S.No: 1	Exp. Name: ***Write a C program to Sort the given elements in Ascending order using Bubble Sort***	Date: 2023-10-28

Aim:

Write a program to **sort** the given elements using `Bubble sort technique` (`Ascending order`) .

Source Code:

bubbleSort.c

```c
#include <stdio.h>

void bubbleSort(int arr[], int n) {
        int temp, swapped;

        do {
            swapped = 0;

            for (int i = 1; i < n; i++) {
                if (arr[i - 1] > arr[i]) {
                temp = arr[i - 1];
                arr[i - 1] = arr[i];
                arr[i] = temp;
                swapped = 1;

                }

            }

        } while (swapped);
}

int main() {
    int n;

    printf("n : ");
    scanf("%d", &n);

    int arr[n];

    for (int i = 0; i < n; i++) {
        printf("a[%d] = ", i);
        scanf("%d", &arr[i]);
    }

    printf("Before sorting : \n");
    for (int i = 0; i < n; i++) {
        printf("a[%d] = %d\n", i, arr[i]);
    }

    bubbleSort(arr, n);

    printf("After sorting : \n");
    for (int i = 0; i < n; i++) {
        printf("a[%d] = %d\n", i, arr[i]);
    }

    return 0;

}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
n :
5
a[0] =
2
a[1] =
7
a[2] =
6
a[3] =
4
a[4] =
1
Before sorting :
a[0] = 2
a[1] = 7
a[2] = 6
a[3] = 4
a[4] = 1
After sorting :
a[0] = 1
a[1] = 2
a[2] = 4
a[3] = 6
a[4] = 7

Test Case - 2
User Output
n :
4
a[0] =
28
a[1] =
34
a[2] =
26
a[3] =
29
Before sorting :
a[0] = 28
a[1] = 34
a[2] = 26
a[3] = 29
After sorting :
a[0] = 26
a[1] = 28
a[2] = 29

```
a[3] = 34
```

Test Case - 3

User Output

```
n :
5
a[0] =
-45
a[1] =
-12
a[2] =
-77
a[3] =
-21
a[4] =
-100
Before sorting :
a[0] = -45
a[1] = -12
a[2] = -77
a[3] = -21
a[4] = -100
After sorting :
a[0] = -100
a[1] = -77
a[2] = -45
a[3] = -21
a[4] = -12
```

S.No: 2	Exp. Name: ***Write a C program to Sort the elements using Insertion Sort Technique***	**Date: 2023-10-28**

Aim:

Write a program to **sort** the given elements using `Insertion sort technique` (`Ascending order`).

Source Code:

insertionSort.c

```
#include<stdio.h>
void main() {
        int a[20], i, n, j, temp;
        printf("n = ");
        scanf("%d", &n);
        // Write the for loop to read array elements
        for (i = 0; i < n; i++) {
         printf("a[%d] = ", i);
                 scanf("%d", &a[i]);
        }

        printf("Before sorting : \n");
        // Write the for loop to display array elements before sorting
        for (i = 0; i < n; i++) {
                printf("a[%d] = %d\n", i, a[i]);
        }
        //Write the code to sort elements
        for (i = 1; i < n; i++) {
                 temp = a[i];
                 j = i - 1;
                 while (j >= 0 && a[j] > temp) {
                 a[j + 1] = a[j];
                 j--;
             }
             a[j + 1] = temp;
    }
        printf("After sorting : \n");
        // Write the for loop to display array elements after sorting
    for (i = 0; i < n; i++) {
            printf("a[%d] = %d\n", i, a[i]);
  }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
n =
5
a[0] =
12
a[1] =

4
a[2] =
27
a[3] =
37
a[4] =
41
Before sorting :
a[0] = 12
a[1] = 4
a[2] = 27
a[3] = 37
a[4] = 41
After sorting :
a[0] = 4
a[1] = 12
a[2] = 27
a[3] = 37
a[4] = 41

Test Case - 2
User Output
n =
7
a[0] =
3
a[1] =
8
a[2] =
2
a[3] =
1
a[4] =
4
a[5] =
9
a[6] =
4
Before sorting :
a[0] = 3
a[1] = 8
a[2] = 2
a[3] = 1
a[4] = 4
a[5] = 9
a[6] = 4
After sorting :
a[0] = 1
a[1] = 2

a[2] = 3
a[3] = 4
a[4] = 4
a[5] = 8
a[6] = 9

Test Case - 3
User Output
n =
5
a[0] =
-36
a[1] =
-100
a[2] =
-43
a[3] =
54
a[4] =
0
Before sorting :
a[0] = -36
a[1] = -100
a[2] = -43
a[3] = 54
a[4] = 0
After sorting :
a[0] = -100
a[1] = -43
a[2] = -36
a[3] = 0
a[4] = 54

S.No: 3	Exp. Name: ***Write a C program to Sort given elements using Selection sort largest element method***	**Date: 2023-10-28**

Aim:

Write a program to **sort** the given array elements using `Selection sort largest element` method(`Ascending order`).

