SABARMATI UNIVERSITY
(Formerly, Calorx Teachers' University)

LAB MANUAL

Data structure

M.C.A Semester I

B.C.A. Semester III

B.Sc. IT Semester III

B.Sc. CS Semester III

INDEX

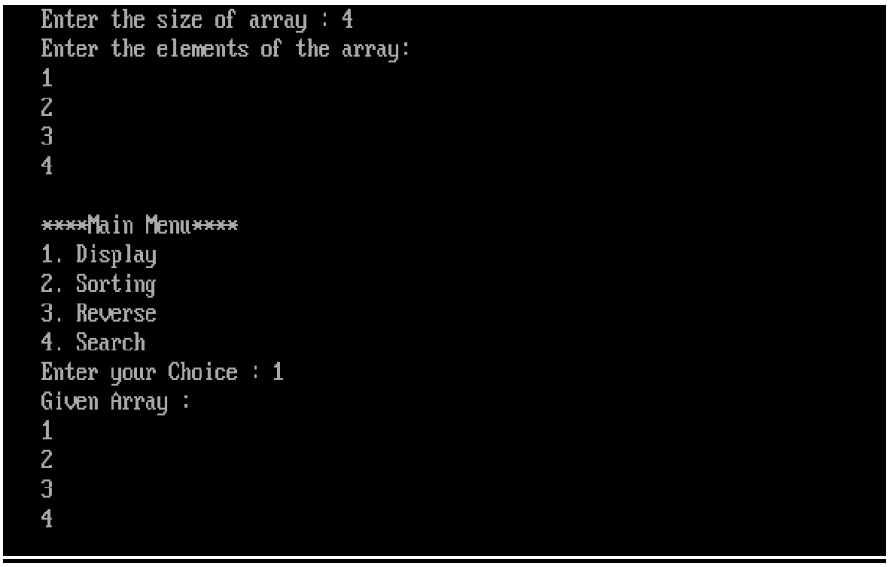
Experiment 1: Write a program to store the elements in 1-D array and perform the operations like searching, sorting and reversing the elements. [Menu Driven].

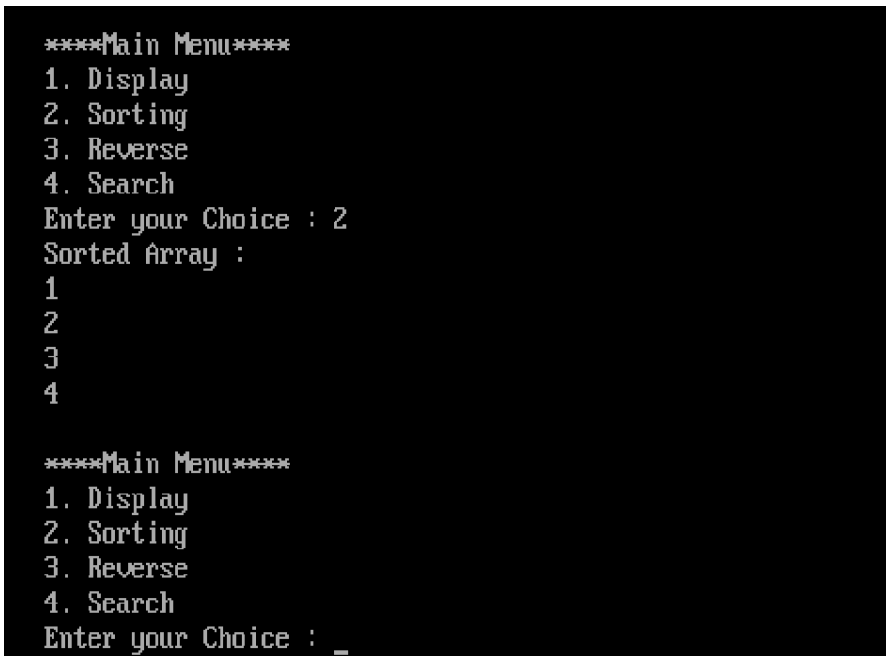
Program:

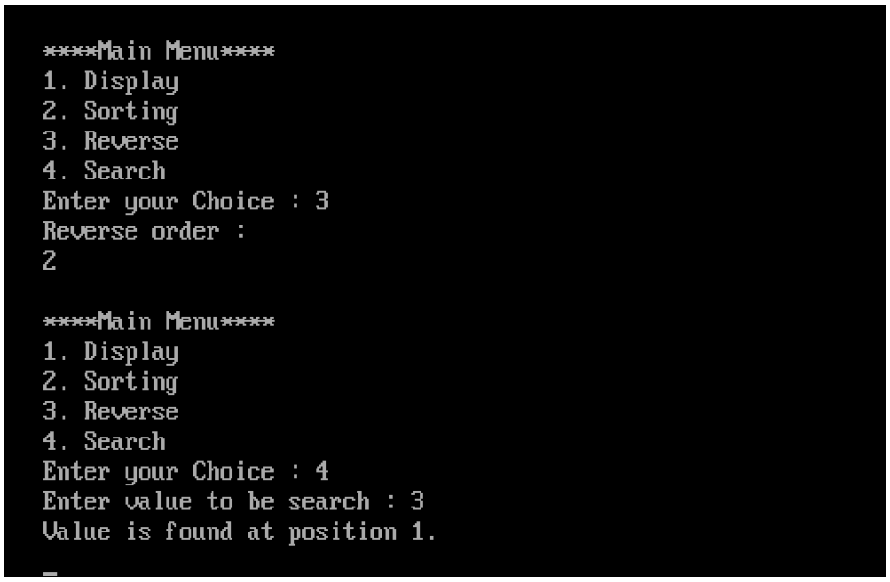
```
#include <iostream.h>
#include <conio.h>
  void disp(int arr[], int size);
  void sort(int arr[], int size);
  void reverse(int arr[], int size);
  void search(int arr[], int val, int size);
    int main() {
        clrscr();
  int size, val;
  cout << "Enter the size of array : ";
  cin >> size;
  int arr[20];
  cout << "Enter the elements of the array:" << endl;
 for (int i = 0; i < size; i++) {
     cin >> arr[i];
  }
        int ch;
  do {
     cout << "\n****Main Menu****\n";
     cout << "1. Display\n";
     cout << "2. Sorting\n";
     cout << "3. Reverse\n";
     cout << "4. Search\n";
     cout << "Enter your Choice : ";
     cin >> ch;
     switch (ch) {
        case 1:
           disp(arr, size);
           break;
        case 2:
           sort(arr, size);
           break;
        case 3:
           reverse(arr, size);
           break;
        case 4:
           cout << "Enter value to be search : ";
           cin >> val;
           search(arr, val, size);
           break;
        default:
           cout << "Invalid choice!\n";
     }
   } while (ch != 4);
    getch();
    return 0;
 }
   void search(int arr[], int val, int size) {
    int i;
```

```
for (i = 0; i < size; i++) {
    if (arr[i] == val) {
        cout << "Value is found at position " << i << "." << endl;
        return;
    }
}
cout << "Value is not found." << endl;
  }
void disp(int arr[], int size) {
cout << "Given Array :\n";
for (int i = 0; i < size; i++) {
    cout << arr[i] << endl;
    }
 }
  void sort(int arr[], int size) {
   int i, j;
   for (i = 0; i < size; i++) {
    for (j = 0; j < size - i - 1; j++) {
        if (arr[j] > arr[j + 1]) {
            int temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
           }
      }
  }
cout << "Sorted Array : \n";
for (i = 0; i < size; i++) {
    cout << arr[i] << endl;
   }
}
void reverse(int arr[], int size) {
int i = 0, j = size - 1;
while (i < j) {
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
    i++;
    j--;
 }
cout << "Reverse order : \n";
{
    cout << arr[i] << endl;
       }
  }
```

Output:


Enter the size of array : 4
Enter the elements of the array:
1
2
3
4
****Main Menu****
1. Display
2. Sorting
3. Reverse
4. Search
Enter your Choice : 1
Given Array :
1
2
3
4


****Main Menu****
1. Display
2. Sorting
3. Reverse
4. Search
Enter your Choice : 2
Sorted Array :
1
2
3
4
****Main Menu****
1. Display
2. Sorting
3. Reverse
4. Search
Enter your Choice : _


****Main Menu****
1. Display
2. Sorting
3. Reverse
4. Search
Enter your Choice : 3
Reverse order :
2
****Main Menu****
1. Display
2. Sorting
3. Reverse
4. Search
Enter your Choice : 4
Enter value to be search : 3
Value is found at position 1.
_

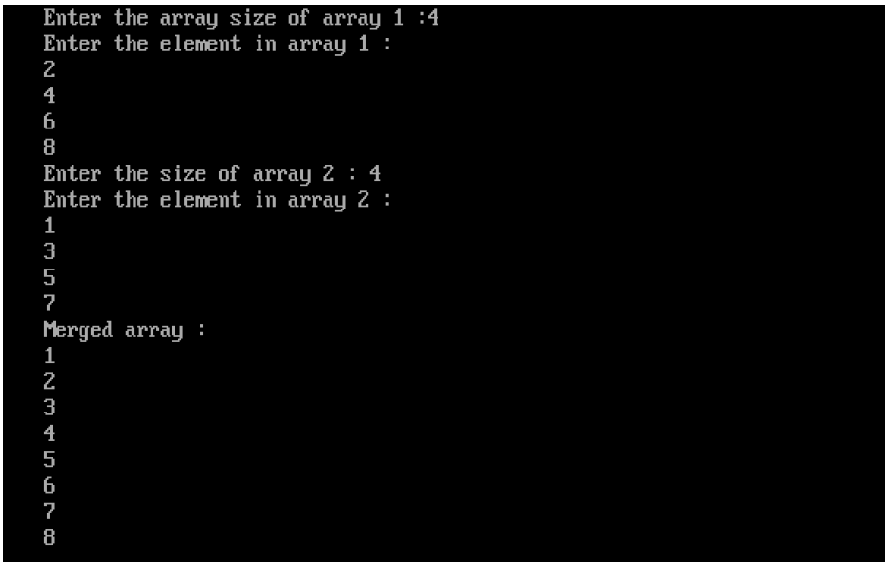
Experiment 2: Write a program to read the two arrays from the user and merge them and display the elements in sorted order [Menu Driven].

Program:

```
#include <iostream.h>
#include <conio.h>
        int main() {
 int arr1[20], arr2[20], arr3[40];
 int i, j, k, size1, size2, temp;
         clrscr();
 cout << "Enter the array size of array 1 :";
 cin >> size1;
 cout << "Enter the element in array 1 :\n";
 for (i = 0; i < size1; i++) {
    cin >> arr1[i];
 }
 cout << "Enter the size of array 2 : ";
 cin >> size2;
 cout << "Enter the element in array 2 : \n";
 for (j = 0; j < size2; j++) {
    cin >> arr2[j];
 }
 for (i = 0; i < size1; i++) {
    for (j = 0; j < size1 - i - 1; j++) {
       if (arr1[j] > arr1[j + 1]) {
          temp = arr1[j];
          arr1[j] = arr1[j + 1];
          arr1[j + 1] = temp;
       }
    }
 }
 for (i = 0; i < size2; i++) {
    for (j = 0; j < size2 - i - 1; j++) {
       if (arr2[j] > arr2[j + 1]) {
          temp = arr2[j];
          arr2[j] = arr2[j + 1];
          arr2[j + 1] = temp;
       }
    }
 }
 i = 0;
 j = 0;
 k = 0;
 while (i < size1 && j < size2) {
    if (arr1[i] < arr2[j]) {
       arr3[k] = arr1[i];
       i++;
       k++;
    } else {
       arr3[k] = arr2[j];
       j++;
       k++;
    }
```

```
  }
  while (i < size1) {
     arr3[k] = arr1[i];
     i++;
     k++;
  }
  while (j < size2) {
     arr3[k] = arr2[j];
     j++;
     k++;
  }
  cout << "Merged array :\n";
  for (k = 0; k < size1 + size2; k++) {
     cout << arr3[k] << endl;
   }
     getch();
     return 0;
}
```

Output :



Experiment 3 : Write a program to perform the Matrix addition, Multiplication and Transpose Operation. [Menu Driven].

