malloc(sizeof(struct node));

    temp->key = item;

    temp->left = temp->right = NULL;

    return temp;

}

// Inorder Traversal

void inorder(struct node *root) {

    if (root != NULL) {

        // Traverse left

        inorder(root->left);

        // Traverse root

        printf("%d ", root->key);

        // Traverse right

        inorder(root->right);

    }


}

void postorder(struct node *root){
```

```
    if(root){
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->key);
    }

}
void preorder(struct node *root){
    if(root){
        printf("%d ", root->key);
        preorder(root->left);
        preorder(root->right);
    }

}
// Insert a node
struct node *insert(struct node *node, int key) {
    // Return a new node if the tree is empty
    if (node == NULL) return newNode(key);
    // Traverse to the right place and insert the node
    if (key < node->key)
    node->left = insert(node->left, key);
    else
    node->right = insert(node->right, key);
    return node;
}
// Find the inorder successor
struct node *minValueNode(struct node *node) {
    struct node *current = node;
```

```
    // Find the leftmost leaf
    while (current && current->left != NULL)
    current = current->left;
    return current;
}
// Deleting a node
struct node *deleteNode(struct node *root, int key) {
    // Return if the tree is empty
    if (root == NULL) return root;
    // Find the node to be deleted
    if (key < root->key)
    root->left = deleteNode(root->left, key);
    else if (key > root->key)
    root->right = deleteNode(root->right, key);
    else {
      // If the node is with only one child or no child
      if (root->left == NULL) {
        struct node *temp = root->right;
        free(root);
        return temp;
      } else if (root->right == NULL) {
        struct node *temp = root->left;
        free(root);
        return temp;
      }
      // If the node has two children
      struct node *temp = minValueNode(root->right);
      // Place the inorder successor in position of the node to be deleted
      root->key = temp->key;
```

```
        root->right = deleteNode(root->right, temp->key);
    }
    return root;
}
// Driver code
int main() {
    struct node *root = NULL;
    int choose,sz;
    int key;
    int f = 1;
    printf("\t\tBinary Search Tree\n\n");
    printf("Enter number of elements :");
    scanf(" %d",&sz);
    for(int i=0;i<sz;i++){
      int data;
      printf("Data[%d]: ",i+1);
      scanf(" %d",&data);
      root=insert(root ,data);
   }
    while(f){
    l1:
        printf("\n1.Insert \n2.Delete \n3.Display \n4.Exit\n");
        printf("\n\nEnter your choice: ");
        scanf("%d",&choose);
        printf("\n");
        switch(choose){
              case 1:
              printf("\n========>  INSERT  <=======\n\n");
              printf("Enter your value: ");
```

```
            scanf("%d",&key);
            root=insert(root ,key);
            break;
            case 2:
            printf("\n========> DELETE <=======\n\n");
            printf("Enter your value to delete: ");
            scanf("%d",&key);
            root=deleteNode(root ,key);
            break;
            case 3:
            printf("\n========> DISPLAY <=======\n\n");
            printf("\nINORDER \n");
            inorder(root);
            printf("\nPOSTORDER \n");
            postorder(root);
            printf("\nPREOREDER \n");
            preorder(root);
            printf("\n");
            break;
            case 4:
            f = 0;
            break;
            default:
            printf("\nEnter valid choice\n");
            goto l1;
            break;
        }
    }
}
```

Program Output:

```
                Binary Search Tree

Enter number of elements :5
Data[1]: 3
Data[2]: 1
Data[3]: 6
Data[4]: 4
Data[5]: 8

1.Insert
2.Delete
3.Display
4.Exit


Enter your choice: 3


========>  DISPLAY  <=======


INORDER
1 3 4 6 8
POSTORDER
1 4 8 6 3
PREOREDER
3 1 6 4 8

1.Insert
2.Delete
3.Display
4.Exit


Enter your choice: 1


========>  INSERT  <=======

Enter your value: 9
```

```
1.Insert
2.Delete
3.Display
4.Exit


Enter your choice: 3


========>  DISPLAY  <=======


INORDER
1 3 4 6 8 9
POSTORDER
1 4 9 8 6 3
PREOREDER
3 1 6 4 8 9

1.Insert
2.Delete
3.Display
4.Exit


Enter your choice: 2


========>  DELETE  <=======

Enter your value to delete: 8

1.Insert
2.Delete
3.Display
4.Exit


Enter your choice: 3


========>  DISPLAY  <=======


INORDER
1 3 4 6 9
POSTORDER
1 4 9 6 3
PREOREDER
3 1 6 4 9
```

```
1.Insert
2.Delete
3.Display
4.Exit


Enter your choice: 4


Process returned 0 (0x0)   execution time : 66.207 s
Press any key to continue.
```

EXERCISE 4 [AVL Tree]

Program Code:

```
#include<stdio.h>
typedef struct node
{
   int data;
   struct node *left,*right;
   int ht;
} node;
node *insert(node *,int);
node *Delete(node *,int);
void preorder(node *);
void inorder(node *);
int height( node *);
node *rotateright(node *);
node *rotateleft(node *);
node *RR(node *);
node *LL(node *);
node *LR(node *);
node *RL(node *);
int BF(node *);
int main()
{
   node *root=NULL;
   int x,n,i,op;
   printf("\t\t AVL Tree Implementation\n\n");
   printf("\n1)Create:");
   do
   {
```

```
printf("\n2)Insert:");
printf("\n3)Delete:");
printf("\n4)Print:");
printf("\n5)Quit:");
printf("\n\nEnter Your Choice:");
scanf("%d",&op);
switch(op)
{
case 1:
   printf("\n=======> CREATE <=======\n\n");
   printf("\nEnter no. of elements:");
   scanf("%d",&n);
   printf("\nEnter tree data:");
   root=NULL;
   for(i=0; i<n; i++)
   {
      scanf("%d",&x);
      root=insert(root,x);
   }
   break;
case 2:
   printf("\n=======> INSERT <=======\n\n");
   printf("\nEnter a data:");
   scanf("%d",&x);
   root=insert(root,x);
   break;
case 3:
   printf("\n=======> DELETE <=======\n\n");
   printf("\nEnter a data:");
```

```
            scanf("%d",&x);
            root=Delete(root,x);
            break;
        case 4:
            printf("\n=======> PREORDER SEQUENCE <=======\n\n");
            preorder(root);
            printf("\n\n=======> INORDER SEQUENCE <=======\n\n");
            inorder(root);
            printf("\n\n=======> POSTORDER SEQUENCE <=======\n\n");
            postorder(root);
            printf("\n");
            break;
        }
    } while(op!=5);
    return 0;
}
node * insert(node *T,int x)
{
    if(T==NULL)
    {
        T=(node*)malloc(sizeof(node));
        T->data=x;
        T->left=NULL;
        T->right=NULL;
    }
    else
        if(x > T->data) // insert in right subtree
        {
            T->right=insert(T->right,x);
```