Source Code:

selectionSort.c

```c
#include <stdio.h>

void selectionSort(int arr[], int n) {
         int i, j, minIndex, temp;

         for (i = 0; i < n - 1; i++) {
             minIndex = i;

             for (j = i + 1; j < n; j++) {
                 if (arr[j] < arr[minIndex]) {
                         minIndex = j;
                 }
             }

             temp = arr[i];
             arr[i] = arr[minIndex];
             arr[minIndex] = temp;
    }
}

int main() {
        int n;

        printf("n = ");
        scanf("%d", &n);

    int arr[n];

    for (int i = 0; i < n; i++) {
            printf("a[%d] = ");
            scanf("%d", &arr[i]);
        }

        printf("Before sorting : \n");
        for (int i = 0; i < n; i++) {
             printf("a[%d] = %d\n", i, arr[i]);
    }

    selectionSort(arr, n);

    printf("After sorting : \n");
    for (int i = 0; i < n; i++) {
        printf("a[%d] = %d\n", i, arr[i]);
    }

    return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
n =

7
a[0] =
5
a[1] =
2
a[2] =
3
a[3] =
4
a[4] =
6
a[5] =
1
a[6] =
9
Before sorting :
a[0] = 5
a[1] = 2
a[2] = 3
a[3] = 4
a[4] = 6
a[5] = 1
a[6] = 9
After sorting :
a[0] = 1
a[1] = 2
a[2] = 3
a[3] = 4
a[4] = 5
a[5] = 6
a[6] = 9

Test Case - 2
User Output
n =
5
a[0] =
45
a[1] =
25
a[2] =
67
a[3] =
89
a[4] =
44
Before sorting :
a[0] = 45
a[1] = 25

a[2] = 67
a[3] = 89
a[4] = 44
After sorting :
a[0] = 25
a[1] = 44
a[2] = 45
a[3] = 67
a[4] = 89

Test Case - 3
User Output
n =
4
a[0] =
-9
a[1] =
-54
a[2] =
-12
a[3] =
-369
Before sorting :
a[0] = -9
a[1] = -54
a[2] = -12
a[3] = -369
After sorting :
a[0] = -369
a[1] = -54
a[2] = -12
a[3] = -9

S.No: 4	Exp. Name: ***Write a C program to sort given elements using Selection sort smallest element method***	**Date: 2023-10-28**

Aim:

Write a program to **sort** the given array elements using `Selection sort smallest element` method(`Ascending order`).

Source Code:

selctionSmallest.c

```c
#include <stdio.h>

void selectionSort(int arr[], int n) {
        int i, j, minIndex, temp;

    for (i = 0; i < n - 1; i++) {
            minIndex = i;

            for (j = i + 1; j < n; j++) {
                if (arr[j] < arr[minIndex]) {
                        minIndex = j;
                }
            }

            temp = arr[i];
            arr[i] = arr[minIndex];
            arr[minIndex] = temp;
    }
}

int main() {
        int n;

        printf("n = ");
        scanf("%d", &n);

        int arr[n];

    for (int i = 0; i < n; i++) {
             printf("a[%d] = ", i);
             scanf("%d", &arr[i]);
    }

    printf("Before sorting : \n");
    for (int i = 0; i < n; i++) {
            printf("a[%d] = %d\n", i, arr[i]);
}

    selectionSort(arr, n);

    printf("After sorting : \n");
    for (int i = 0; i < n; i++) {
        printf("a[%d] = %d\n", i, arr[i]);
    }

    return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
n =

5
a[0] =
15
a[1] =
45
a[2] =
1
a[3] =
20
a[4] =
30
Before sorting :
a[0] = 15
a[1] = 45
a[2] = 1
a[3] = 20
a[4] = 30
After sorting :
a[0] = 1
a[1] = 15
a[2] = 20
a[3] = 30
a[4] = 45

Test Case - 2
User Output
n =
4
a[0] =
-15
a[1] =
-12
a[2] =
-48
a[3] =
-79
Before sorting :
a[0] = -15
a[1] = -12
a[2] = -48
a[3] = -79
After sorting :
a[0] = -79
a[1] = -48
a[2] = -15
a[3] = -12