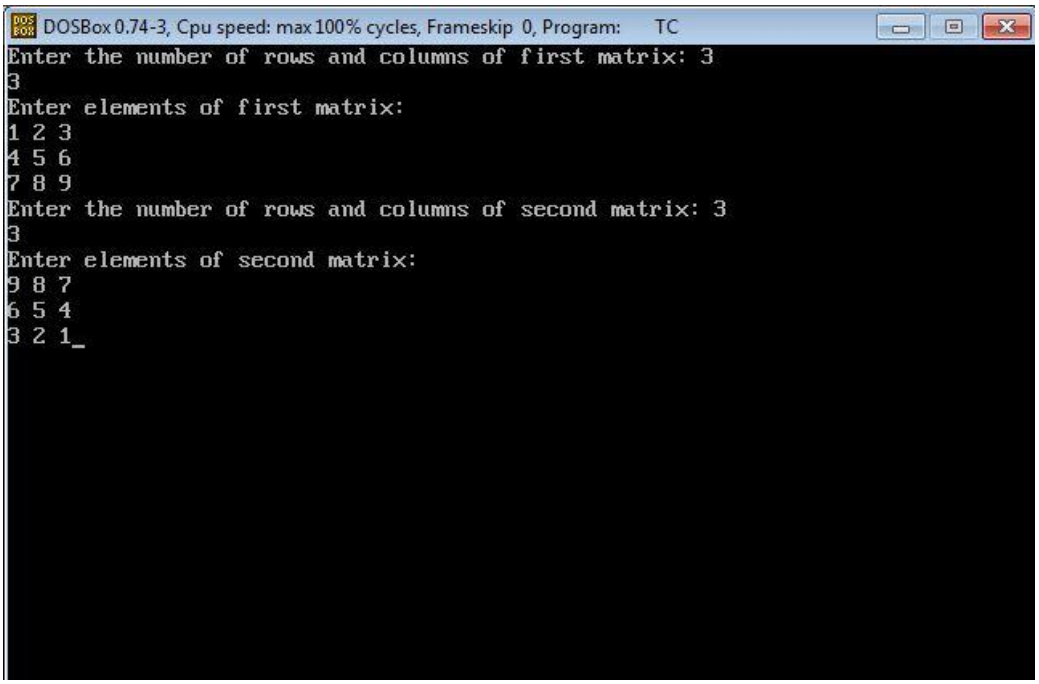
Program:

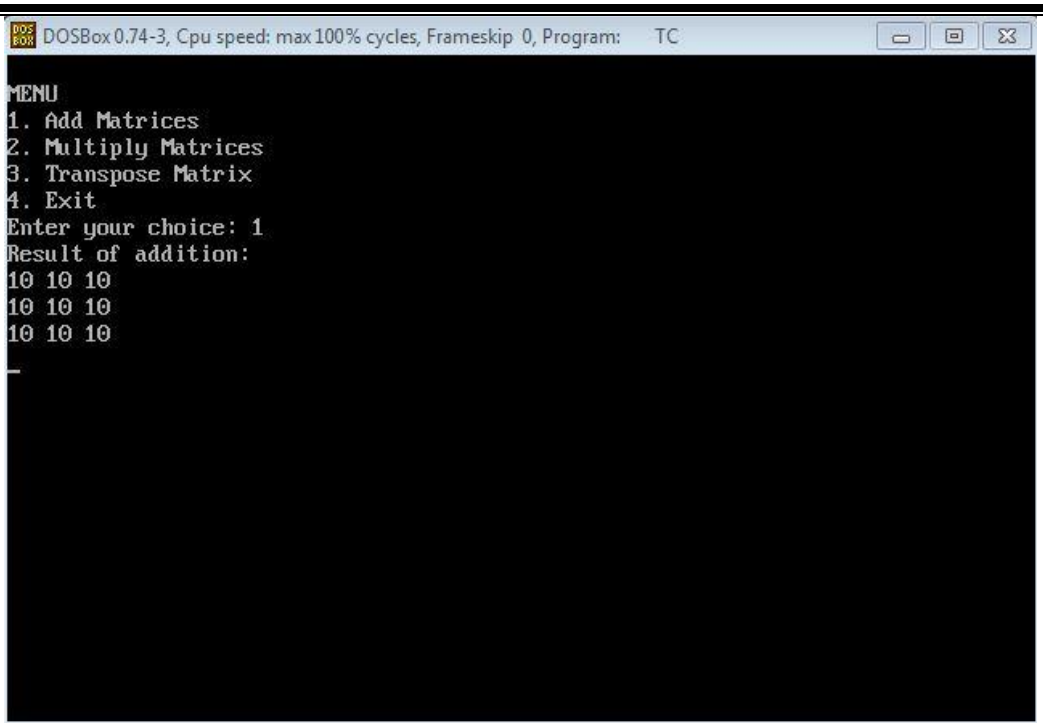
```
#include <iostream.h>
#include <conio.h>

 void matrixAddition(int A[][100], int B[][100], int result[][100], int rows, int cols);
 void matrixMultiplication(int A[][100], int B[][100], int result[][100], int rows1, int cols1, int cols2);
 void matrixTranspose(int A[][100], int transpose[][100], int rows, int cols);
 void matrixAddition(int A[][100], int B[][100], int result[][100], int rows, int cols) {
  for(int i = 0; i < rows; ++i) {
     for(int j = 0; j < cols; ++j) {
        result[i][j] = A[i][j] + B[i][j];
          }
        }
   }
   void matrixMultiplication(int A[][100], int B[][100], int result[][100], int rows1, int cols1, int cols2) {
      for(int i = 0; i < rows1; ++i) {
      for(int j = 0; j < cols2; ++j) {
        result[i][j] = 0;
        for(int k = 0; k < cols1; ++k) {
            result[i][j] += A[i][k] * B[k][j];
            }
        }
     }
   }
    void matrixTranspose(int A[][100], int transpose[][100], int rows, int cols) {
     for(int i = 0; i < rows; ++i) {
      for(int j = 0; j < cols; ++j) {
        transpose[j][i] = A[i][j];
        }
    }
   }
  void main() {
     int choice, rows1, cols1, rows2, cols2;
     int A[100][100], B[100][100], result[100][100], transpose[100][100];
        clrscr();
   cout << "Enter the number of rows and columns of first matrix: ";
   cin >> rows1 >> cols1;
   cout << "Enter elements of first matrix:" << endl;
   for(int i = 0; i < rows1; ++i) {
     for(int j = 0; j < cols1; ++j) {
        cin >> A[i][j];
      }
    }
  cout << "Enter the number of rows and columns of second matrix: ";
  cin >> rows2 >> cols2;
  cout << "Enter elements of second matrix:" << endl;
  for(i=0; i < rows2; ++i) {
```

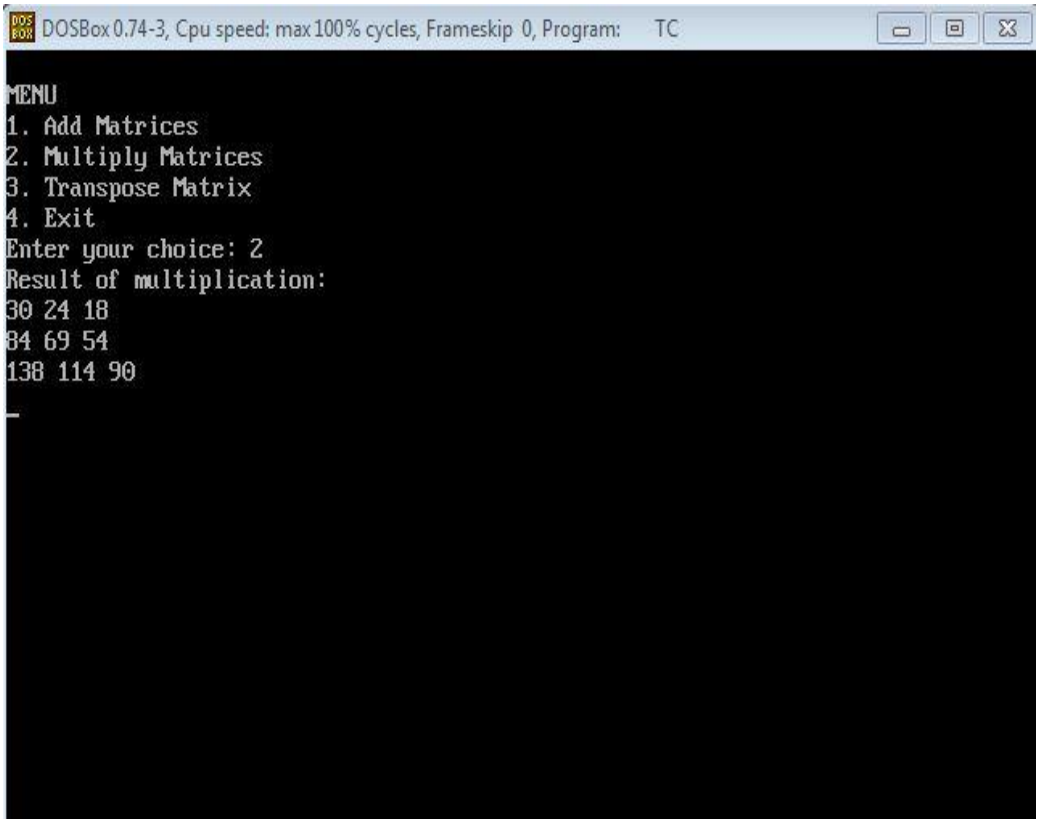
```
    for(int j = 0; j < cols2; ++j) {
      cin >> B[i][j];
    }
  }
  do {
    clrscr();
    cout << "\nMENU\n";
    cout << "1. Add Matrices\n";
    cout << "2. Multiply Matrices\n";
    cout << "3. Transpose Matrix\n";
    cout << "4. Exit\n";
    cout << "Enter your choice: ";
    cin >> choice;
    switch(choice) {
      case 1:
        if(rows1 == rows2 && cols1 == cols2) {
          matrixAddition(A, B, result, rows1, cols1);
          cout << "Result of addition:" << endl;
          for(int i = 0; i < rows1; ++i) {
            for(int j = 0; j < cols1; ++j) {
              cout << result[i][j] << " ";
            }
            cout << endl;
          }
        } else {
          cout << "Matrix addition is not possible. The matrices must have the same dimensions." << endl;
        }
        break;
      case 2:
        if(cols1 == rows2) {
          matrixMultiplication(A, B, result, rows1, cols1, cols2);
          cout << "Result of multiplication:" << endl;
          for(int i = 0; i < rows1; ++i) {
            for(int j = 0; j < cols2; ++j) {
              cout << result[i][j] << " ";
            }
            cout << endl;
          }
        } else {
         cout << "Matrix multiplication is not possible. Number of columns of first matrix must be equal to
the number of rows of second matrix." << endl;
        }
        break;
      case 3:
        cout << "Transpose of first matrix:" << endl;
        matrixTranspose(A, transpose, rows1, cols1);
        for(int i = 0; i < cols1; ++i) {
          for(int j = 0; j < rows1; ++j) {
            cout << transpose[i][j] << " ";
          }
```