```
            if(BF(T)==-2)
               if(x>T->right->data)
                  T=RR(T);
               else
                  T=RL(T);
         }
         else
            if(x<T->data)
            {
               T->left=insert(T->left,x);
               if(BF(T)==2)
                  if(x < T->left->data)
                     T=LL(T);
                  else
                     T=LR(T);
            }
      T->ht=height(T);
      return(T);
}
node * Delete(node *T,int x)
{
      node *p;
      if(T==NULL)
      {
         return NULL;
      }
      else
         if(x > T->data) // insert in right subtree
         {
```

```
        T->right=Delete(T->right,x);
        if(BF(T)==2)
            if(BF(T->left)>=0)
                T=LL(T);
            else
                T=LR(T);
    }
    else
        if(x<T->data)
        {
            T->left=Delete(T->left,x);
            if(BF(T)==-2) //Rebalance during windup
                if(BF(T->right)<=0)
                    T=RR(T);
                else
                    T=RL(T);
        }
        else
        {
//data to be deleted is found
            if(T->right!=NULL)
            {   //delete its inorder succesor
                p=T->right;
                while(p->left!= NULL)
                    p=p->left;
                T->data=p->data;
                T->right=Delete(T->right,p->data);
                if(BF(T)==2)//Rebalance during windup
                    if(BF(T->left)>=0)
```

```
                    T=LL(T);
                else
                    T=LR(T);
              \
            }
            else
               return(T->left);
         }
   T->ht=height(T);
   return(T);
}
int height(node *T)
{
   int lh,rh;
   if(T==NULL)
      return(0);
   if(T->left==NULL)
      lh=0;
   else
      lh=1+T->left->ht;
   if(T->right==NULL)
      rh=0;
   else
      rh=1+T->right->ht;
   if(lh>rh)
      return(lh);
   return(rh);
}
node * rotateright(node *x)
```

```
{
   node *y;
   y=x->left;
   x->left=y->right;
   y->right=x;
   x->ht=height(x);
   y->ht=height(y);
   return(y);
}
node * rotateleft(node *x)
{
   node *y;
   y=x->right;
   x->right=y->left;
   y->left=x;
   x->ht=height(x);
   y->ht=height(y);
   return(y);
}
node * RR(node *T)
{
   T=rotateleft(T);
   return(T);
}
node * LL(node *T)
{
   T=rotateright(T);
   return(T);
}
```

```
node * LR(node *T)
{
   T->left=rotateleft(T->left);
   T=rotateright(T);
   return(T);
}
node * RL(node *T)
{
   T->right=rotateright(T->right);
   T=rotateleft(T);
   return(T);
}
int BF(node *T)
{
   int lh,rh;
   if(T==NULL)
      return(0);
   if(T->left==NULL)
      lh=0;
   else
      lh=1+T->left->ht;
   if(T->right==NULL)
      rh=0;
   else
      rh=1+T->right->ht;
   return(lh-rh);
}
void preorder(node *T)
{
```

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

}
```

Program Output:

```
                AVL Tree Implementation


1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:

Enter Your Choice:1

=======> CREATE <=======


Enter no. of elements:6

Enter tree data:10 20 30 40 50 25

2)Insert:
3)Delete:
4)Print:
5)Quit:

Enter Your Choice:4

=======> PREORDER SEQUENCE <=======

30 20 10 25 40 50

=======> INORDER SEQUENCE <=======

10 20 25 30 40 50

=======> POSTORDER SEQUENCE <=======

10 25 20 50 40 30

2)Insert:
3)Delete:
4)Print:
5)Quit:

Enter Your Choice:2

=======> INSERT <=======


Enter a data:35
```

```
2)Insert:
3)Delete:
4)Print:
5)Quit:

Enter Your Choice:4

=======> PREORDER SEQUENCE <=======

30 20 10 25 40 35 50

=======> INORDER SEQUENCE <=======

10 20 25 30 35 40 50

=======> POSTORDER SEQUENCE <=======

10 25 20 35 50 40 30

2)Insert:
3)Delete:
4)Print:
5)Quit:

Enter Your Choice:5

Process returned 0 (0x0)   execution time : 85.736 s
Press any key to continue.
```

EXERCISE 5 [B-Tree]