Test Case - 3

User Output
n =
5
a[0] =
34
a[1] =
68
a[2] =
95
a[3] =
41
a[4] =
23
Before sorting :
a[0] = 34
a[1] = 68
a[2] = 95
a[3] = 41
a[4] = 23
After sorting :
a[0] = 23
a[1] = 34
a[2] = 41
a[3] = 68
a[4] = 95

S.No: 5	Exp. Name: ***Write a C program to Sort given elements using Merge sort***	Date: 2023-10-28

Aim:

Write a program to `sort` (`Ascending order`) the given elements using `merge sort` technique.

At the time of execution, the program should print the message on the console as:

```
Enter array size :
```

For example, if the user gives the **input** as:

```
Enter array size : 5
```

Next, the program should print the following message on the console as:

```
Enter 5 elements :
```

if the user gives the **input** as:

```
Enter 5 elements : 34 67 12 45 22
```

then the program should **print** the result as:

```
Before sorting the elements are : 34 67 12 45 22
After sorting the elements are : 12 22 34 45 67
```

Note: Do use the **printf()** function with a **newline** character (`\n`).

Source Code:

MergeSortMain.c

```
#include <stdio.h>
#include "MergeSortFunctions.c"
void main() {
        int arr[15], i, n;
        printf("Enter array size : ");
        scanf("%d", &n);
        printf("Enter %d elements : ", n);
        for (i = 0; i < n; i++) {
                scanf("%d", &arr[i]);
        }
        printf("Before sorting the elements are : ");
        display(arr, n);
        splitAndMerge(arr, 0, n - 1);
        printf("After sorting the elements are : ");
        display(arr, n);
}
```

MergeSortFunctions.c

```c
#include <stdio.h>

void merge(int arr[], int l, int m, int r) {
         int n1 = m - l + 1;
         int n2 = r - m;

         int L[n1], R[n2];

         for (int i = 0; i < n1; i++)
             L[i] = arr[l + i];
         for (int i = 0; i < n2; i++)
             R[i] = arr[m + 1 + i];

         int i = 0, j = 0, k = l;
         while (i < n1 && j < n2) {
         if (L[i] <= R[j]) {
                 arr[k] = L[i];
                 i++;
         } else {
                 arr[k] = R[j];
                 j++;
         }
         k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }

}

void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

void display(int arr[], int n) {
         for (int i = 0; i < n; i++) {
                 printf("%d ", arr[i]);
         }
         printf("\n");
}
```

```
            int m = l + (r - l) / 2;
            splitAndMerge(arr, l, m);
            splitAndMerge(arr, m + 1, r);
            merge(arr, l, m, r);
            }
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter array size :
5
Enter 5 elements :
34 67 12 45 22
Before sorting the elements are : 34 67 12 45 22
After sorting the elements are : 12 22 34 45 67

Test Case - 2
User Output
Enter array size :
8
Enter 8 elements :
77 55 22 44 99 33 11 66
Before sorting the elements are : 77 55 22 44 99 33 11 66
After sorting the elements are : 11 22 33 44 55 66 77 99

Test Case - 3
User Output
Enter array size :
5
Enter 5 elements :
-32 -45 -67 -46 -14
Before sorting the elements are : -32 -45 -67 -46 -14
After sorting the elements are : -67 -46 -45 -32 -14

S.No: 6	Exp. Name: ***Write a C program to Sort given elements using Quick sort***	Date: 2023-10-28

Aim:

Write a program to `sort` (`Ascending order`) the given elements using `quick sort` technique.

Note: Pick the first element as pivot. You will not be awarded marks if you do not follow this instruction.

At the time of execution, the program should print the message on the console as:

```
Enter array size :
```

For example, if the user gives the **input** as:

```
Enter array size : 5
```

Next, the program should print the following message on the console as:

```
Enter 5 elements :
```

if the user gives the **input** as:

```
Enter 5 elements : 34 67 12 45 22
```

then the program should **print** the result as:

```
Before sorting the elements are : 34 67 12 45 22
After sorting the elements are : 12 22 34 45 67
```

Note: Do use the **printf()** function with a **newline** character (`\n`).

Source Code:

QuickSortMain.c