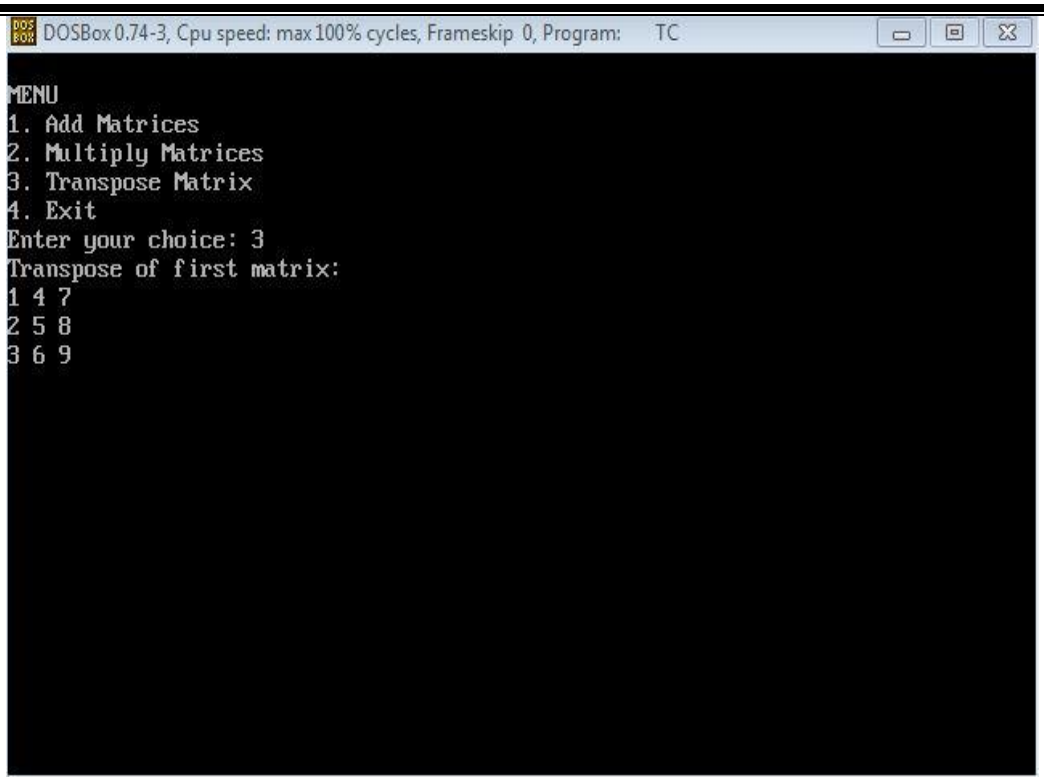
```
            cout << endl;
          }
          break;
        case 4:
          cout << "Exiting...";
          break;
        default:
          cout << "Invalid choice. Please enter a valid choice (1-4)." << endl;
     }
    getch();
  } while(choice != 4);
        getch();
}
```

Output :




DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
MENU
1. Add Matrices
2. Multiply Matrices
3. Transpose Matrix
4. Exit
Enter your choice: 1
Result of addition:
10 10 10
10 10 10
10 10 10


DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
MENU
1. Add Matrices
2. Multiply Matrices
3. Transpose Matrix
4. Exit
Enter your choice: 2
Result of multiplication:
30 24 18
84 69 54
138 114 90


DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
MENU
1. Add Matrices
2. Multiply Matrices
3. Transpose Matrix
4. Exit
Enter your choice: 3
Transpose of first matrix:
1 4 7
2 5 8
3 6 9

Experiment 4 : Write a program to create a single linked list and display the node elements in reverse order.

Program :

```
#include <iostream.h>
#include <conio.h>
 struct Node {
  int info;
  Node* next;
};
   Node* create(Node* start) {
   Node* new_node = NULL;
   Node* temp = NULL;
  int val;
  cout << "Enter -1 value to exit list.\n";
  cout << "Enter the value : \n";
  cin >> val;
  while (val != -1) {
     new_node = new Node;
     new_node->info = val;
     if (start == NULL) {
        start = new_node;
        new_node->next = NULL;
     }
     else {
        temp = start;
        while (temp->next != NULL) {
           temp = temp->next;
        }
        temp->next = new_node;
        new_node->next = NULL;
     }
     cout << "Enter the value : \n";
     cin >> val;
  }
  cout << "List is successfully created.\n";
  return start;
}

  Node* display(Node* start) {
  Node* temp = start;
  cout << "List is :\n";
  while (temp != NULL) {
     cout << temp->info << "\t";
     temp = temp->next;
  }
  return start;
}
 void reverse(Node* start) {
  Node* prev = NULL;
  Node* current = start;
  Node* next_node;
  while (current != NULL) {
```

```
    next_node = current->next;
    current->next = prev;
    prev = current;
    current = next_node;
  }
  start = prev;
  display(start);
}
        int main() {
  Node* start = NULL;
  start = create(start);
  start = display(start);
  cout << "\n";
  cout << "Reverse \t";
  reverse(start);
  getch();
  return 0;
}
```

Output :

Experiment 5 : Write a program to search the elements in the linked list and display the same.

Program:

```
#include<iostream.h>
#include<conio.h>
        class Node {
        public:
        int info;
        Node* next;
       Node(int val) : info(val), next(NULL) {}
     };
 class LinkedList {
 private:
           Node* start;
 public:
           LinkedList() : start(NULL) {}
    Node* create() {
    Node* new_node = NULL;
    Node* temp = NULL;
    int val;
    cout << "Enter -1 value to exit list.\n";
    cout << "Enter the value : \n";
    cin >> val;
    while (val != -1) {
       new_node = new Node(val);
       if (start == NULL) {
          start = new_node;
       } else {
          temp = start;
          while (temp->next != NULL) {
             temp = temp->next;
          }
          temp->next = new_node;
       }
       cout << "Enter the value : \n";
       cin >> val;
    }
    cout << "List is successfully created.\n";
    return start;
 }
 void display(Node* start) {
    Node* temp = start;
    cout << "List is :\n";
    while (temp != NULL) {
       cout << temp->info << "\t";
       temp = temp->next;
    }
 }
 Node* search(Node* start) {
    int val, count = 1;
    Node* temp = start;
    cout << "\nWhich value are you looking for?\n";
    cin >> val;
```

```
    while (temp != NULL && temp->info != val) {
       temp = temp->next;
       count++;
    }
    if (temp == NULL) {
       cout << "Value not found\n";
    } else {
       cout << "Value found at " << count << " node\n";
    }
    return start;
     }
 };
 int main() {
 LinkedList list;
 Node* start = list.create();
 list.display(start);
 cout << "\n";
  list.search(start);
   return 0;
 }
```

Output:

```
C:\TURBOC3\BIN>TC
Enter -1 value to exit list.
Enter the value :
10
Enter the value :
30
Enter the value :
40
Enter the value :
50
Enter the value :
-1
List is successfully created.
List is :
10      30      40      50

Which value are you looking for?
30
Value found at 2 node
```

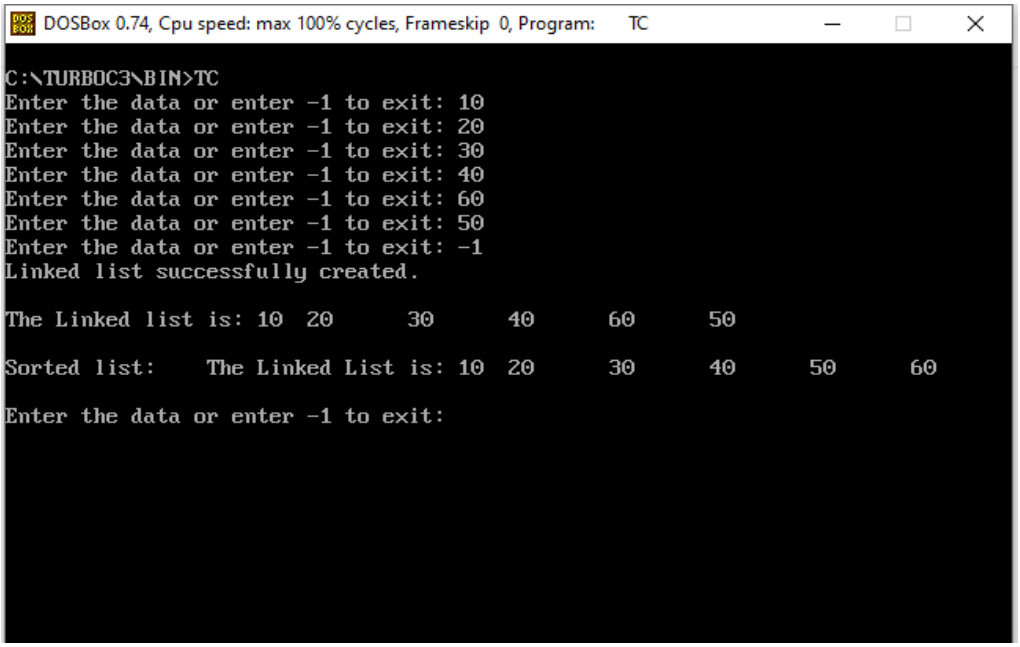
Experiment 6 : Write a program to create double linked list and sort the elements in the linked list.

Program:

```
#include <iostream.h>
        class Node {
  public:
            int data;
  Node* next;
  Node* prev;
  Node(int val) : data(val), next(NULL), prev(NULL) {}
};
class LinkedList {
  private:
            Node* start;
  public:
           LinkedList() : start(NULL) {}
      Node* create() {
      Node* new_node = NULL;
      int val;
      cout << "Enter the data or enter -1 to exit: ";
      cin >> val;
      while (val != -1) {
        new_node = new Node(val);
       if (start == NULL) {
           start = new_node;
        } else {
           Node* temp = start;
           while (temp->next != NULL) {
              temp = temp->next;
           }
           temp->next = new_node;
           new_node->prev = temp;
        }
        cout << "Enter the data or enter -1 to exit: ";
        cin >> val;
      }
      cout << "Linked list successfully created." << endl;
      return start;
  }
  Node* display() {
      Node* temp = start;
      cout << "\nThe Linked list is: ";
      while (temp != NULL) {
         cout << temp->data << "\t";
         temp = temp->next;
      }
      cout << endl;
      return start;
  }
  Node* sort() {
      Node* temp1 = start;
      while (temp1->next != NULL) {
         Node* temp2 = start;
```

```
        while (temp2->next != NULL) {
            Node* temp = temp2->next;
            if (temp2->data > temp->data) {
                int x = temp->data;
                temp->data = temp2->data;
                temp2->data = x;
            }
            temp2 = temp2->next;
        }
        temp1 = temp1->next;
    }
    Node* temp = start;
    cout << "The Linked List is: ";
    while (temp != NULL) {
        cout << temp->data << "\t";
        temp = temp->next;
    }
    cout << endl;
    return start;
  }
};
        int main() {
  LinkedList list;
  list.create();
  list.display();
  cout << "\nSorted list:\t";
  list.sort();
  return 0;
}
```

Output:



Experiment 7 : Write a program to implement the concept of Stack with Push, Pop,Peek, Display and Exit operations.