Program Code:

```
#include <stdio.h>
#include <stdlib.h>
 #define MAX 4
 #define MIN 2

 struct btreeNode {
     int val[MAX + 1], count;
     struct btreeNode *link[MAX + 1];
 };

 struct btreeNode *root;

 /* creating new node */
 struct btreeNode * createNode(int val, struct btreeNode *child) {
     struct btreeNode *newNode;
     newNode = (struct btreeNode *)malloc(sizeof(struct btreeNode));
     newNode->val[1] = val;
     newNode->count = 1;
     newNode->link[0] = root;
     newNode->link[1] = child;
     return newNode;
 }

 /* Places the value in appropriate position */
 void addValToNode(int val, int pos, struct btreeNode *node,
                   struct btreeNode *child) {
     int j = node->count;
```

```
    while (j > pos) {
        node->val[j + 1] = node->val[j];
        node->link[j + 1] = node->link[j];
        j--;
    }
    node->val[j + 1] = val;
    node->link[j + 1] = child;
    node->count++;
}

/* split the node */
void splitNode (int val, int *pval, int pos, struct btreeNode *node,
  struct btreeNode *child, struct btreeNode **newNode) {
    int median, j;

    if (pos > MIN)
        median = MIN + 1;
    else
        median = MIN;

    *newNode = (struct btreeNode *)malloc(sizeof(struct btreeNode));
    j = median + 1;
    while (j <= MAX) {
        (*newNode)->val[j - median] = node->val[j];
        (*newNode)->link[j - median] = node->link[j];
        j++;
    }
    node->count = median;
    (*newNode)->count = MAX - median;
```

```
    if (pos <= MIN) {
        addValToNode(val, pos, node, child);
    } else {
        addValToNode(val, pos - median, *newNode, child);
    }
    *pval = node->val[node->count];
    (*newNode)->link[0] = node->link[node->count];
    node->count--;
}

/* sets the value val in the node */
int setValueInNode(int val, int *pval,
  struct btreeNode *node, struct btreeNode **child) {

    int pos;
    if (!node) {
        *pval = val;
        *child = NULL;
        return 1;
    }

    if (val < node->val[1]) {
        pos = 0;
    } else {
        for (pos = node->count;
            (val < node->val[pos] && pos > 1); pos--);
        if (val == node->val[pos]) {
            printf("Duplicates not allowed\n");
```

```
            return 0;
        }
    }
    if (setValueInNode(val, pval, node->link[pos], child)) {
        if (node->count < MAX) {
            addValToNode(*pval, pos, node, *child);
        } else {
            splitNode(*pval, pval, pos, node, *child, child);
            return 1;
        }
    }
    return 0;
}

/* insert val in B-Tree */
void insertion(int val) {
    int flag, i;
    struct btreeNode *child;

    flag = setValueInNode(val, &i, root, &child);
    if (flag)
        root = createNode(i, child);
}

/* copy successor for the value to be deleted */
void copySuccessor(struct btreeNode *myNode, int pos) {
    struct btreeNode *dummy;
    dummy = myNode->link[pos];
```

```
    for (;dummy->link[0] != NULL;)
        dummy = dummy->link[0];
    myNode->val[pos] = dummy->val[1];

}

/* removes the value from the given node and rearrange values */
void removeVal(struct btreeNode *myNode, int pos) {
    int i = pos + 1;
    while (i <= myNode->count) {
        myNode->val[i - 1] = myNode->val[i];
        myNode->link[i - 1] = myNode->link[i];
        i++;
    }
    myNode->count--;
}

/* shifts value from parent to right child */
void doRightShift(struct btreeNode *myNode, int pos) {
    struct btreeNode *x = myNode->link[pos];
    int j = x->count;

    while (j > 0) {
        x->val[j + 1] = x->val[j];
        x->link[j + 1] = x->link[j];
    }
    x->val[1] = myNode->val[pos];
    x->link[1] = x->link[0];
    x->count++;
```

```
    x = myNode->link[pos - 1];
    myNode->val[pos] = x->val[x->count];
    myNode->link[pos] = x->link[x->count];
    x->count--;
    return;
}

/* shifts value from parent to left child */
void doLeftShift(struct btreeNode *myNode, int pos) {
    int j = 1;
    struct btreeNode *x = myNode->link[pos - 1];

    x->count++;
    x->val[x->count] = myNode->val[pos];
    x->link[x->count] = myNode->link[pos]->link[0];

    x = myNode->link[pos];
    myNode->val[pos] = x->val[1];
    x->link[0] = x->link[1];
    x->count--;

    while (j <= x->count) {
        x->val[j] = x->val[j + 1];
        x->link[j] = x->link[j + 1];
        j++;
    }
    return;
}
```

```
/* merge nodes */
void mergeNodes(struct btreeNode *myNode, int pos) {
    int j = 1;
    struct btreeNode *x1 = myNode->link[pos], *x2 = myNode->link[pos - 1];

    x2->count++;
    x2->val[x2->count] = myNode->val[pos];
    x2->link[x2->count] = myNode->link[0];

    while (j <= x1->count) {
        x2->count++;
        x2->val[x2->count] = x1->val[j];
        x2->link[x2->count] = x1->link[j];
        j++;
    }

    j = pos;
    while (j < myNode->count) {
        myNode->val[j] = myNode->val[j + 1];
        myNode->link[j] = myNode->link[j + 1];
        j++;
    }
    myNode->count--;
    free(x1);
}

/* adjusts the given node */
void adjustNode(struct btreeNode *myNode, int pos) {
```

```
    if (!pos) {
        if (myNode->link[1]->count > MIN) {
            doLeftShift(myNode, 1);
        } else {
            mergeNodes(myNode, 1);
        }
    } else {
        if (myNode->count != pos) {
            if(myNode->link[pos - 1]->count > MIN) {
                doRightShift(myNode, pos);
            } else {
                if (myNode->link[pos + 1]->count > MIN) {
                    doLeftShift(myNode, pos + 1);
                } else {
                    mergeNodes(myNode, pos);
                }
            }
        } else {
            if (myNode->link[pos - 1]->count > MIN)
                doRightShift(myNode, pos);
            else
                mergeNodes(myNode, pos);
        }
    }
}

/* delete val from the node */
int delValFromNode(int val, struct btreeNode *myNode) {
    int pos, flag = 0;
```

```
if (myNode) {
        if (val < myNode->val[1]) {
                pos = 0;
                flag = 0;
        } else {
                for (pos = myNode->count;
                        (val < myNode->val[pos] && pos > 1); pos--);
                 if (val == myNode->val[pos]) {
                        flag = 1;
                } else {
                        flag = 0;
                }
        }
        if (flag) {
                if (myNode->link[pos - 1]) {
                        copySuccessor(myNode, pos);
                        flag = delValFromNode(myNode->val[pos], myNode->link[pos]);
                        if (flag == 0) {
                                printf("Given data is not present in B-Tree\n");
                        }
                } else {
                        removeVal(myNode, pos);
                }
        } else {
                flag = delValFromNode(val, myNode->link[pos]);
        }
        if (myNode->link[pos]) {
                if (myNode->link[pos]->count < MIN)
                        adjustNode(myNode, pos);
```

```
        }
    }
    return flag;
}

/* delete val from B-tree */
void deletion(int val, struct btreeNode *myNode) {
    struct btreeNode *tmp;
    if (!delValFromNode(val, myNode)) {
        printf("Given value is not present in B-Tree\n");
        return;
    } else {
        if (myNode->count == 0) {
            tmp = myNode;
            myNode = myNode->link[0];
            free(tmp);
        }
    }
    root = myNode;
    return;
}

/* search val in B-Tree */
void searching(int val, int *pos, struct btreeNode *myNode) {
    if (!myNode) {
        return;
    }

    if (val < myNode->val[1]) {
```

```
            *pos = 0;
        } else {
            for (*pos = myNode->count;
                (val < myNode->val[*pos] && *pos > 1); (*pos)--);
            if (val == myNode->val[*pos]) {
                printf("Given data %d is present in B-Tree", val);
                return;
            }
        }
        searching(val, pos, myNode->link[*pos]);
        return;
}

/* B-Tree Traversal */
void traversal(struct btreeNode *myNode) {
        int i;
        if (myNode) {
            for (i = 0; i < myNode->count; i++) {
                traversal(myNode->link[i]);
                printf("%d ", myNode->val[i + 1]);
            }
            traversal(myNode->link[i]);
        }
}

int main() {
        int val, ch;
        printf("\t\t B Tree Implementation\n\n");
        while (1) {
```

```
l1:
printf("1. Insertion\t2. Deletion\n");
printf("3. Searching\t4. Traversal\n");
printf("5. Exit\n\nEnter your choice:");
scanf("%d", &ch);
switch (ch) {
	case 1:
		printf("\n==INSERTION==\n");
		printf("Enter your input:");
		scanf("%d", &val);
		insertion(val);
		break;
	case 2:
		printf("\n==DELETION==\n");
		printf("Enter the element to delete:");
		scanf("%d", &val);
		deletion(val, root);
		break;
	case 3:
		printf("\n==SEARCHING==\n");
		printf("Enter the element to search:");
		scanf("%d", &val);
		searching(val, &ch, root);
		break;
	case 4:
		printf("\n==TRAVERSAL==\n");
		traversal(root);
		break;
	case 5:
```

```
                printf("\n==THANK YOU ==\n");
                exit(0);
            default:
                printf("U have entered wrong option!!\n");
                goto l1;
                break;

        }
        printf("\n");
    }
}
```