```
#include <stdio.h>
#include "QuickSortFunctions.c"
void main() {
        int arr[15], i, n;
        printf("Enter array size : ");
        scanf("%d", &n);
        printf("Enter %d elements : ", n);
        for (i = 0; i < n; i++) {
                scanf("%d", &arr[i]);
        }
        printf("Before sorting the elements are : ");
        display(arr, n);
        quickSort(arr, 0, n - 1);
        printf("After sorting the elements are : ");
        display(arr, n);
}
```

QuickSortFunctions.c

```c
#include <stdio.h>

void display(int arr[], int n) {
        for (int i = 0; i < n; i++) {
            printf("%d ", arr[i]);
    }
        printf("\n");
}

int partition(int arr[], int low, int high) {
        int pivot = arr[low];
        int i = low;
        int j = high;

        while (i < j) {
            while (arr[i] <= pivot && i < high) {
                i++;
            }
            while (arr[j] > pivot) {
            j--;
        }
        if (i < j) {
            int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
        }
    }

    arr[low] = arr[j];
    arr[j] = pivot;

    return j;
 }

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pivotIndex = partition(arr, low, high);
        quickSort(arr, low, pivotIndex - 1);
        quickSort(arr, pivotIndex + 1, high);
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter array size :
5
Enter 5 elements :
34 67 12 45 22
Before sorting the elements are : 34 67 12 45 22
After sorting the elements are : 12 22 34 45 67

Test Case - 2
User Output
Enter array size :
8
Enter 8 elements :
77 55 22 44 99 33 11 66
Before sorting the elements are : 77 55 22 44 99 33 11 66
After sorting the elements are : 11 22 33 44 55 66 77 99

Test Case - 3
User Output
Enter array size :
5
Enter 5 elements :
-32 -45 -67 -46 -14
Before sorting the elements are : -32 -45 -67 -46 -14
After sorting the elements are : -67 -46 -45 -32 -14

S.No: 7	Exp. Name: ***Write a C program to Search an element using Linear Search process***	Date: 2023-10-29

Aim:

Write a program to **search** a key element within the given array of elements using `Linear search` process.

Source Code:

linearSearch.c

```
#include <stdio.h>

int linearSearch(int arr[], int n, int key) {
        for (int i = 0; i < n; i++) {
             if (arr[i] == key) {
                 return i;
            }
        }
        return -1;
}

int main() {
        int n, key;

        printf("n = ");
        scanf("%d", &n);

        int arr[n];

        for (int i = 0; i < n; i++) {
                printf("a[%d] = ", i);
        scanf("%d", &arr[i]);
    }

    printf("Search key : ");
    scanf("%d", &key);

    int result = linearSearch(arr, n, key);

    if (result != -1) {
        printf("Key %d is found at position %d.\n", key, result);
    } else {
        printf("Key %d is not found.\n", key);
    }

    return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
n =

4
a[0] =
153
a[1] =
264
a[2] =
357
a[3] =
598
Search key :
100
Key 100 is not found.

Test Case - 2
User Output
n =
5
a[0] =
-15
a[1] =
-24
a[2] =
-36
a[3] =
-11
a[4] =
-20
Search key :
-11
Key -11 is found at position 3.

Test Case - 3
User Output
n =
5
a[0] =
24
a[1] =
36
a[2] =
11
a[3] =
45
a[4] =
28
Search key :
11

```
Key 11 is found at position 2.
```

S.No: 8	Exp. Name: ***Write a C program to Search an element using Binary Search process***	**Date: 2023-10-29**

Aim:

Write a program to **search** a key element in the given array of elements using `Binary search`.

Source Code:

`binarySearch.c`

```c
#include <stdio.h>

int binary_search(int arr[], int n, int key) {
        int left = 0, right = n - 1;
        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (arr[mid] == key)
                return mid;
            if (arr[mid] < key)
                left = mid + 1;
                else
                right = mid - 1;

        }
        return -1;

}

void bubble_sort(int arr[], int n) {
        int temp;
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }

            }
        }

}

int main() {
        int n, key;
        printf("n = ");
        scanf("%d", &n);
        int arr[n];
        for (int i = 0; i < n; i++) {
        printf("a[%d] = ", i);
        scanf("%d", &arr[i]);
    }
    printf("Search key = ");
    scanf("%d", &key);

    bubble_sort(arr, n);

    printf("After sorting :\n");
    for (int i = 0; i < n; i++) {
        printf("a[%d] = %d\n", i, arr[i]);
    }

    int result = binary_search(arr, n, key);
    if (result == -1) {
        printf("Key %d is not found in the array.\n", key);
```

```
    }

    return 0;
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
n =
5
a[0] =
15
a[1] =
29
a[2] =
67
a[3] =
10
a[4] =
23
Search key =
10
After sorting :
a[0] = 10
a[1] = 15
a[2] = 23
a[3] = 29
a[4] = 67
Key 10 is found at position 0.