Program :

```
#include <iostream.h>
#define MAX 30
        int stack[MAX];
        int top = -1;
        void push();
        int pop();
        int peek();
        void display();
        int main() {
          int choice;
  do {
     cout << "\n **** Main Menu **** \n";
     cout << "1. Push\n";
     cout << "2. Pop\n";
     cout << "3. Peek\n";
     cout << "4. Display\n";
     cout << "Enter your choice: ";
     cin >> choice;
     cout << endl;
    switch (choice) {
        case 1:
           push();
           break;
        case 2:
           pop();
           break;
        case 3:
           peek();
           break;
        case 4:
           display();
           break;
        case 5:
           break;
     }
  } while (choice != 5);
    return 0;
 }
   void push() {
   int val;
   if (top == MAX - 1) {
     cout << "Stack is full." << endl;
   } else {
     cout << "Enter the value to be pushed: ";
     cin >> val;
     stack[++top] = val;
     cout << "Successfully pushed." << endl;
      }
   }
```

```
int pop() {
  if (top == -1) {
     cout << "Stack is already empty." << endl;
  } else {
     int val = stack[top];
     top--;
     cout << "The value is popped: " << val << endl;
   }
    return 0;
}
    int peek() {
   if (top == -1) {
     cout << "Stack is empty." << endl;
  } else {
     int topmost = stack[top];
     cout << "The topmost element of stack: " << topmost << endl;
  }
  return 0;
   }
    void display() {
   if (top == -1) {
     cout << "Stack is empty." << endl;
  } else {
     cout << "Stack is: ";
     for (int i = top; i >= 0; i--) {
        cout << "\t" << stack[i];
      }
       cout << endl;
    }
   }
```

Output :

```
C:\TURBOC3\BIN>TC

 **** Main Menu ****
1. Push
2. Pop
3. Peek
4. Display
Enter your choice: 1

Enter the value to be pushed: 10
Successfully pushed.

 **** Main Menu ****
1. Push
2. Pop
3. Peek
4. Display
Enter your choice:
```

```
 **** Main Menu ****
1. Push
2. Pop
3. Peek
4. Display
Enter your choice: 1

Enter the value to be pushed: 20
Successfully pushed.

 **** Main Menu ****
1. Push
2. Pop
3. Peek
4. Display
Enter your choice:
```

```
 **** Main Menu ****
1. Push
2. Pop
3. Peek
4. Display
Enter your choice: 1

Enter the value to be pushed: 30
Successfully pushed.

 **** Main Menu ****
1. Push
2. Pop
3. Peek
4. Display
Enter your choice:
```

```
 **** Main Menu ****
1. Push
2. Pop
3. Peek
4. Display
Enter your choice: 4

Stack is:       30      20      10

 **** Main Menu ****
1. Push
2. Pop
3. Peek
4. Display
Enter your choice:
```

```
 **** Main Menu ****
1. Push
2. Pop
3. Peek
4. Display
Enter your choice: 3

The topmost element of stack: 30

 **** Main Menu ****
1. Push
2. Pop
3. Peek
4. Display
Enter your choice: 2

The value is popped: 30

 **** Main Menu ****
1. Push
2. Pop
3. Peek
4. Display
Enter your choice:
```

```
 **** Main Menu ****
1. Push
2. Pop
3. Peek
4. Display
Enter your choice: 4

Stack is:      20      10

 **** Main Menu ****
1. Push
2. Pop
3. Peek
4. Display
Enter your choice:
```

Experiment 8 : Write a program to implement the concept of Stack using array to convert infix expression in postfix expression.

Program :

```
#include <iostream.h>
#include <stdlib.h>
#include <string.h>
        struct stack
 {
  int size;
  int top;
  char *arr;
 };
 int stackTop(struct stack* sp){
  return sp->arr[sp->top];
}
 int isEmpty(struct stack *ptr)
{
  if (ptr->top == -1)
  {
     return 1;
  }
  else
  {
     return 0;
  }
 }
 int isFull(struct stack *ptr)
{
  if (ptr->top == ptr->size - 1)
  {
     return 1;
  }
  else
  {
     return 0;
  }
 }
 void push(struct stack* ptr, char val){
  if(isFull(ptr)){
     cout << "Stack Overflow! Cannot push " << val << " to the stack" << endl;
  }
  else{
     ptr->top++;
     ptr->arr[ptr->top] = val;
  }
 }
 char pop(struct stack* ptr){
  if(isEmpty(ptr)){
     cout << "Stack Underflow! Cannot pop from the stack" << endl;
     return -1;
  }
  else{
```

```
    char val = ptr->arr[ptr->top];
    ptr->top--;
    return val;
  }
 }
 int precedence(char ch){
  if(ch == '*' || ch=='/')
    return 3;
  else if(ch == '+' || ch=='-')
    return 2;
  else
    return 0;
 }
 int isOperator(char ch){
  if(ch=='+' || ch=='-' ||ch=='*' || ch=='/')
    return 1;
  else
    return 0;
}
  char* infixToPostfix(char* infix){
  struct stack * sp = (struct stack *) malloc(sizeof(struct stack));
  sp->size = 10;
  sp->top = -1;
  sp->arr = (char *) malloc(sp->size * sizeof(char));
  char * postfix = (char *) malloc((strlen(infix)+1) * sizeof(char));
  int i=0; // Track infix traversal
  int j = 0; // Track postfix addition
  while (infix[i]!='\0')
  {
    if(!isOperator(infix[i])){
      postfix[j] = infix[i];
      j++;
      i++;
    }
    else{
      if(precedence(infix[i])> precedence(stackTop(sp))){
        push(sp, infix[i]);
        i++;
      }
      else{
        postfix[j] = pop(sp);
        j++;
      }
    }
  }
  while (!isEmpty(sp))
  {
    postfix[j] = pop(sp);
    j++;
  }
  postfix[j] = '\0';
  return postfix;
 }
 int main()
```

```
{
  char * infix = "x-y/z-k*d";
  cout << "postfix is " << infixToPostfix(infix);
  return 0;
}
```

Output:

Experiment 9 : Write a program to implement the concept of Queue with Insert, Delete, Display and Exit operations.

Program :

```
#include <iostream.h>
 int queue[100], n = 100, front = -1, rear = -1;
 void Insert() {
  int val;
  if (rear == n - 1)
     cout << "Queue Overflow" << endl;
  else {
     if (front == -1)
        front = 0;
     cout << "Insert the element in queue : " << endl;
     cin >> val;
     rear++;
     queue[rear] = val;
  }
 }
 void Delete() {
  if (front == -1 || front > rear) {
     cout << "Queue Underflow ";
     return ;
  } else {
     cout << "Element deleted from queue is : " << queue[front] << endl;
     front++;;
      }
  }
    void Display() {
    if (front == -1)
     cout << "Queue is empty" << endl;
  else {
     cout << "Queue elements are : ";
     for (int i = front; i <= rear; i++)
        cout << queue[i] << " ";
     cout << endl;
         }
    }
 int main() {
 int ch;
 cout << "1) Insert element to queue" << endl;
 cout << "2) Delete element from queue" << endl;
 cout << "3) Display all the elements of queue" << endl;
 cout << "4) Exit" << endl;
 do {
    cout << "Enter your choice : " << endl;
    cin >> ch;
    switch (ch) {
       case 1: Insert();
               break;
       case 2: Delete();
               break;
       case 3: Display();
```

```
            break;
      case 4: cout << "Exit" << endl;
            break;
      default: cout << "Invalid choice" << endl;
    }
     } while(ch != 4);
    return 0;
  }
```

Output:

```
C:\TURBOC3\BIN>TC
1) Insert element to queue
2) Delete element from queue
3) Display all the elements of queue
4) Exit
Enter your choice :
1
Insert the element in queue :
2
Enter your choice :
1
Insert the element in queue :
3
Enter your choice :
1
Insert the element in queue :
4
Enter your choice :
2
Element deleted from queue is : 2
Enter your choice :
3
Queue elements are : 3 4
Enter your choice :
```

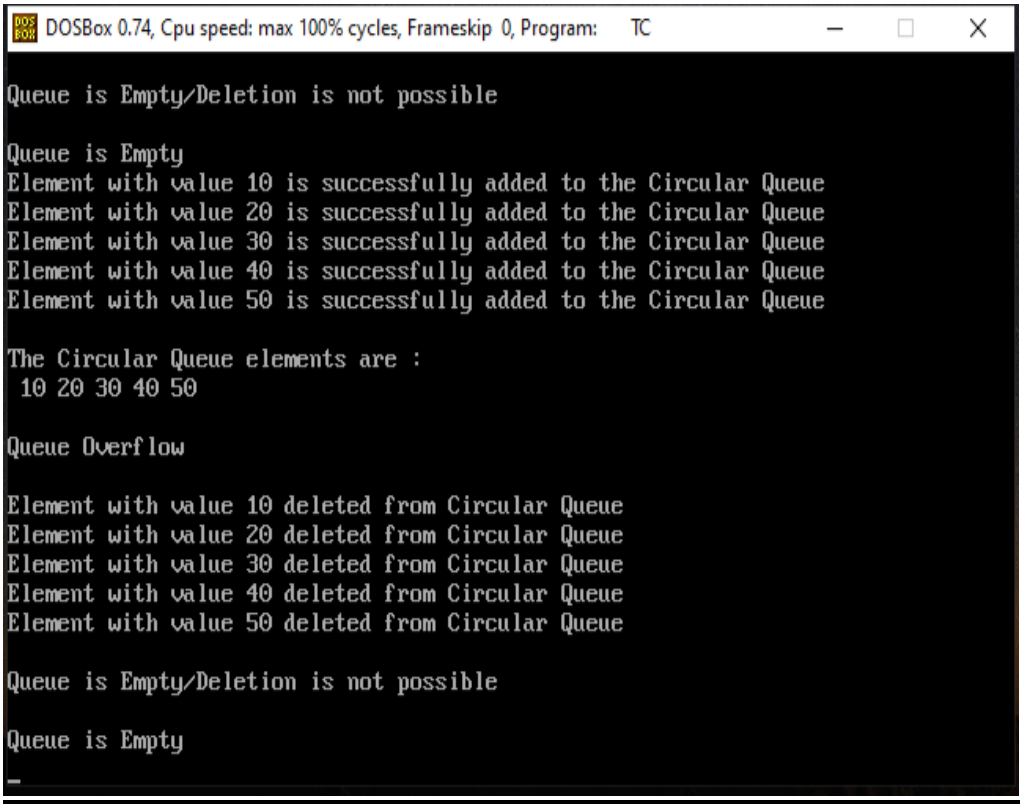
Experiment 10 : Write a program to implement the concept of Circular Queue.

Program :

```
#include <iostream.h>
const int MAX_SIZE = 5;
struct circular_queue {
 int arr[MAX_SIZE];
 int front, rear;
};
void initialise_queue(circular_queue *cq) {
  cq->front = -1;
  cq->rear = -1;
}
    void enQueue(circular_queue *cq, int value) {
     if((cq->front == 0 && cq->rear == MAX_SIZE-1) || (cq->front == cq->rear+1)) {
      cout << "\nQueue Overflow\n\n";
      return;
   }
     if(cq->front == -1) {
      cq->front = 0;
      cq->rear = 0;
   }
    else {
      if(cq->rear == MAX_SIZE-1)
          cq->rear = 0;
      else
          cq->rear += 1;
   }
    cq->arr[cq->rear] = value;
  cout << "Element with value " << value;
  cout << " is successfully added to the Circular Queue\n";
}
void deQueue(circular_queue *cq) {
  if(cq->front == -1) {
      cout << "\nQueue is Empty/Deletion is not possible\n";
      return;
   }
  cout << "Element with value " << cq->arr[cq->front];
  cout << " deleted from Circular Queue\n";
  if(cq->front == cq->rear) {
      cq->front = -1;
      cq->rear = -1;
   }
  else {
      if(cq->front == MAX_SIZE-1)
          cq->front = 0;
      else
          cq->front = cq->front+1;
   }
}
void display_queue(circular_queue *cq) {
  int curr_front = cq->front, curr_rear = cq->rear;
  if(cq->front == -1) {
```

```
    cout << "\nQueue is Empty\n";
    return;
  }
  cout << "\nThe Circular Queue elements are :\n";
  cout << " ";
  if(curr_front <= curr_rear) {
    while(curr_front <= curr_rear) {
      cout << cq->arr[curr_front] << " ";
      curr_front++;
    }
  }
  /*
    If front >= rear we have to go first
    from front to last element(MAX_SIZE-1)
    then from 0 to rear end
  */
  else {
    // Iterating from front to last element(MAX_SIZE-1)
    while(curr_front <= MAX_SIZE-1) {
      cout << cq->arr[curr_front] << " ";
      curr_front++;
    }
        curr_front = 0;
    while(curr_front <= curr_rear) {
      cout << cq->arr[curr_front] << " ";
      curr_front++;
    }
  }
    cout << "\n";
}
 int main() {
 circular_queue q;
 initialise_queue(&q);
 enQueue(&q, 10);
 enQueue(&q, 20);
 enQueue(&q, 30);
 enQueue(&q, 40);
 enQueue(&q, 50);
 display_queue(&q);
 enQueue(&q, 60);
 deQueue(&q);
 deQueue(&q);
 deQueue(&q);
 deQueue(&q);
 deQueue(&q);
 deQueue(&q);
 display_queue(&q);
   return 0;
}
```

Output :


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Queue is Empty/Deletion is not possible
Queue is Empty
Element with value 10 is successfully added to the Circular Queue
Element with value 20 is successfully added to the Circular Queue
Element with value 30 is successfully added to the Circular Queue
Element with value 40 is successfully added to the Circular Queue
Element with value 50 is successfully added to the Circular Queue
The Circular Queue elements are :
10 20 30 40 50
Queue Overflow
Element with value 10 deleted from Circular Queue
Element with value 20 deleted from Circular Queue
Element with value 30 deleted from Circular Queue
Element with value 40 deleted from Circular Queue
Element with value 50 deleted from Circular Queue
Queue is Empty/Deletion is not possible
Queue is Empty

Experiment 11: Write a program to implement the concept of Deque.