Program Output:

```
                B Tree Implementation

1. Insertion    2. Deletion
3. Searching    4. Traversal
5. Exit

Enter your choice:1

==INSERTION==
Enter your input:100

1. Insertion    2. Deletion
3. Searching    4. Traversal
5. Exit

Enter your choice:1

==INSERTION==
Enter your input:200

1. Insertion    2. Deletion
3. Searching    4. Traversal
5. Exit

Enter your choice:1

==INSERTION==
Enter your input:300

1. Insertion    2. Deletion
3. Searching    4. Traversal
5. Exit

Enter your choice:1

==INSERTION==
Enter your input:400

1. Insertion    2. Deletion
3. Searching    4. Traversal
5. Exit

Enter your choice:4

==TRAVERSAL==
100 200 300 400
```

```
1. Insertion    2. Deletion
3. Searching    4. Traversal
5. Exit

Enter your choice:2

==DELETION==
Enter the element to delete:200

1. Insertion    2. Deletion
3. Searching    4. Traversal
5. Exit

Enter your choice:4

==TRAVERSAL==
100 300 400
1. Insertion    2. Deletion
3. Searching    4. Traversal
5. Exit

Enter your choice:3

==SEARCHING==
Enter the element to search:300
Given data 300 is present in B-Tree
1. Insertion    2. Deletion
3. Searching    4. Traversal
5. Exit

Enter your choice:5

==THANK YOU ==

Process returned 0 (0x0)   execution time : 31.559 s
Press any key to continue.
```

EXERCISE 6.1 [Breadth First Search]

Program Code:

```
#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;
   printf("\t\tBreadth First Search");
        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");

}
```

Program Output:

```
               Breadth First Search
 Enter the number of vertices:3

 Enter graph data in matrix form:
2 4 5
2 3 4
1 7 8

 Enter the starting vertex:2

 The node which are reachable are:
1       2       3
Process returned 3 (0x3)   execution time : 20.916 s
Press any key to continue.
```

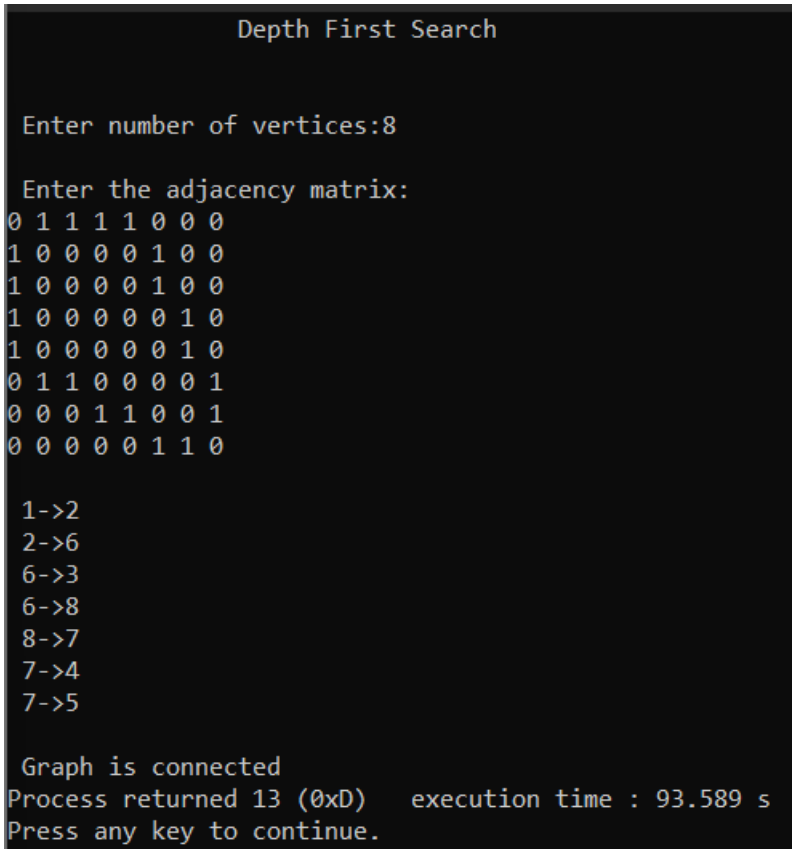
EXERCISE 6.2 [Depth First Search]

Program Code:

```
#include<stdio.h>
#include<conio.h>
int a[20][20],reach[20],n;
void dfs(int v) {
        int i;
        reach[v]=1;
        for (i=1;i<=n;i++)
          if(a[v][i] && !reach[i]) {
                printf("\n %d->%d",v,i);
                dfs(i);
        }
}
void main() {
        int i,j,count=0;
   printf("\t\tDepth First Search\n\n");
        printf("\n Enter number of vertices:");
        scanf("%d",&n);
        for (i=1;i<=n;i++) {
                reach[i]=0;
                for (j=1;j<=n;j++)
                  a[i][j]=0;
        }
        printf("\n Enter the adjacency matrix:\n");
        for (i=1;i<=n;i++)
          for (j=1;j<=n;j++)
            scanf("%d",&a[i][j]);
        dfs(1);
```

```
        printf("\n");
        for (i=1;i<=n;i++) {
                if(reach[i])
                  count++;
        }
        if(count==n)
          printf("\n Graph is connected"); else
          printf("\n Graph is not connected");
        getch();
}
```

Program Output:



EXERCISE 7 [Prim's Algorithm**]**

Program Code:

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

#define infinity 9999
#define MAX 20

int G[MAX][MAX],spanning[MAX][MAX],n;

int prims();

int main()
{
   int i,j,total_cost;
   printf("\t\tPrim's Algorithm\n\n");
   printf("Enter no. of vertices:");
   scanf("%d",&n);
   printf("\nEnter the adjacency matrix:\n");
   for(i=0;i<n;i++)
      for(j=0;j<n;j++)
         scanf("%d",&G[i][j]);
   total_cost=prims();
   printf("\nspanning tree matrix:\n");
   for(i=0;i<n;i++)
   {
      printf("\n");
      for(j=0;j<n;j++)
```

```
            printf("%d\t",spanning[i][j]);
    }
    printf("\n\nTotal cost of spanning tree=%d",total_cost);
    return 0;
}

int prims()
{
    int cost[MAX][MAX];
    int u,v,min_distance,distance[MAX],from[MAX];
    int visited[MAX],no_of_edges,i,min_cost,j;
    //create cost[][] matrix,spanning[][]
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
        {
            if(G[i][j]==0)
                cost[i][j]=infinity;
            else
                cost[i][j]=G[i][j];
            spanning[i][j]=0;
        }
    //initialise visited[],distance[] and from[]
    distance[0]=0;
    visited[0]=1;
    for(i=1;i<n;i++)
    {
        distance[i]=cost[0][i];
        from[i]=0;
        visited[i]=0;
```

```
    }
    min_cost=0; //cost of spanning tree
    no_of_edges=n-1; //no. of edges to be added
    while(no_of_edges>0)
    {
    //find the vertex at minimum distance from the tree
       min_distance=infinity;
       for(i=1;i<n;i++)
          if(visited[i]==0&&distance[i]<min_distance)
          {
             v=i;
             min_distance=distance[i];
          }
       u=from[v];
       //insert the edge in spanning tree
       spanning[u][v]=distance[v];
       spanning[v][u]=distance[v];
       no_of_edges--;
       visited[v]=1;
    for(i=1;i<n;i++)
       if(visited[i]==0&&cost[i][v]<distance[i])
       {
          distance[i]=cost[i][v];
          from[i]=v;
       }
    min_cost=min_cost+cost[u][v];
    }
    return(min_cost);
}
```

Program Output:

```
               Prim's Algorithm

Enter no. of vertices:5

Enter the adjacency matrix:
0 2 0 6 0
2 0 3 8 5
0 3 0 0 7
6 8 0 0 9
0 5 7 9 0

spanning tree matrix:

0       2       0       6       0
2       0       3       0       5
0       3       0       0       0
6       0       0       0       0
0       5       0       0       0

Total cost of spanning tree=16
Process returned 0 (0x0)   execution time : 94.107 s
Press any key to continue.
```

EXERCISE 8 [Kruskal's Algorithm]

Program Code:

```
#include<stdio.h>
#define MAX 30
typedef struct edge
{
   int u,v,w;
}edge;

typedef struct edgelist
{
   edge data[MAX];
   int n;
}edgelist;

edgelist elist;

int G[MAX][MAX],n;
edgelist spanlist;

void kruskal();
int find(int belongs[],int vertexno);
void union1(int belongs[],int c1,int c2);
void sort();
void print();

void main()
{
   int i,j,total_cost;
```

```
    printf("\t\tkruskal's Algorithm\n\n");
    printf("\nEnter number of vertices:");
    scanf("%d",&n);
    printf("\nEnter the adjacency matrix:\n");
    for(i=0;i<n;i++)
       for(j=0;j<n;j++)
          scanf("%d",&G[i][j]);
    kruskal();
    print();
}

void kruskal()
{
    int belongs[MAX],i,j,cno1,cno2;
    elist.n=0;

    for(i=1;i<n;i++)
       for(j=0;j<i;j++)
       {
          if(G[i][j]!=0)
          {
             elist.data[elist.n].u=i;
             elist.data[elist.n].v=j;
             elist.data[elist.n].w=G[i][j];
             elist.n++;
          }
       }

    sort();
```

```
    for(i=0;i<n;i++)
        belongs[i]=i;
    spanlist.n=0;
    for(i=0;i<elist.n;i++)
    {
        cno1=find(belongs,elist.data[i].u);
        cno2=find(belongs,elist.data[i].v);
    if(cno1!=cno2)
    {
        spanlist.data[spanlist.n]=elist.data[i];
        spanlist.n=spanlist.n+1;
        union1(belongs,cno1,cno2);
    }
    }
}

int find(int belongs[],int vertexno)
{
    return(belongs[vertexno]);
}

void union1(int belongs[],int c1,int c2)
{
    int i;
    for(i=0;i<n;i++)
    if(belongs[i]==c2)
    belongs[i]=c1;
}
```

```
void sort()
{
   int i,j;
   edge temp;
   for(i=1;i<elist.n;i++)
      for(j=0;j<elist.n-1;j++)
         if(elist.data[j].w>elist.data[j+1].w)
         {
            temp=elist.data[j];
            elist.data[j]=elist.data[j+1];
            elist.data[j+1]=temp;
         }
}

void print()
{
   int i,cost=0;
   for(i=0;i<spanlist.n;i++)
   {
      printf("\n%d\t%d\t%d",spanlist.data[i].u,spanlist.data[i].v,spanlist.data[i].w);
      cost=cost+spanlist.data[i].w;
   }

   printf("\n\nCost of the spanning tree=%d",cost);
}
```

Program Code:

```
               kruskal's Algorithm


Enter number of vertices:5

Enter the adjacency matrix:
0 2 0 6 0
2 0 3 8 5
0 3 0 0 7
6 8 0 0 9
0 5 7 9 0

1       0       2
2       1       3
4       1       5
3       0       6

Cost of the spanning tree=16
Process returned 30 (0x1E)   execution time : 39.067 s
Press any key to continue.
```

EXERCISE 9 [Dijikstra's Algorithm]

Program Code:

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 9999
#define MAX 10
void dijkstra(int G[MAX][MAX],int n,int startnode);

int main()
{
   int G[MAX][MAX],i,j,n,u;
   printf("\t\tDijkstra's Algorithm\n\n");
   printf("Enter no. of vertices:");
   scanf("%d",&n);
   printf("\nEnter the adjacency matrix:\n");
   for(i=0;i<n;i++)
      for(j=0;j<n;j++)
         scanf("%d",&G[i][j]);
   printf("\nEnter the starting node:");
   scanf("%d",&u);
   dijkstra(G,n,u);
   return 0;
}

void dijkstra(int G[MAX][MAX],int n,int startnode)
{

   int cost[MAX][MAX],distance[MAX],pred[MAX];
   int visited[MAX],count,mindistance,nextnode,i,j;
```

```
for(i=0;i<n;i++)
   for(j=0;j<n;j++)
      if(G[i][j]==0)
         cost[i][j]=INFINITY;
      else
         cost[i][j]=G[i][j];
//initialize pred[],distance[] and visited[]
for(i=0;i<n;i++)
{
   distance[i]=cost[startnode][i];
   pred[i]=startnode;
   visited[i]=0;
}
distance[startnode]=0;
visited[startnode]=1;
count=1;
while(count<n-1)
{
   mindistance=INFINITY;
   //nextnode gives the node at minimum distance
   for(i=0;i<n;i++)
      if(distance[i]<mindistance&&!visited[i])
      {
         mindistance=distance[i];
         nextnode=i;
      }
   //check if a better path exists through nextnode
   visited[nextnode]=1;
   for(i=0;i<n;i++)
```

```
            if(!visited[i])
               if(mindistance+cost[nextnode][i]<distance[i])
               {
                  distance[i]=mindistance+cost[nextnode][i];
                  pred[i]=nextnode;
               }
         count++;
      }

      //print the path and distance of each node
      for(i=0;i<n;i++)
         if(i!=startnode)
         {
            printf("\nDistance of node%d=%d",i,distance[i]);
            printf("\nPath=%d",i);
            j=i;
            do
            {
               j=pred[j];
               printf("<-%d",j);
            }while(j!=startnode);
         }
   }
```

Program Output:

```
               Dijkstra's Algorithm

Enter no. of vertices:5

Enter the adjacency matrix:
0 10 0 30 100
0 50 0 20 10
30 0 20 60
0
100 0 10 60 0
10 50 0 0 0

Enter the starting node:0

Distance of node1=10
Path=1<-0
Distance of node2=40
Path=2<-3<-0
Distance of node3=30
Path=3<-0
Distance of node4=20
Path=4<-1<-0
Process returned 0 (0x0)   execution time : 3932.830 s
Press any key to continue.
```