Test Case - 2
User Output
n =
4
a[0] =
-24
a[1] =
-36
a[2] =
-10
a[3] =
-87
Search key =
-10

After sorting :
a[0] = -87
a[1] = -36
a[2] = -24
a[3] = -10
Key -10 is found at position 3.

Test Case - 3
User Output
n =
5
a[0] =
2
a[1] =
3
a[2] =
4
a[3] =
1
a[4] =
5
Search key =
9
After sorting :
a[0] = 1
a[1] = 2
a[2] = 3
a[3] = 4
a[4] = 5
Key 9 is not found in the array.

S.No: 9	Exp. Name: ***Write a C program to implement different Operations on Stack using Arrays.***	**Date: 2023-10-29**

Aim:

Write a C program to implement stack operations using **arrays**.

Source Code:

StackUsingArray.c

```
#include <stdio.h>
#include <stdlib.h>
#define STACK_MAX_SIZE 10
#include "StackOperations.c"

int main() {
        int op, x;
        while(1) {
                printf("1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit\n");
                printf("Enter your option : ");
                scanf("%d", &op);
                switch(op) {
                        case 1:
                                printf("Enter element : ");
                                scanf("%d", &x);
                                push(x);
                                break;
                        case 2:
                                pop();
                                break;
                        case 3:
                                display();
                                break;
                        case 4:
                                isEmpty();
                                break;
                        case 5:
                                peek();
                                break;
                        case 6:
                                exit(0);
                }
        }
}
```

StackOperations.c

```
#include <stdio.h>

#define STACK_MAX_SIZE 10

int stack[STACK_MAX_SIZE];
int top = -1;

void push(int element) {
        if (top == STACK_MAX_SIZE - 1) {
            printf("Stack is overflow.\n");
        } else {
            stack[++top] = element;
            printf("Successfully pushed.\n");
        }
}

void pop() {
        if (top == -1) {
            printf("Stack is underflow.\n");
        } else {
            printf("Popped value = %d\n", stack[top--]);
        }
}

void display() {
        if (top == -1) {
            printf("Stack is empty.\n");
    } else {
            printf("Elements of the stack are : ");
            for (int i = top; i >= 0; i--) {
                printf("%d ", stack[i]);
            }
            printf("\n");
   }
}

int isEmpty() {
        if (top == -1) {
            printf("Stack is empty.\n");
            return 1;
        } else {
            printf("Stack is not empty.\n");
            return 0;
        }
}

int peek() {
        if (top == -1) {
            printf("Stack is underflow.\n");
            return -1;
        } else {
            printf("Peek value = %d\n", stack[top]);
            return stack[top];
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
10
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
20
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
30
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
3
Elements of the stack are : 30 20 10
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
5
Peek value = 30
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Popped value = 30
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Popped value = 20
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
3
Elements of the stack are : 10
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
5
Peek value = 10
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
4
Stack is not empty.

1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Popped value = 10
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
3
Stack is empty.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
4
Stack is empty.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
6

Test Case - 2
User Output
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
1
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
2
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
3
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
4
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
5
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit

Enter your option :
1
Enter element :
6
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
7
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
8
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
9
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
10
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
11
Stack is overflow.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
6

Test Case - 3
User Output
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
5
Stack is underflow.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
6

S.No: 10	Exp. Name: ***Write a C program to implement different Operations on Queue using Array representation***	Date: 2023-10-29

Aim:

Write a program to implement queue using **arrays**.

Source Code:

QueueUsingArray.c

```
#include <conio.h>
#include <stdio.h>
#include "QueueOperations.c"
int main() {
        int op, x;
        while(1) {
                printf("1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit\n");
                printf("Enter your option : ");
                scanf("%d",&op);
                switch(op) {
                        case 1:
                                printf("Enter element : ");
                                scanf("%d",&x);
                                enqueue(x);
                                break;
                        case 2:
                                dequeue();
                                break;
                        case 3:
                                display();
                                break;
                        case 4:
                                isEmpty();
                                break;
                        case 5:
                                size();
                                break;
                        case 6: exit(0);
                }
        }
}
```

QueueOperations.c

```
#include <stdio.h>

#define MAX_SIZE 100

int front = -1;
int rear = -1;
int queue[MAX_SIZE];

void isEmpty() {
        if (front == -1 && rear == -1)
            printf("Queue is empty.\n");
        else
            printf("Queue is not empty.\n");