Program:

```
#include <iostream.h>
#define max 5
  int front = -1;
  int rear = -1;
  int insert_rear();
  int insert_front();
    void display();
    int deleteq_rear();
    int deleteq_front();
    int q[max];
          int main() {
          int choice;
   do {
      cout << "\n **** Main Menu **** \n";
      cout << "1. Insert From Rear\n";
      cout << "2. Insert From Front\n";
      cout << "3. Delete From Front\n";
      cout << "4. Delete From Rear\n";
      cout << "5. Display\n";
      cout << "Enter your choice : ";
      cin >> choice;
      cout << "\n";
             switch(choice) {
         case 1:
            insert_rear();
            break;
         case 2:
            insert_front();
            break;
         case 3:
            deleteq_front();
            break;
         case 4:
            deleteq_rear();
            break;
         case 5:
            display();
            break;
         case 6:
            break;
      }
   } while(choice != 6);
   return 0;
 }
   int insert_rear() {
   int val;
   cout << "Enter value: ";
   cin >> val;
      if ((rear + 1) % max == front) {
      cout << "Queue is full." << endl;
```

```
    return 0;
  } else if (rear == -1) {
    rear = front = 0;
    q[rear] = val;
    cout << "Inserted successfully." << endl;
    return val;
  } else {
    rear = (rear + 1) % max;
    q[rear] = val;
    cout << "Inserted successfully." << endl;
    return val;
  }
}
 int insert_front() {
 int val;
 cout << "Enter value: ";
 cin >> val;
    if ((rear + 1) % max == front) {
    cout << "Queue is full." << endl;
    return 0;
  } else if (front == -1) {
    rear = front = 0;
    q[front] = val;
    cout << "Inserted successfully." << endl;
    return val;
  } else {
    front = (front - 1 + max) % max;
    q[front] = val;
    cout << "Inserted successfully." << endl;
    return val;
   }
 }
 int deleteq_front() {
 int val;
 if (front == -1) {
    cout << "Queue is empty." << endl;
    return -1;
 } else if (front == rear) {
    val = q[front];
    front = rear = -1;
    cout << "Deleted value: " << val << endl;
    return val;
 } else {
    val = q[front];
    front = (front + 1) % max;
    cout << "Deleted value: " << val << endl;
    return val;
  }
}
 int deleteq_rear() {
 int val;
 if (rear == -1) {
    cout << "Queue is empty." << endl;
    return -1;
```

```
  } else if (front == rear) {
     val = q[rear];
     front = rear = -1;
     cout << "Deleted value: " << val << endl;
     return val;
  } else {
     val = q[rear];
     rear = (rear - 1 + max) % max;
     cout << "Deleted value: " << val << endl;
     return val;
  }
}
 void display() {
  int i;
  if (front == -1) {
     cout << "Queue is empty." << endl;
  } else {
     cout << "Queue is: ";
     for (i = front; i != rear; i = (i + 1) % max) {
        cout << q[i] << " ";
     }
     cout << q[i] << endl;
    }
  }
```

Output:

```
 **** Main Menu ****
1.Insert From Rear
2.Insert From Front
3.Delete From Front
4.Delete From Rear
5.Display
Enter your choice :1

Enter value :20
Inserted successfully.
 **** Main Menu ****
1.Insert From Rear
2.Insert From Front
3.Delete From Front
4.Delete From Rear
5.Display
Enter your choice :1

Enter value :30
Inserted successfully.
 **** Main Menu ****
1.Insert From Rear
2.Insert From Front
3.Delete From Front
4.Delete From Rear
5.Display
Enter your choice :1

Enter value :40
Inserted successfully.
```

```
 **** Main Menu ****
1.Insert From Rear
2.Insert From Front
3.Delete From Front
4.Delete From Rear
5.Display
Enter your choice :2

Enter value :10
Inserted successfully.
 **** Main Menu ****
1.Insert From Rear
2.Insert From Front
3.Delete From Front
4.Delete From Rear
5.Display
Enter your choice :1

Enter value :50
Inserted successfully.
 **** Main Menu ****
1.Insert From Rear
2.Insert From Front
3.Delete From Front
4.Delete From Rear
5.Display
Enter your choice :5

Queue is : 10  20  30  40  50
 **** Main Menu ****
1.Insert From Rear
2.Insert From Front
3.Delete From Front
4.Delete From Rear
5.Display
Enter your choice :3

Deleted value : 10
 **** Main Menu ****
```

```
 **** Main Menu ****
1.Insert From Rear
2.Insert From Front
3.Delete From Front
4.Delete From Rear
5.Display
Enter your choice :4

Deleted value : 50
 **** Main Menu ****
1.Insert From Rear
2.Insert From Front
3.Delete From Front
4.Delete From Rear
5.Display
Enter your choice :5

Queue is : 20  30  40
```

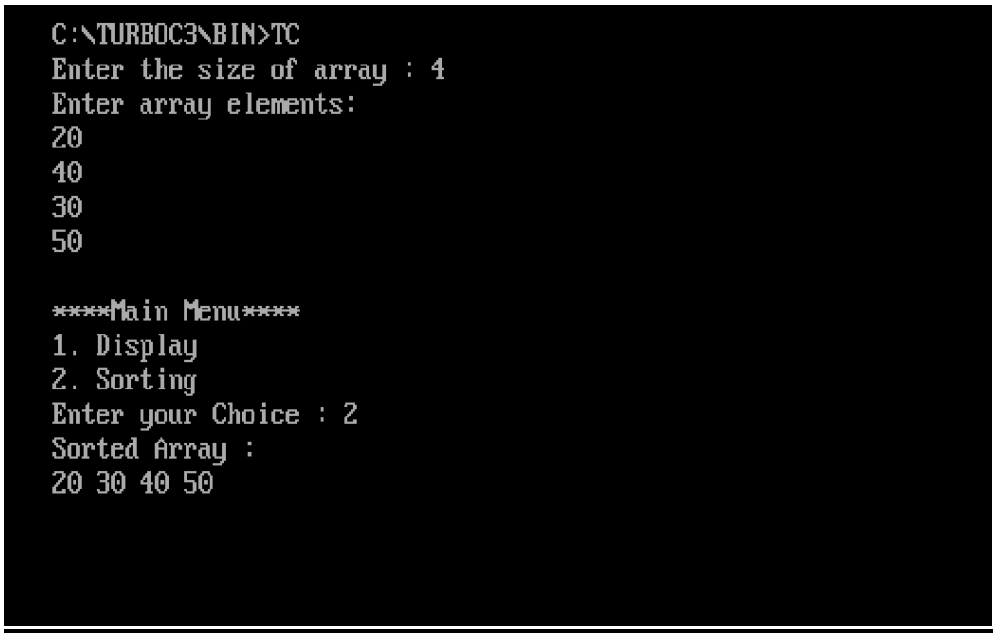
Experiment 12: Write a program to implement bubble sort.

Program :

```
#include <iostream.h>
 const int MAX_SIZE = 20;
 int arr[MAX_SIZE];
 void disp(int size) {
   cout << "Given Array :" << endl;
   for (int i = 0; i < size; i++) {
      cout << arr[i] << endl;
   }
 }
  void sort(int size) {
   int i, j;
   for (i = 0; i < size; i++) {
      for (j = 0; j < size - i - 1; j++) {
         if (arr[j] > arr[j + 1]) {
            int temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
         }
      }
   }
   cout << "Sorted Array : " << endl;
   for (i = 0; i < size; i++) {
      cout << arr[i] << " ";
   }
   cout << endl;
 }
  int main() {
   int size, ch;
   cout << "Enter the size of array : ";
   cin >> size;
   cout << "Enter array elements: " << endl;
   for (int i = 0; i < size; i++) {
      cin >> arr[i];
   }
   do {
      cout << "\n****Main Menu****\n";
      cout << "1. Display\n";
      cout << "2. Sorting\n";
      cout << "Enter your Choice : ";
      cin >> ch;
       switch (ch) {
         case 1:
            disp(size);
            break;
         case 2:
            sort(size);
            break;
         default:
            cout << "Invalid choice" << endl;
      }
```

```
  } while (ch != 2);
return 0;
}
```

Output:

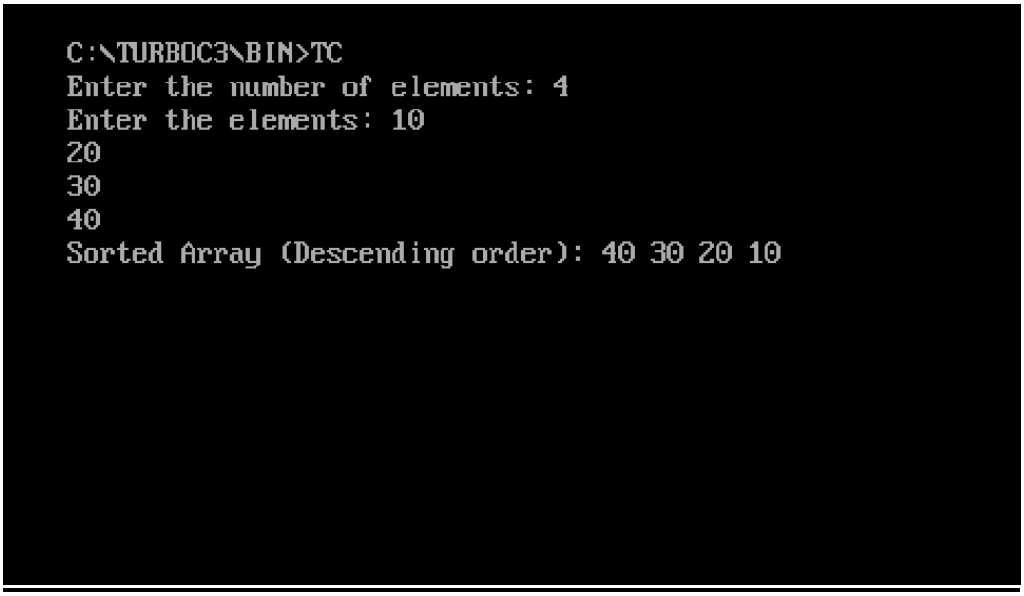


Experiment 13: Write a program to implement selection sort algorithm to arrange a list of elements in descending order.

Program :

```
#include <iostream.h>
 const int MAX_SIZE = 100;
 void selectionSortDescending(int arr[], int n) {
  for (int i = 0; i < n - 1; i++) {
     int maxIndex = i;
     for (int j = i + 1; j < n; j++) {
        if (arr[j] > arr[maxIndex]) {
           maxIndex = j;
        }
     }
     if (maxIndex != i) {
        // Swap arr[i] and arr[maxIndex]
        int temp = arr[i];
        arr[i] = arr[maxIndex];
        arr[maxIndex] = temp;
     }
  }
 }
 void displayArray(int arr[], int n) {
  cout << "Sorted Array (Descending order): ";
  for (int i = 0; i < n; i++) {
     cout << arr[i] << " ";
  }
  cout << endl;
 }
  int main() {
  int n;
  cout << "Enter the number of elements: ";
  cin >> n;
  if (n > MAX_SIZE || n <= 0) {
     cout << "Invalid size of the array. Please enter a positive number less than or equal to " << MAX_SIZE <<
endl;
     return 1;
  }
  int arr[MAX_SIZE];
  cout << "Enter the elements: ";
  for (int i = 0; i < n; i++) {
     cin >> arr[i];
   }
   selectionSortDescending(arr, n);
   displayArray(arr, n);
    return 0;
  }
```

Output:


C:\TURBOC3\BIN>TC
Enter the number of elements: 4
Enter the elements: 10
20
30
40
Sorted Array (Descending order): 40 30 20 10

Experiment 14: Write a program to implement insertion sort algorithm to arrange a list of integers in ascending order.