EXERCISE 10 [Floyd warshall ’s Algorithm**]**

Program Code:

```
// Floyd-Warshall Algorithm in C
#include <stdio.h>
// defining the number of vertices
#define nV 4

#define INF 999
int n=0;
void printMatrix(int matrix[][n]);

// Implementing floyd warshall algorithm
void floydWarshall(int graph[][n]) {
  int matrix[n][n], i, j, k;

  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      matrix[i][j] = graph[i][j];

  // Adding vertices individually
  for (k = 0; k < n; k++) {
    for (i = 0; i < n; i++) {
      for (j = 0; j < n; j++) {
        if (matrix[i][k] + matrix[k][j] < matrix[i][j])
          matrix[i][j] = matrix[i][k] + matrix[k][j];
      }
    }
  }
```

```
  printf ("The following matrix shows the shortest distances"
          " between every pair of vertices \n");
  printMatrix(matrix);
}

void printMatrix(int matrix[][n]) {
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      if (matrix[i][j] == INF)
        printf("%4s", "INF");
      else
        printf("%4d", matrix[i][j]);
    }
    printf("\n");
  }
}

int main() {
  int i,j,u;
    printf("\t\tFloyd's Algorithm\n\n");
    printf("Enter no. of vertices:");
    scanf("%d",&n);
    int G[n][n];
    printf("NOTE: Use (-1) for Infinity\n");
    printf("\nEnter the adjacency matrix:\n");
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            int t;
            scanf(" %d",&t);
```

```
            if(t!=-1){
                G[i][j]=t;
            }
            else
                G[i][j]=INF;
        }
    }
    printf("\n");
    floydWarshall(G);
}
```

Program Output:

```
               Floyd's Algorithm

Enter no. of vertices:4
NOTE: Use (-1) for Infinity

Enter the adjacency matrix:
0 3 -1 5
2 0 -1 4
-1 1 0 -1
-1 -1 2 0

The following matrix shows the shortest distances between every pair of vertices
   0   3   7   5
   2   0   6   4
   3   1   0   5
   5   3   2   0

Process returned 0 (0x0)   execution time : 23.356 s
Press any key to continue.
```

EXERCISE 11.1 [Open Hashing]

Program Code:

```
#include<stdio.h>
#include<stdlib.h>
#define max 7
typedef struct node
{
	int x;
	struct node* next;
}node;
node* a[max];
void insert()
{
	int d;
	printf("Enter the element to be inserted : ");
	scanf("%d",&d);
	int x=d%max;
	node* newnode=(node*)malloc(sizeof(node));
	newnode->x=d;
	newnode->next=NULL;
	if(a[x]==NULL)
		a[x]=newnode;
	else
	{
		node *ptr;
		ptr=a[x];
		int flag=1;
		while(ptr->next!=NULL)
```

```
		{
			if(ptr->x==d)
				flag=0;
			ptr=ptr->next;
		}
		if(ptr->x==d)
				flag=0;
		if(flag==1)
			ptr->next=newnode;
		else
			printf("Element already exists !");
	}
}
void search()
{
	printf("Enter the element you want to search :");
	int d;
	scanf("%d",&d);
	int x=d%max;
	node* ptr=a[x];
	int flag=0;
	while(ptr!=NULL)
	{
		if(ptr->x==d)
			flag=1;
		ptr=ptr->next;
	}
	if(flag==1)
		printf("Element present");
```

```
        else
                printf("Element not found");
}
void display()
{
        for(int i=0;i<max;i++)
        {
                node* ptr=a[i];
                printf("Index %d - ",i);
                while(ptr!=NULL)
                {
                        printf("%d, ",ptr->x);
                        ptr=ptr->next;
                }
                printf("\n");
        }
}
void main()
{

        for(int i=0;i<max;i++)
                a[i]=NULL;
   printf("\t\tOpen Hashing\n\n");
        printf("1. Insert\n2. Search\n3. Display\n4. Exit");
        while(1)
        {
                printf("\nEnter your choice :");
                int ch;
                scanf("%d",&ch);
```

```
                switch(ch)
                {
                        case 1:insert();
                                break;
                        case 2:search();
                                break;
                        case 3:display();
                                break;
                        case 4:exit(1);
                        default:printf("Wrong choice\n");
                }
        }
}
```

Program Output:

```
                Open Hashing

1. Insert
2. Search
3. Display
4. Exit
Enter your choice :1
Enter the element to be inserted : 50

Enter your choice :1
Enter the element to be inserted : 700

Enter your choice :1
Enter the element to be inserted : 76

Enter your choice :1
Enter the element to be inserted : 85

Enter your choice :1
Enter the element to be inserted : 92

Enter your choice :1
Enter the element to be inserted : 73

Enter your choice :1
Enter the element to be inserted : 101

Enter your choice :3
Index 0 - 700,
Index 1 - 50, 85, 92,
Index 2 -
Index 3 - 73, 101,
Index 4 -
Index 5 -
Index 6 - 76,

Enter your choice :4

Process returned 1 (0x1)   execution time : 821.227 s
Press any key to continue.
```

EXERCISE 11.2 [Closed Hashing]

Program Code:

```
#include<stdio.h>
#include<stdlib.h>
#define max 10
int a[max];
void insert()
{
	int d;
	printf("Enter the value to be entered : ");
	scanf("%d",&d);
	int x=d%max;
	if(a[x]==-1)
		a[x]=d;
	else
	{
		int flag=1,pos;
		for(int i=x;i<max;i++)
		{
			if(a[i]==-1)
			{
				pos=i;
				flag=0;
				break;
			}
			else if(a[i]==d)
			{
				flag=2;
```

```
                        break;
                }
        }
        if(flag==1)
        {
                for(int i=0;i<x;i++)
                {
                        if(a[i]==-1)
                        {
                                pos=i;
                                flag=0;
                                break;
                        }
                        else if(a[i]==d)
                        {
                                flag=2;
                                break;
                        }
                }
        }
        if(flag==1)
                printf("Hash table is full !");
        else if(flag==2)
                printf("element already exists !");
        else
                a[pos]=d;
    }
}
void display()
```

```
{
        for(int i=0;i<max;i++)
                printf("%d-%d\n",i,a[i]);
}
void main()
{
        for(int i=0;i<max;i++)
                a[i]=-1;
   printf("\t\tClosed Hashing\n\n");
        printf("1. Insert\n2. Display\n3. Exit");
        int n;
        while(1)
        {
                printf("\nEnter your choice : ");
                scanf("%d",&n);
                switch(n)
                {
                        case 1: insert();
                                break;
                        case 2: display();
                                break;
                        case 3:exit(0);
                        default:printf("Wrong choice\n");
                }
        }
}
```

Program Output:

```
               Closed Hashing

1. Insert
2. Display
3. Exit
Enter your choice : 1
Enter the value to be entered : 89

Enter your choice : 1
Enter the value to be entered : 18

Enter your choice : 1
Enter the value to be entered : 49

Enter your choice : 1
Enter the value to be entered : 58

Enter your choice : 1
Enter the value to be entered : 69

Enter your choice : 2
0-49
1-58
2-69
3--1
4--1
5--1
6--1
7--1
8-18
9-89

Enter your choice : 3

Process returned 0 (0x0)   execution time : 116.071 s
Press any key to continue.
```

EXERCISE 12.1 [Max-Heap]