}

void size() {
        if (front == -1 && rear == -1)
           printf("Queue size : 0\n");
        else
           printf("Queue size : %d\n", rear - front + 1);
}

void enqueue(int element) {
        if (rear == MAX_SIZE - 1) {
            printf("Queue is full.\n");
        } else {
            if (front == -1) {
                front = 0;
            }
            rear++;
            queue[rear] = element;
             printf("Successfully inserted.\n");
    }
}

void dequeue() {
        if (front == -1) {
            printf("Queue is underflow.\n");
        } else {
            printf("Deleted element = %d\n", queue[front]);
            if (front == rear) {
                front = rear = -1;
            } else {
                 front++;
            }
        }
}

void display() {
        if (front == -1) {
            printf("Queue is empty.\n");
        } else {
            printf("Elements in the queue : ");
            for (int i = front; i <= rear; i++) {
```

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


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2
Queue is underflow.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
3
Queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
4
Queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
5
Queue size : 0
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
14
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
78
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
53
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
3
Elements in the queue : 14 78 53
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :

5
Queue size : 3
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
6

Test Case - 2
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
25
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2
Deleted element = 25
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2
Queue is underflow.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
3
Queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
65
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
3
Elements in the queue : 65
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
4
Queue is not empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2
Deleted element = 65
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
4
Queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit

```
Enter your option :
5
Queue size : 0
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
63
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
5
Queue size : 1
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
6
```

S.No: 11	Exp. Name: ***Implementation of Circular Queue using Arrays***	Date: 2023-10-31

Aim:

Write a C program to implement circular queue using **arrays**.

Note: Define the **MAX** value as 5.

Source Code:

CQueueUsingArray.c

```
#include <stdio.h>
#include <stdlib.h>
#include "CQueueOperations.c"

int main() {
        int op, x;
        while(1) {
                printf("1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit\n");
                printf("Enter your option : ");
                scanf("%d",&op);
                switch(op) {
                        case 1:
                                printf("Enter element : ");
                                scanf("%d",&x);
                                enqueue(x);
                                break;
                        case 2:
                                dequeue();
                                break;
                        case 3:
                                display();
                                break;
                        case 4:
                                isEmpty();
                                break;
                        case 5:
                                size();
                                break;
                        case 6: exit(0);
                }
        }
}
```

CQueueOperations.c

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

#define MAX 5

int circularQueue[MAX];
int front = -1, rear = -1;

void enqueue(int element) {
        if ((front == 0 && rear == MAX - 1) || (rear == front - 1 && front != -1)) {
            printf("Circular queue is overflow.\n");
        } else if (front == -1) {
            front = rear = 0;
            circularQueue[rear] = element;
            printf("Successfully inserted.\n");
    } else if (rear == MAX - 1 && front != 0) {
            rear = 0;
            circularQueue[rear] = element;
            printf("Successfully inserted.\n");
        } else {
            rear++;
            circularQueue[rear] = element;
            printf("Successfully inserted.\n");
        }
}

void dequeue() {
        if (front == -1) {
            printf("Circular queue is underflow.\n");
    } else if (front == rear) {
            printf("Deleted element = %d\n", circularQueue[front]);
            front = rear = -1;
        } else {
            printf("Deleted element = %d\n", circularQueue[front]);
            if (front == MAX - 1)
                front = 0;
            else
                front++;
        }
}

void display() {
        if (front == -1) {
            printf("Circular queue is empty.\n");
    } else {
            printf("Elements in the circular queue : ");
            if (rear >= front) {
                for (int i = front; i <= rear; i++) {
                    printf("%d ", circularQueue[i]);
                }
        } else {
               for (int i = front; i < MAX; i++) {
                   printf("%d ", circularQueue[i]);
               }
               for (int i = 0; i <= rear; i++) {
```

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

void isEmpty() {
        if (front == -1) {
            printf("Circular queue is empty.\n");
        } else {
            printf("Circular queue is not empty.\n");
        }
}

void size() {
        if (front == -1) {
            printf("Circular queue size : 0\n");
    } else if (front <= rear) {
            printf("Circular queue size : %d\n", rear - front + 1);
        } else {
            printf("Circular queue size : %d\n", MAX - front + rear + 1);
    }
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
2
Circular queue is underflow.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
3
Circular queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
4
Circular queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
12
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
34

```
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
56
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
3
Elements in the circular queue : 12 34 56
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
38
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
25
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
3
Elements in the circular queue : 12 34 56 38 25
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
56
Circular queue is overflow.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
2
Deleted element = 12
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
2
Deleted element = 34
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
3
Elements in the circular queue : 56 38 25
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
4
Circular queue is not empty.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
```

```
5
Circular queue size : 3
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
11
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
3
Elements in the circular queue : 56 38 25 11
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
6
```