Program:

```
#include <iostream.h>
 const int MAX_SIZE = 100;
 void insertionSortAscending(int arr[], int n) {
   for (int i = 1; i < n; i++) {
       int key = arr[i];
       int j = i - 1;
       while (j >= 0 && arr[j] > key) {
           arr[j + 1] = arr[j];
           j = j - 1;
       }
       arr[j + 1] = key;
   }
 }
  void displayArray(int arr[], int n) {
   cout << "Sorted Array (Ascending order): ";
   for (int i = 0; i < n; i++) {
       cout << arr[i] << " ";
   }
   cout << endl;
  }
 int main() {
   int n;
   cout << "Enter the number of elements: ";
   cin >> n;
  if (n > MAX_SIZE || n <= 0) {
       cout << "Invalid size of the array. Please enter a positive number less than or equal to " << MAX_SIZE <<
endl;
       return 1;
   }
   int arr[MAX_SIZE];
   cout << "Enter the elements: ";
   for (int i = 0; i < n; i++) {
       cin >> arr[i];
   }
     insertionSortAscending(arr, n);
      displayArray(arr, n);
       return 0;
 }
```

Output:

```
Enter the number of elements: 3
Enter the elements: 20
30
10
Sorted Array (Ascending order): 10 20 30
_
```

Experiment 15: Write a program to implement merge sort algorithm to sort a list of integers in ascending order.

Program:

```
#include <iostream.h>
const int MAX_SIZE = 6;
 void merge(int arr[], int low, int mid, int high) {
const int n1 = mid - low + 1;
const int n2 = high - mid;
int L[MAX_SIZE], R[MAX_SIZE];
  for (int i = 0; i < n1; i++)
    L[i] = arr[low + i];
for (int j = 0; j < n2; j++)
    R[j] = arr[mid + 1 + j];
i = 0, j = 0;
 for (int k = low; i < n1 && j < n2; k++) {
    if (L[i] <= R[j]) {
        arr[k] = L[i];
        i++;
    } else {
        arr[k] = R[j];
        j++;
    }
 }
while (i < n1) {
    arr[low + i + j] = L[i];
    i++;
 }
 while (j < n2) {
    arr[low + i + j] = R[j];
    j++;
    }
 }
 void mergeSort(int arr[], int low, int high) {
 if (low < high) {
    int mid = low + (high - low) / 2;
   mergeSort(arr, low, mid);
    mergeSort(arr, mid + 1, high);
    merge(arr, low, mid, high);
 }
 }
  int main() {
  int arr[MAX_SIZE] = {12, 11, 13, 5, 6, 7};
 cout << "Original Array: ";
 for (int i = 0; i < MAX_SIZE; i++)
    cout << arr[i] << " ";
cout << endl;
mergeSort(arr, 0, MAX_SIZE - 1);
cout << "Sorted Array: ";
for (i = 0; i < MAX_SIZE; i++)
    cout << arr[i] << " ";
cout << endl;
return 0;
```

```
}
```

Output:

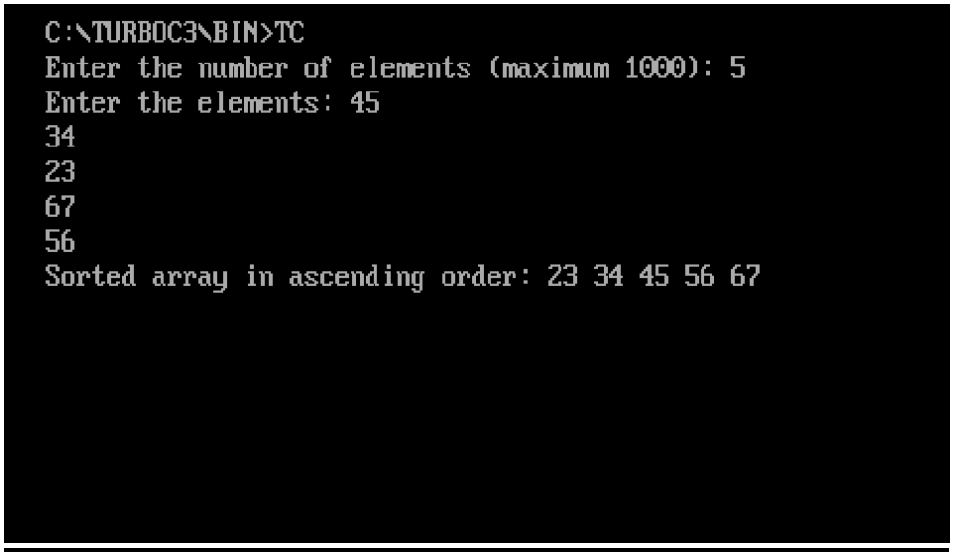

C:\TURBOC3\BIN>TC
Original Array: 12 11 13 5 6 7
Sorted Array: 5 6 7 11 12 13

Experiment 16: Write a program to implement quick sort algorithm to arrange a list of elements in ascending order.

Program:

```
#include <iostream.h>
 void swap(int* a, int* b) {
   int temp = *a;
   *a = *b;
   *b = temp;
 }
  int partition(int arr[], int low, int high) {
   int pivot = arr[high];
   int i = low - 1;
 for (int j = low; j <= high - 1; j++) {
       if (arr[j] <= pivot) {
          i++;
          swap(&arr[i], &arr[j]);
       }
   }
   swap(&arr[i + 1], &arr[high]);
   return i + 1;
 }
 void quickSort(int arr[], int low, int high) {
   if (low < high) {
       int pi = partition(arr, low, high);
       quickSort(arr, low, pi - 1);
       quickSort(arr, pi + 1, high);
   }
 }
  int main() {
   const int maxSize = 1000;
   int arr[maxSize];
      int n;
   cout << "Enter the number of elements (maximum " << maxSize << "): ";
   cin >> n;
   if (n > maxSize) {
       cout << "Number of elements exceeds maximum size. Exiting.\n";
       return 1;
   }
   cout << "Enter the elements: ";
   for (int i = 0; i < n; i++) {
       cin >> arr[i];
   }
   quickSort(arr, 0, n - 1);
   cout << "Sorted array in ascending order: ";
   for ( i = 0; i < n; i++) {
       cout << arr[i] << " ";
   }
   cout << endl;
     return 0;
  }
```

Output:

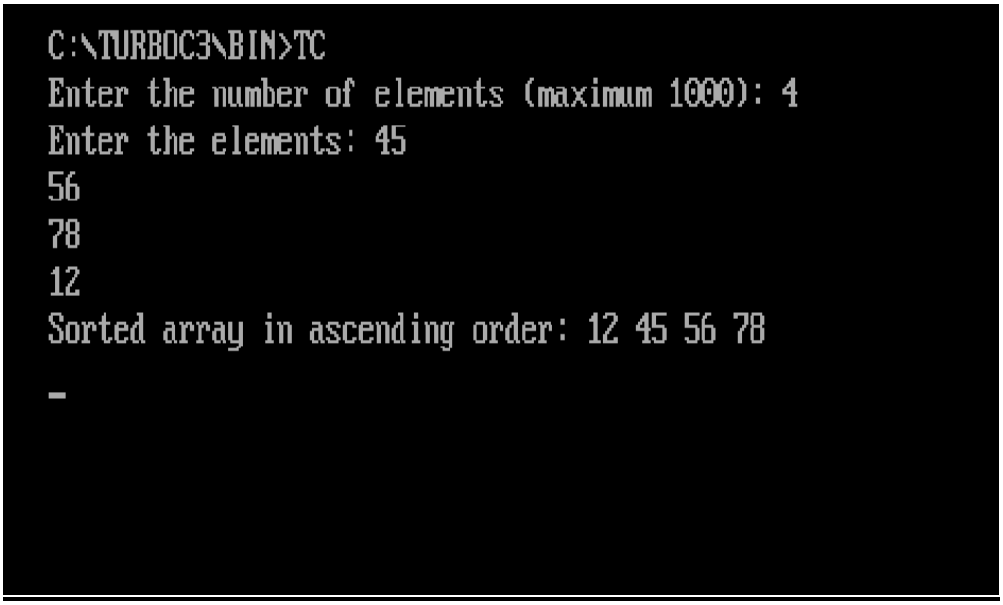

C:\TURBOC3\BIN>TC
Enter the number of elements (maximum 1000): 5
Enter the elements: 45
34
23
67
56
Sorted array in ascending order: 23 34 45 56 67

Experiment 17: Write a program to implement heap sort algorithm to sort a list of integers in ascending order.

Program:

```
#include <iostream.h>
 void swap(int* a, int* b) {
  int temp = *a;
  *a = *b;
  *b = temp;
 }
 void heapify(int arr[], int n, int i) {
  int largest = i;
  int left = 2 * i + 1;
  int right = 2 * i + 2;
  if (left < n && arr[left] > arr[largest])
     largest = left;
  if (right < n && arr[right] > arr[largest])
     largest = right;
  if (largest != i) {
     swap(&arr[i], &arr[largest]);
     heapify(arr, n, largest);
  }
 }
 void heapSort(int arr[], int n) {
   for (int i = n / 2 - 1; i >= 0; i--)
     heapify(arr, n, i);
  for ( i = n - 1; i > 0; i--) {
     swap(&arr[0], &arr[i]);
     heapify(arr, i, 0);
  }
 }
 int main() {
  const int maxSize = 1000;
  int arr[maxSize];
  int n;
  cout << "Enter the number of elements (maximum " << maxSize << "): ";
  cin >> n;
   if (n > maxSize) {
     cout << "Number of elements exceeds maximum size. Exiting.\n";
     return 1;
  }
 cout << "Enter the elements: ";
  for (int i = 0; i < n; i++) {
     cin >> arr[i];
  }
  heapSort(arr, n);
  cout << "Sorted array in ascending order: ";
  for ( i = 0; i < n; i++) {
     cout << arr[i] << " ";
  }
  cout << endl;
    return 0;
  }
```

Output:

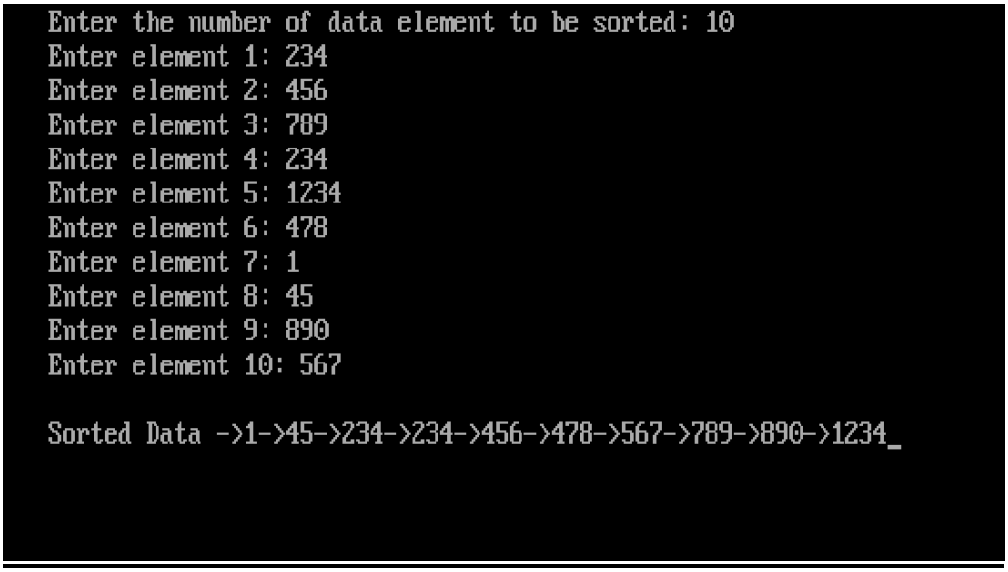

C:\TURBOC3\BIN>TC
Enter the number of elements (maximum 1000): 4
Enter the elements: 45
56
78
12
Sorted array in ascending order: 12 45 56 78

Experiment 18: . Write a program to implement radix sort algorithm to sort a list of integers in ascending order.