Program Code:

```
// Max-Heap data structure in c
#include <stdio.h>
int n=0;
int size = 0;
void swap(int *a, int *b)
{
  int temp = *b;
  *b = *a;
  *a = temp;
}
void heapify(int array[], int size, int i)
{
  if (size == 1)
  {
    printf("Single element in the heap");
  }
  else
  {
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;
    if (l < size && array[l] > array[largest])
      largest = l;
    if (r < size && array[r] > array[largest])
      largest = r;
    if (largest != i)
```

```
    {
      swap(&array[i], &array[largest]);
      heapify(array, size, largest);
    }
  }
}
void insert(int array[], int newNum)
{
  if (size == 0)
  {
    array[0] = newNum;
    size += 1;
  }
  else
  {
    array[size] = newNum;
    size += 1;
    for (int i = size / 2 - 1; i >= 0; i--)
    {
      heapify(array, size, i);
    }
  }
}
void deleteRoot(int array[], int num)
{
  int i;
  for (i = 0; i < size; i++)
  {
    if (num == array[i])
```

```
      break;
  }
  swap(&array[i], &array[size - 1]);
  size -= 1;
  for (int i = size / 2 - 1; i >= 0; i--)
  {
    heapify(array, size, i);
  }
}
void printArray(int array[], int size)
{
  for (int i = 0; i < size; ++i)
    printf("%d ", array[i]);
  printf("\n");
}
int main()
{
    int fg=1;
printf("\t\tMax Heap\n\n");

printf("Enter Number Of Element : ");
scanf(" %d",&n);
int e,c;
int array[n];
for (int i=0 ; i<n;i++){
    printf("Enter Data[%d]",i);
    scanf(" %d",&e);
    insert(array, e);
}
```

```
printf("\nArray Created : \n");
printArray(array, size);
while(fg){
   printf("\n1.INSERT\n2.DELETE\n3.DISPLAY\n4.EXIT\n\n");
   printf("Enter Your Choice:");
   scanf(" %d",&c);
   switch(c){
      case 1:
         printf("Enter The Element:");
         scanf(" %d",&e);
         insert(array, e);
         printf("Element added!\n");
         printArray(array, size);
         printf("\n");
         break;
      case 2:
         printf("Enter The Element To Be Deleted:");
         scanf(" %d",&e);
         deleteRoot(array, e);
         printf("Element Deleted!\n");
         printArray(array, size);
         printf("\n");
         break;
      case 3:
         printArray(array, size);
         printf("\n");
         break;
      case 4:
         fg=0;
```

```
            printf("Thank You !\n");
            break;
        }
    }



    }
```

Program Code:-

```
                Max Heap

Enter Number Of Element : 6
Enter Data[0]3
Enter Data[1]9
Enter Data[2]2
Enter Data[3]1
Enter Data[4]4
Enter Data[5]5

Array Created :
9 4 5 1 3 2

1.INSERT
2.DELETE
3.DISPLAY
4.EXIT

Enter Your Choice:1
Enter The Element:7
Element added!
9 4 7 1 3 2 5


1.INSERT
2.DELETE
3.DISPLAY
4.EXIT

Enter Your Choice:2
Enter The Element To Be Deleted:9
Element Deleted!
7 4 5 1 3 2


1.INSERT
2.DELETE
3.DISPLAY
4.EXIT

Enter Your Choice:4
Thank You !

Process returned 0 (0x0)   execution time : 68.067 s
Press any key to continue.
```

EXERCISE 12.2 [Min-Heap]

Program Code:

```
// Min-Heap data structure in C


#include <stdio.h>
int n=0;
int size = 0;
void swap(int *a, int *b)
{
  int temp = *b;
  *b = *a;
  *a = temp;
}
void heapify(int array[], int size, int i)
{
  if (size == 1)
  {
    printf("Single element in the heap");
  }
  else
  {
    int smallest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;
    if (l < size && array[l] < array[smallest])
      smallest = l;
    if (r < size && array[r] < array[smallest])
```

```
      smallest = r;
    if (smallest != i)
    {
      swap(&array[i], &array[smallest]);
      heapify(array, size, smallest);
    }
  }
}
void insert(int array[], int newNum)
{
  if (size == 0)
  {
    array[0] = newNum;
    size += 1;
  }
  else
  {
    array[size] = newNum;
    size += 1;
    for (int i = size / 2 - 1; i >= 0; i--)
    {
      heapify(array, size, i);
    }
  }
}
void deleteRoot(int array[], int num)
{
  int i;
  for (i = 0; i < size; i++)
```

```
  {
    if (num == array[i])
      break;
  }

  swap(&array[i], &array[size - 1]);
  size -= 1;
  for (int i = size / 2 - 1; i >= 0; i--)
  {
    heapify(array, size, i);
  }
}
void printArray(int array[], int size)
{
  for (int i = 0; i < size; ++i)
    printf("%d ", array[i]);
  printf("\n");
}
int main()
{
    int fg=1;
printf("\t\tMin Heap\n\n");

printf("Enter Number Of Element : ");
scanf(" %d",&n);
int e,c;
int array[n];
for (int i=0 ; i<n;i++){
    printf("Enter Data[%d]: ",i);
```

```
    scanf(" %d",&e);
    insert(array, e);
}
printf("\nArray Created : \n");
printArray(array, size);
while(fg){
    printf("\n1.INSERT\n2.DELETE\n3.DISPLAY\n4.EXIT\n\n");
    printf("Enter Your Choice:");
    scanf(" %d",&c);
    switch(c){
        case 1:
            printf("Enter The Element:");
            scanf(" %d",&e);
            insert(array, e);
            printf("Element added!\n");
            printArray(array, size);
            printf("\n");
            break;
        case 2:
            printf("Enter The Element To Be Deleted:");
            scanf(" %d",&e);
            deleteRoot(array, e);
            printf("Element Deleted!\n");
            printArray(array, size);
            printf("\n");
            break;
        case 3:
            printArray(array, size);
            printf("\n");
```

```
            break;
        case 4:
            fg=0;
            printf("Thank You !\n");
            break;


    }
}


}
```

Program Output:

```
                Min Heap

Enter Number Of Element : 4
Enter Data[0]: 10
Enter Data[1]: 40
Enter Data[2]: 50
Enter Data[3]: 5

Array Created :
5 10 50 40

1.INSERT
2.DELETE
3.DISPLAY
4.EXIT

Enter Your Choice:2
Enter The Element To Be Deleted:10
Element Deleted!
5 40 50


1.INSERT
2.DELETE
3.DISPLAY
4.EXIT

Enter Your Choice:1
Enter The Element:20
Element added!
5 20 50 40


1.INSERT
2.DELETE
3.DISPLAY
4.EXIT

Enter Your Choice:4
Thank You !

Process returned 0 (0x0)   execution time : 34.624 s
Press any key to continue.
```

EXERCISE 13[Trie Structure**]**

Program Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#define ARRAY_SIZE(a) sizeof(a)/sizeof(a[0])
#define ALPHABET_SIZE (26)
#define CHAR_TO_INDEX© ((int)c – (int)’a’)
Struct TrieNode
{
   Struct TrieNode *children[ALPHABET_SIZE];
   Bool isEndOfWord;
};
Int hasChildren(struct TrieNode* curr)
{
   For (int I = 0; I < ALPHABET_SIZE; i++)
   {
      If (curr->children[i]) {
         Return 1;      // child found
      }
   }