Test Case - 2

User Output

```
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
5
Circular queue size : 0
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
34
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
55
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
26
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
77
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
```

```
Enter element :
38
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
59
Circular queue is overflow.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
3
Elements in the circular queue : 34 55 26 77 38
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
5
Circular queue size : 5
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
2
Deleted element = 34
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
2
Deleted element = 55
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
2
Deleted element = 26
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
3
Elements in the circular queue : 77 38
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
4
Circular queue is not empty.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
5
Circular queue size : 2
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
6
```

S.No: 12	Exp. Name: ***Write a C program to implement different Operations on Stack using Linked Lists***	Date: 2023-10-31

Aim:

Write a program to implement `stack` using **linked lists**.

Source Code:

StackUsingLL.c

```
#include <stdio.h>
#include <stdlib.h>
#include "StackOperationsLL.c"

int main() {
        int op, x;
        while(1) {
                printf("1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit\n");
                printf("Enter your option : ");
                scanf("%d", &op);
                switch(op) {
                        case 1:
                                printf("Enter element : ");
                                scanf("%d", &x);
                                push(x);
                                break;
                        case 2:
                                pop();
                                break;
                        case 3:
                                display();
                                break;
                        case 4:
                                isEmpty();
                                break;
                        case 5:
                                peek();
                                break;
                        case 6:
                                exit(0);
                }
        }
}
```

StackOperationsLL.c

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

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

struct Node* top = NULL;

void push(int element) {
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        if (newNode == NULL) {
            printf("Memory allocation failed. Stack is full.\n");
            return;
        }

        newNode->data = element;
        newNode->next = top;
        top = newNode;
        printf("Successfully pushed.\n");
}

void pop() {
        if (top == NULL) {
            printf("Stack is underflow.\n");
            return;
        }

        struct Node* temp = top;
        top = top->next;
        printf("Popped value = %d\n", temp->data);
        free(temp);
}

void display() {
    if (top == NULL) {
            printf("Stack is empty.\n");
            return;
    }

        printf("Elements of the stack are : ");
        struct Node* current = top;
        while (current != NULL) {
              printf("%d ", current->data);
              current = current->next;
        }
        printf("\n");
}

void isEmpty() {
        if (top == NULL) {
            printf("Stack is empty.\n");
        } else {
            printf("Stack is not empty.\n");
```

```
void peek() {
        if (top == NULL) {
           printf("Stack is underflow.\n");
           return;
        }

        printf("Peek value = %d\n", top->data);
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
33
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
22
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
55
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
66
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
3
Elements of the stack are : 66 55 22 33
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Popped value = 66
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :

2
Popped value = 55
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
3
Elements of the stack are : 22 33
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
5
Peek value = 22
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
4
Stack is not empty.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
6

Test Case - 2
User Output
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Stack is underflow.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
3
Stack is empty.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
5
Stack is underflow.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
4
Stack is empty.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
23
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
24
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit

Enter your option :
3
Elements of the stack are : 24 23
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
5
Peek value = 24
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Popped value = 24
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Popped value = 23
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Stack is underflow.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
4
Stack is empty.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
6

S.No: 13	Exp. Name: ***Write a C program to implement different Operations on Queue using Linked Lists***	**Date: 2023-11-01**

Aim:

Write a program to implement queue using **linked lists**.

```
Sample Input and Output:
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 1
    Enter element : 57
    Successfully inserted.
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 1
    Enter element : 87
    Successfully inserted.
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 5
    Queue size : 2
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 3
    Elements in the queue : 57 87
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 2
    Deleted value = 57
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 2
    Deleted value = 87
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 3
    Queue is empty.
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 5
    Queue size : 0
    1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
    Enter your option : 6
```

Source Code:

QueueUsingLL.c

```c
#include <conio.h>
#include <stdio.h>
#include "QueueOperationsLL.c"
int main() {
        int op, x;
        while(1) {
                printf("1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit\n");
                printf("Enter your option : ");
                scanf("%d",&op);
                switch(op) {
                        case 1:
                                printf("Enter element : ");
                                scanf("%d",&x);
                                enqueue(x);
                                break;
                        case 2:
                                dequeue();
                                break;
                        case 3:
                                display();
                                break;
                        case 4:
                                isEmpty();
                                break;
                        case 5:
                                size();
                                break;
                        case 6: exit(0);
                }
        }
}
```