Program:

```
#include <iostream.h>
  int getMax(int arr[], int n)
  {
  int max = arr[0];
  for (int i = 1; i < n; i++)
     if (arr[i] > max)
        max = arr[i];
  return max;
  }
  void countSort(int arr[], int n, int exp)
  {
   int output[1000], i, count[10] = {0};
  for (i = 0; i < n; i++)
     count[(arr[i] / exp) % 10]++;
  for (i = 1; i < 10; i++)
     count[i] += count[i - 1];
    for (i = n - 1; i >= 0; i--)
  {
     output[count[(arr[i] / exp) % 10] - 1] = arr[i];
     count[(arr[i] / exp) % 10]--;
  }
  for (i = 0; i < n; i++)
     arr[i] = output[i];
  }
  void radixsort(int arr[], int n)
  {
  int exp, m;
  m = getMax(arr, n);
  for (exp = 1; m / exp > 0; exp *= 10)
     countSort(arr, n, exp);
  }
    int main()
 {
  const int maxSize = 1000;
  int n, i;
  cout << "\nEnter the number of data element to be sorted: ";
  cin >> n;
  int arr[maxSize];
  for (i = 0; i < n; i++)
  {
     cout << "Enter element " << i + 1 << ": ";
     cin >> arr[i];
  }
   radixsort(arr, n);
  cout << "\nSorted Data ";
  for (i = 0; i < n; i++)
  cout << "->" << arr[i];
    return 0;
   }
```

Output:


Enter the number of data element to be sorted: 10
Enter element 1: 234
Enter element 2: 456
Enter element 3: 789
Enter element 4: 234
Enter element 5: 1234
Enter element 6: 478
Enter element 7: 1
Enter element 8: 45
Enter element 9: 890
Enter element 10: 567
Sorted Data ->1->45->234->234->456->478->567->789->890->1234

Experiment 19: Write a program to implement all the functions of a dictionary (ADT) using hashing.

Program :

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
#define SIZE 10
class Dictionary {
private:
  struct Node {
     int key;
     char* value;
     Node* next;
  };
  Node* table[SIZE];
  int hashFunction(int key) {
     return key % SIZE;
  }
public:
  Dictionary() {
     for (int i = 0; i < SIZE; i++) {
        table[i] = NULL;
     }
  }
  void insert(int key, char* value) {
     int index = hashFunction(key);
     Node* newNode = new Node;
     newNode->key = key;
     newNode->value = new char[strlen(value) + 1];
     strcpy(newNode->value, value);
     newNode->next = table[index];
     table[index] = newNode;
  }
  char* search(int key) {
     int index = hashFunction(key);
     Node* current = table[index];
     while (current != NULL) {
        if (current->key == key) {
           return current->value;
        }
        current = current->next;
     }
     return "Key not found";
  }
  void remove(int key) {
     int index = hashFunction(key);
     Node* current = table[index];
     Node* prev = NULL;
     while (current != NULL) {
        if (current->key == key) {
           if (prev == NULL) {
              table[index] = current->next;
           } else {
```

```
                prev->next = current->next;
            }
            delete[] current->value;
            delete current;
            return;
        }
        prev = current;
        current = current->next;
    }
  }
   void display() {
    for (int i = 0; i < SIZE; i++) {
        cout << "Bucket " << i << ": ";
        Node* current = table[i];
        while (current != NULL) {
            cout << "(" << current->key << ", " << current->value << ") ";
            current = current->next;
        }
        cout << endl;
      }
  }
};
int main() {
 clrscr();
 Dictionary dict;
 dict.insert(1, "One");
 dict.insert(11, "Eleven");
 dict.insert(21, "Twenty-one");
 cout << "Dictionary Contents:" << endl;
 dict.display();
 cout << "Value for key 11: " << dict.search(11) << endl;
 dict.remove(11);
 cout << "Dictionary Contents after removing key 11:" << endl;
 dict.display();
   getch();
   return 0;
}
```

Output:

```
Dictionary Contents:
Bucket 0:
Bucket 1: (21, Twenty-one) (11, Eleven) (1, One)
Bucket 2:
Bucket 3:
Bucket 4:
Bucket 5:
Bucket 6:
Bucket 7:
Bucket 8:
Bucket 9:
Value for key 11: Eleven
Dictionary Contents after removing key 11:
Bucket 0:
Bucket 1: (21, Twenty-one) (1, One)
Bucket 2:
Bucket 3:
Bucket 4:
Bucket 5:
Bucket 6:
Bucket 7:
Bucket 8:
Bucket 9:
```

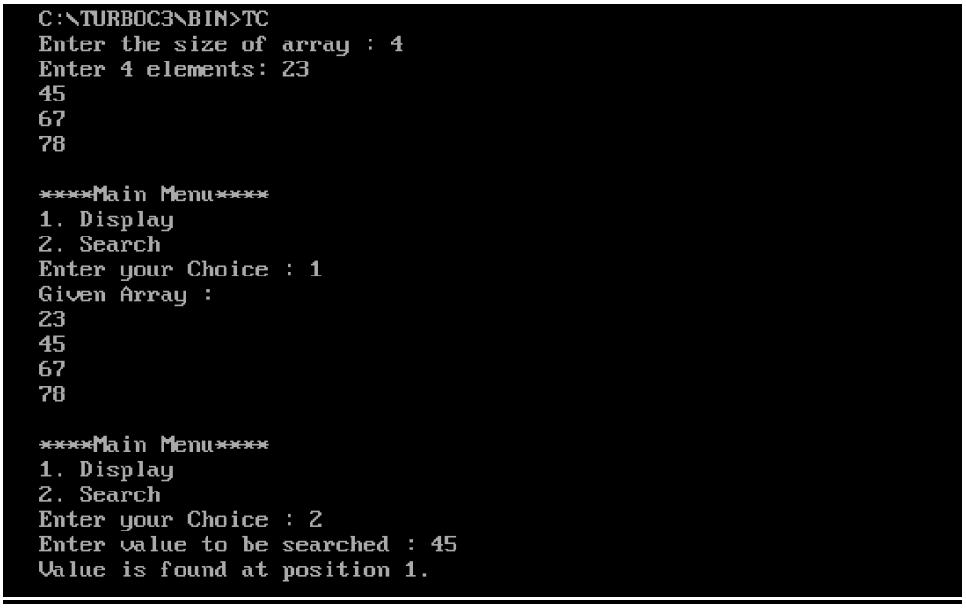
Experiment 20: Write a program to search the element using sequential search.

Program :

```
#include<iostream.h>
#include<conio.h>

 int size, val;
 int arr[20];
  void disp(int size);
  void search(int val, int size);
 int main() {
 int i, ch;
 cout << "Enter the size of array : ";
 cin >> size;
 cout << "Enter " << size << " elements: ";
 for (i = 0; i < size; i++) {
    cin >> arr[i];
  }
 do {
    cout << "\n****Main Menu****\n";
    cout << "1. Display\n";
    cout << "2. Search\n";
    cout << "Enter your Choice : ";
    cin >> ch;
    switch (ch) {
       case 1:
          disp(size);
          break;
       case 2:
          cout << "Enter value to be searched : ";
          cin >> val;
          search(val, size);
          break;
    }
 } while (ch != 2);
 getch();
 return 0;
 }
void search(int val, int size) {
 int i;
 for (i = 0; i < size; i++) {
    if (arr[i] == val) {
       cout << "Value is found at position " << i << ".";
       return;
    }
 }
 cout << "Value is not found.";
 }
void disp(int size) {
  cout << "Given Array :\n";
 for (int i = 0; i < size; i++) {
    cout << arr[i] << endl;
   }
```

```
}
```

Output:


C:\TURBOC3\BIN>TC
Enter the size of array : 4
Enter 4 elements: 23
45
67
78
****Main Menu****
1. Display
2. Search
Enter your Choice : 1
Given Array :
23
45
67
78
****Main Menu****
1. Display
2. Search
Enter your Choice : 2
Enter value to be searched : 45
Value is found at position 1.

Experiment 21: Write a program to search for a key element in a list of sorted elements using binary search.

Program:

```
#include <iostream.h>
#include <conio.h>
#define MAX_SIZE 100

 int binarySearch(int arr[], int size, int key) {

 int low = 0;
 int high = size - 1;
 while (low <= high) {
    int mid = low + (high - low) / 2;
    if (arr[mid] == key)
       return mid;
    else if (arr[mid] < key)
       low = mid + 1;
    else
       high = mid - 1;
  }
 return -1;
}
  int main() {
     clrscr();
 int size;
 cout << "Enter the size of the array (up to " << MAX_SIZE << "): ";
 cin >> size;
 if (size > MAX_SIZE) {
    cout << "Array size exceeds maximum size. Exiting." << endl;
    return 1;
 }
 int arr[MAX_SIZE];
 cout << "Enter " << size << " sorted elements: ";
 for (int i = 0; i < size; i++) {
    cin >> arr[i];
 }
 int key;
 cout << "Enter the key element to search: ";
 cin >> key;
 int index = binarySearch(arr, size, key);
 if (index != -1)
    cout << "Element found at index " << index << endl;
 else
    cout << "Element not found" << endl;
   getch();
  return 0;
}
```

Output:

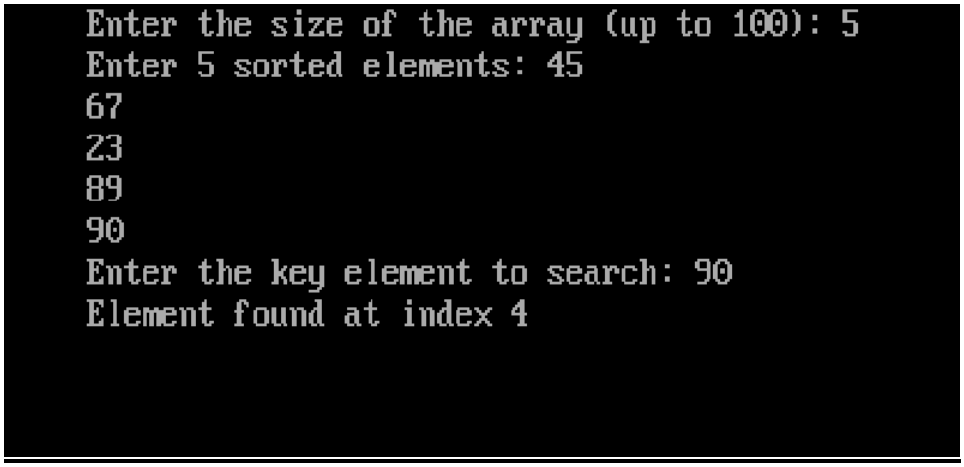
```
Enter the size of the array (up to 100): 4
Enter 4 sorted elements: 34
56
67
23
Enter the key element to search: 67
Element found at index 2
_
```

Experiment 22: Write a program to search for a key element in a list of sorted elements using linear search.

Program:

```
#include <iostream.h>
#include <conio.h>
#define MAX_SIZE 100
int linearSearch(int arr[], int size, int key) {
 for (int i = 0; i < size; i++) {
    if (arr[i] == key)
       return i;
    else if (arr[i] > key)
       return -1;
 }
 return -1;
}
  int main() {
   clrscr();
  int size;
 cout << "Enter the size of the array (up to " << MAX_SIZE << "): ";
 cin >> size;
 if (size > MAX_SIZE) {
    cout << "Array size exceeds maximum size. Exiting." << endl;
    return 1;
 }
   int arr[MAX_SIZE];
     cout << "Enter " << size << " sorted elements: ";
 for (int i = 0; i < size; i++) {
    cin >> arr[i];
 }
 int key;
 cout << "Enter the key element to search: ";
 cin >> key;
 int index = linearSearch(arr, size, key);
 if (index != -1)
    cout << "Element found at index " << index << endl;
 else
    cout << "Element not found" << endl;
    getch();
    return 0;
 }
```

Output:

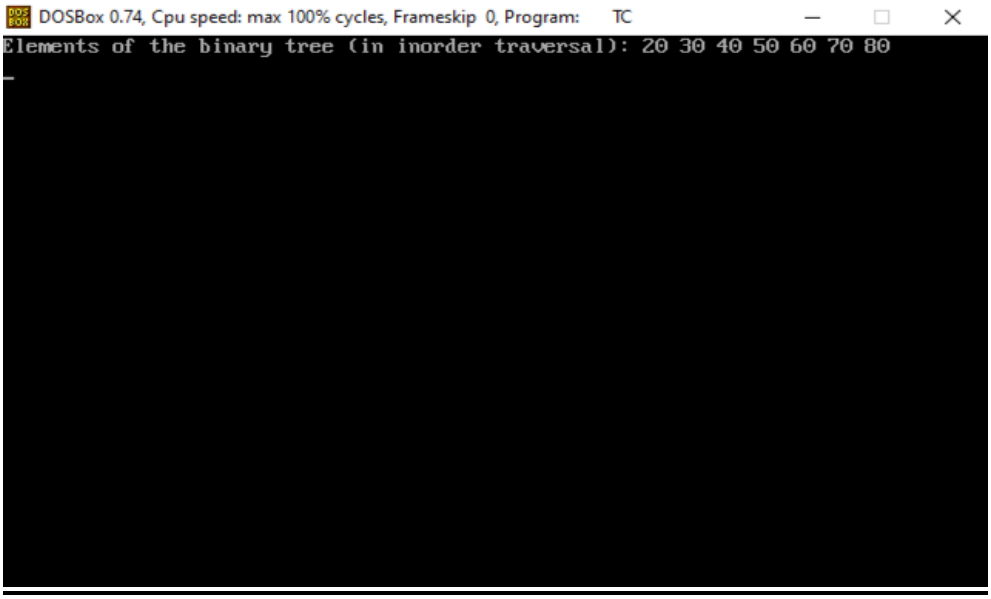

Enter the size of the array (up to 100): 5
Enter 5 sorted elements: 45
67
23
89
90
Enter the key element to search: 90
Element found at index 4

Experiment 23: Write a program to create the tree and display the elements.

Program:

```
#include <iostream.h>
#include <conio.h>

        struct TreeNode {
      int data;
     TreeNode* left;
     TreeNode* right;
  };
  TreeNode* createNode(int value) {
     TreeNode* newNode = new TreeNode;
     newNode->data = value;
     newNode->left = NULL;
     newNode->right = NULL;
   return newNode;
  }
   TreeNode* insert(TreeNode* root, int value) {
  if (root == NULL) {
     return createNode(value);
  }
  if (value < root->data) {
     root->left = insert(root->left, value);
  } else if (value > root->data) {
     root->right = insert(root->right, value);
  }
      return root;
   }
  void inorderTraversal(TreeNode* root) {
        if (root != NULL) {
     I    norderTraversal(root->left);
          cout << root->data << " ";
     inorderTraversal(root->right);
     }
   }
    int main() {
        clrscr();
   TreeNode* root = NULL;
  root = insert(root, 50);
  root = insert(root, 30);
  root = insert(root, 20);
  root = insert(root, 40);
  root = insert(root, 70);
  root = insert(root, 60);
  root = insert(root, 80);

  cout << "Elements of the binary tree (in inorder traversal): ";
  inorderTraversal(root);
  cout << endl;
  getch();
  return 0;
}
```

Output:


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Elements of the binary tree (in inorder traversal): 20 30 40 50 60 70 80

Experiment 24: Write a program that uses functions to perform the followings:

a. Create a binary tree of integers.

b. Traverse the same binary search tree non recursively in inorder, postorder and preorder

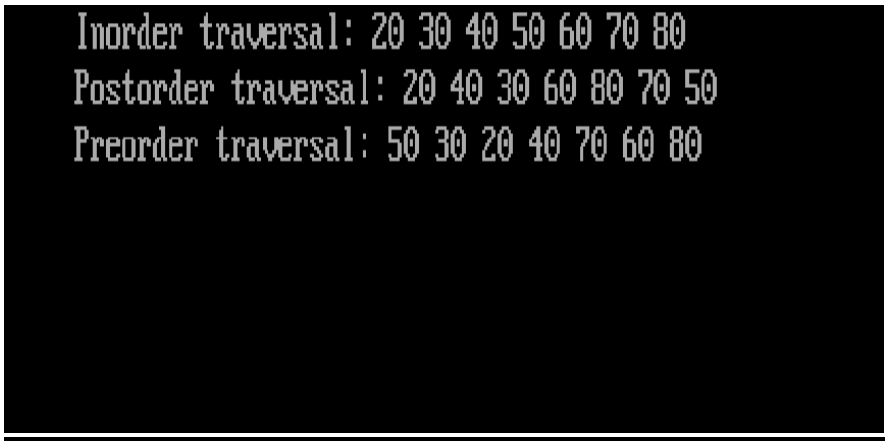
program :

```
#include <iostream.h>
#include <conio.h>
 struct TreeNode {
  int data;
  TreeNode* left;
  TreeNode* right;
};
struct Stack {
  int top;
  int capacity;
TreeNode** array;
};
Stack* createStack(int capacity) {
  Stack* stack = new Stack;
  stack->capacity = capacity;
  stack->top = -1;
  stack->array = new TreeNode*[capacity];
  return stack;
}
  int isEmpty(Stack* stack) {
  return stack->top == -1;
}
void push(Stack* stack, TreeNode* item) {
  stack->array[++stack->top] = item;
}
 TreeNode* pop(Stack* stack) {
  if (isEmpty(stack))
      return NULL;
  return stack->array[stack->top--];
}
  TreeNode* createNode(int value) {
  TreeNode* newNode = new TreeNode;
  newNode->data = value;
  newNode->left = NULL;
  newNode->right = NULL;
  return newNode;
}
  TreeNode* insert(TreeNode* root, int value) {
  if (root == NULL) {
      return createNode(value);
  }
  if (value < root->data) {
      root->left = insert(root->left, value);
  } else if (value > root->data) {
      root->right = insert(root->right, value);
   }
  return root;
```

```
}
  void inorderTraversal(TreeNode* root) {
  if (root == NULL)
     return;
   Stack* stack = createStack(100);
  TreeNode* current = root;
  while (current != NULL || !isEmpty(stack)) {
     while (current != NULL) {
        push(stack, current);
        current = current->left;
     }
     current = pop(stack);
     cout << current->data << " ";
     current = current->right;
  }
 }
 void postorderTraversal(TreeNode* root) {
  if (root == NULL)
     return;
  Stack* stack1 = createStack(100);
  Stack* stack2 = createStack(100);
  push(stack1, root);
  while (!isEmpty(stack1)) {
     TreeNode* current = pop(stack1);
     push(stack2, current);
     if (current->left)
        push(stack1, current->left);
     if (current->right)
        push(stack1, current->right);
  }
while (!isEmpty(stack2)) {
     cout << pop(stack2)->data << " ";
  }
 }
 void preorderTraversal(TreeNode* root) {
  if (root == NULL)
     return;
  Stack* stack = createStack(100);
  push(stack, root);
  while (!isEmpty(stack)) {
     TreeNode* current = pop(stack);
     cout << current->data << " ";
     if (current->right)
        push(stack, current->right);
     if (current->left)
        push(stack, current->left);
  }
 }
  int main() {
  clrscr();
  TreeNode* root = NULL;
```

```
  root = insert(root, 50);
  root = insert(root, 30);
  root = insert(root, 20);
  root = insert(root, 40);
  root = insert(root, 70);
  root = insert(root, 60);
  root = insert(root, 80);

  cout << "Inorder traversal: ";
  inorderTraversal(root);
  cout << endl;
  cout << "Postorder traversal: ";
  postorderTraversal(root);
  cout << endl;
  cout << "Preorder traversal: ";
  preorderTraversal(root);
  cout << endl;
  getch();
  return 0;
}
```

Output:

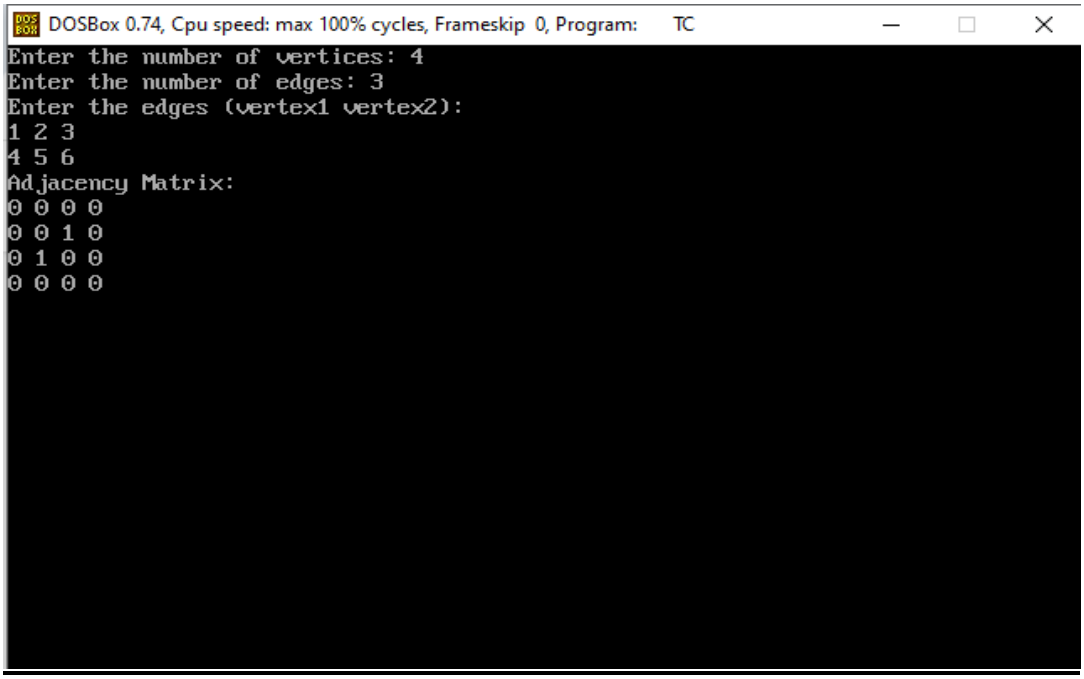


Experiment 25 : Write a program to generate the adjacency matrix.

Program:

```
#include <iostream.h>
#include <conio.h>
  const int MAX_SIZE = 100;
  void generateAdjacencyMatrix(int matrix[][MAX_SIZE], int numVertices) {
  for (int i = 0; i < numVertices; i++) {
     for (int j = 0; j < numVertices; j++) {
        matrix[i][j] = 0;
     }
  }
  int numEdges;
  cout << "Enter the number of edges: ";
  cin >> numEdges;
  cout << "Enter the edges (vertex1 vertex2):" << endl;
  for (i = 0; i < numEdges; i++) {
     int vertex1, vertex2;
     cin >> vertex1 >> vertex2;
     matrix[vertex1][vertex2] = 1;
     matrix[vertex2][vertex1] = 1;
  }
}
  void displayAdjacencyMatrix(int matrix[][MAX_SIZE], int numVertices) {
  cout << "Adjacency Matrix:" << endl;
  for (int i = 0; i < numVertices; i++) {
     for (int j = 0; j < numVertices; j++) {
        cout << matrix[i][j] << " ";
     }
     cout << endl;
  }
}
 int main() {
  clrscr();
  int numVertices;
  cout << "Enter the number of vertices: ";
  cin >> numVertices;
  int adjacencyMatrix[MAX_SIZE][MAX_SIZE];
  generateAdjacencyMatrix(adjacencyMatrix, numVertices);
  displayAdjacencyMatrix(adjacencyMatrix, numVertices);
  getch();
  return 0;
}
```

Output:



References:

1. **https://www.javatpoint.com/cpp-program**
2. **https://www.geeksforgeeks.org/cpp-programming-examples**
3. **https://www.w3schools.com/cpp/cpp_intro.asp**