   Return 0;
}
```

```
Struct TrieNode *getNode(void)
{
   Struct TrieNode *pNode = NULL;
   pNode = (struct TrieNode *)malloc(sizeof(struct TrieNode));
   if (pNode)
   {
   Int I;
   pNode->isEndOfWord = false;
   for (I = 0; I < ALPHABET_SIZE; i++)
      pNode->children[i] = NULL;
   }
   Return pNode;
}
Void insert(struct TrieNode *root, const char *key)
{
   Int level;
   Int length = strlen(key);
   Int index;
   Struct TrieNode *pCrawl = root;
   For (level = 0; level < length; level++)
   {
      Index = CHAR_TO_INDEX(key[level]);
      If (!pCrawl->children[index])
         pCrawl->children[index] = getNode();
      pCrawl = pCrawl->children[index];
   }
```

```
    pCrawl->isEndOfWord = true;
}
Bool isLeafNode(struct TrieNode* root)
{
    Return root->isEndOfWord != false;
}
Void display(struct TrieNode* root, char str[], int level)
{
    // If node is leaf node, it indicates end
    // of string, so a null character is added
    // and string is displayed

    If (isLeafNode(root))
    {
        Str[level] = '\0';
        Printf("%s \n",str);
    }

    Int I;
    For (I = 0; I < ALPHABET_SIZE; i++)
    {
        // if NON NULL child is found
        // add parent key to str and
        // call the display function recursively
        // for child node
        If (root->children[i])
```

```
        {
            Str[level] = I + ‘a’;
            Display(root->children[i], str, level + 1);
        }
    }
}
Bool search(struct TrieNode *root, const char *key)
{
    Int level;
    Int length = strlen(key);
    Int index;
    Struct TrieNode *pCrawl = root;
    For (level = 0; level < length; level++)
    {
        Index = CHAR_TO_INDEX(key[level]);
        If (!pCrawl->children[index])
            Return false;
        pCrawl = pCrawl->children[index];
    }
    Return (pCrawl->isEndOfWord);
}
Int deletion(struct TrieNode **pCrawl, char* str)
{
    // return 0 if Trie is empty
    If (*pCrawl == NULL) {
        Return 0;
```

```
}

// if the end of the string is not reached
If (*str)
{
   // recur for the node corresponding to the next character in
   // the string and if it returns 1, delete the current node
   // (if it is non-leaf)
   If (*pCrawl != NULL && (*pCrawl)->children[*str – ‘a’] != NULL &&
      Deletion(&((*pCrawl)->children[*str – ‘a’]), str + 1) &&
      (*pCrawl)->isEndOfWord == 0)
   {
      If (!hasChildren(*pCrawl))
      {
         Free(*pCrawl);
         (*pCrawl) = NULL;
         Return 1;
      }
      Else {
         Return 0;
      }
   }
}

// if the end of the string is reached
If (*str == ‘\0’ && (*pCrawl)->isEndOfWord)
```

```
    {
        // if the current node is a leaf node and doesn't have any children
        If (!hasChildren(*pCrawl))
        {
            Free(*pCrawl);    // delete the current node
            (*pCrawl) = NULL;
            Return 1;       // delete the non-leaf parent nodes
        }

        // if the current node is a leaf node and has children
        Else {
            // mark the current node as a non-leaf node (DON'T DELETE IT)
            (*pCrawl)->isEndOfWord = 0;
            Return 0;       // don't delete its parent nodes
        }
    }

    Return 0;
}
Int main()
{

    Printf("\t\tImplementation of Tries Structure \n\n");
    Printf("Create:\n");
    Int n =0;
    Char str[20];
```

```
Char output[][32] = {“Not present in trie”, “Present in trie”};
Struct TrieNode *root = getNode();
Printf(“Enter The Number Of Words:”);
Scanf(“ %d”,&n);
For (int i=0;i<n;i++){
   Printf(“\nEnter Your String [%d]:”,i);
   Scanf(“ %[^\n]%*c”,&str);
   Insert(root, str);
   Printf(“The Entered String is : %s\n”,str);
}
Int fg=1,c;
While(fg){
   Printf(“\n1.INSERT\n2.DISPLAY\n3.SEARCH\n4.DELETE\n5.EXIT\n\n”);
   Printf(“Enter Your Choice: “);
   Scanf(“ %d”,&c);
   Switch©{
      Case 1:
         Printf(“\n\tINSERT\n”);
         Printf(“\nEnter Your String:”);
         Scanf(“ %[^\n]%*c”,&str);
         Insert(root, str);
         Printf(“Element add!\n”);
         Break;
      Case 2:
         Printf(“\n\tDISPLAY\n”);
         Int level = 0;
```

```
            Char strs[20];
            Printf(“Contents Of Trie:\n”);
            Display(root, strs, level);
            Break;
        Case 3:
            Printf(“\n\tSEARCH\n”);
            Printf(“\nEnter Your String:”);
            Scanf(“ %[^\n]%*c”,&str);
            Printf(“%s --- %s\n”, str, output[search(root, str)] );
            Break;
        Case 4:
            Printf(“\n\tDELETE\n”);
            Printf(“\nEnter Your String:”);
            Scanf(“ %[^\n]%*c”,&str);
            Deletion(&root, str);
            Break;
        Case 5:
            Fg=0;
            Printf(“\t\tThank You!\n”);
            Break;

    }
  }

}
```

Program Output:

```
                Implementation of Tries Structure

Create:
Enter The Number Of Words:5

Enter Your String [0]:peter
The Entered String is : peter

Enter Your String [1]:harry
The Entered String is : harry

Enter Your String [2]:john
The Entered String is : john

Enter Your String [3]:michal
The Entered String is : michal

Enter Your String [4]:tom
The Entered String is : tom

1.INSERT
2.DISPLAY
3.SEARCH
4.DELETE
5.EXIT

Enter Your Choice: 1

        INSERT

Enter Your String:roy
Element add!

1.INSERT
2.DISPLAY
3.SEARCH
4.DELETE
5.EXIT

Enter Your Choice: 2
```

```
        DISPLAY
Contents Of Trie:
harry
john
michal
peter
roy
tom

1.INSERT
2.DISPLAY
3.SEARCH
4.DELETE
5.EXIT

Enter Your Choice: 4

        DELETE

Enter Your String:tom

1.INSERT
2.DISPLAY
3.SEARCH
4.DELETE
5.EXIT

Enter Your Choice: 2

        DISPLAY
Contents Of Trie:
harry
john
michal
peter
roy

1.INSERT
2.DISPLAY
3.SEARCH
4.DELETE
5.EXIT
```

```
1.INSERT
2.DISPLAY
3.SEARCH
4.DELETE
5.EXIT

Enter Your Choice: 3

        SEARCH

Enter Your String:peter
peter --- Present in trie

1.INSERT
2.DISPLAY
3.SEARCH
4.DELETE
5.EXIT

Enter Your Choice: 5
                Thank You!

Process returned 0 (0x0)   execution time : 122.420 s
Press any key to continue.
```