QueueOperationsLL.c

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

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

struct Node* front = NULL;
struct Node* rear = NULL;

void enqueue(int element) {
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        if (newNode == NULL) {
            printf("Memory allocation failed. Queue is full.\n");
            return;
        }

        newNode->data = element;
        newNode->next = NULL;

        if (rear == NULL) {
            front = rear = newNode;
        } else {
            rear->next = newNode;
            rear = newNode;
        }
        printf("Successfully inserted.\n");
}

void dequeue() {
        if (front == NULL) {
            printf("Queue is underflow.\n");
            return;
        }

        struct Node* temp = front;
        front = front->next;

        if (front == NULL) {
            rear = NULL;
    }

        printf("Deleted value = %d\n", temp->data);
        free(temp);
}

void display() {
        if (front == NULL) {
            printf("Queue is empty.\n");
            return;
    }

        printf("Elements in the queue : ");
        struct Node* current = front;
```

```
            current = current->next;
        }
        printf("\n");
}

void isEmpty() {
        if (front == NULL) {
            printf("Queue is empty.\n");
        } else {
            printf("Queue is not empty.\n");
        }
}

void size() {
        int count = 0;
        struct Node* current = front;
        while (current != NULL) {
            count++;
            current = current->next;
        }

        printf("Queue size : %d\n", count);
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2
Queue is underflow.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
3
Queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
4
Queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
5
Queue size : 0
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
44
Successfully inserted.

1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
55
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
66
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
67
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
3
Elements in the queue : 44 55 66 67
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2
Deleted value = 44
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2
Deleted value = 55
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
5
Queue size : 2
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
4
Queue is not empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
6

Test Case - 2
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
23

```
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
234
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
45
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
456
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2
Deleted value = 23
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
3
Elements in the queue : 234 45 456
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2
Deleted value = 234
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
3
Elements in the queue : 45 456
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
4
Queue is not empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
5
Queue size : 2
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
6
```

S.No: 14	Exp. Name: ***Implementation of Circular Queue using Linked List***	Date: 2023-11-01

Aim:

Write a program to implement `circular queue` using **linked lists**.

```
Sample Input and Output:
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 1
    Enter element : 15
    Successfully inserted.
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 1
    Enter element : 16
    Successfully inserted.
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 1
    Enter element : 17
    Successfully inserted.
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 3
    Elements in the circular queue : 15 16 17
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 5
    Circular queue size : 3
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 2
    Deleted value = 15
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 2
    Deleted value = 16
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 2
    Deleted value = 17
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 3
    Circular queue is empty.
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 4
    Circular queue is empty.
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 5
    Circular queue size : 0
    1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
    Enter your option : 6
```

Source Code:

`CQueueLL.c`

```c
#include <stdlib.h>
#include <stdio.h>
#include "CQueueOperationsLL.c"
int main() {
        int op, x;
        while(1) {
                printf("1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit\n");
                printf("Enter your option : ");
                scanf("%d",&op);
                switch(op) {
                        case 1:
                                printf("Enter element : ");
                                scanf("%d",&x);
                                enqueue(x);
                                break;
                        case 2:
                                dequeue();
                                break;
                        case 3:
                                display();
                                break;
                        case 4:
                                isEmpty();
                                break;
                        case 5:
                                size();
                                break;
                        case 6: exit(0);
                }
        }
}
```

CQueueOperationsLL.c

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

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

struct Node* front = NULL;
struct Node* rear = NULL;

void enqueue(int element) {
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        if (newNode == NULL) {
            printf("Memory allocation failed. Circular queue is full.\n");
            return;
        }

    newNode->data = element;
        newNode->next = NULL;

        if (rear == NULL) {
            front = rear = newNode;
            rear->next = front;
        } else {
            rear->next = newNode;
            rear = newNode;
            rear->next = front;
        }

          printf("Successfully inserted.\n");
}

void dequeue() {
        if (front == NULL) {
            printf("Circular queue is underflow.\n");
            return;
        }

        struct Node* temp = front;
            if (front == rear) {
                front = rear = NULL;
        } else {
                front = front->next;
                rear->next = front;
        }

          printf("Deleted value = %d\n", temp->data);
          free(temp);
}

void display() {
        if (front == NULL) {
            printf("Circular queue is empty.\n");
            return;
```

```
        printf("Elements in the circular queue : ");
        struct Node* current = front;
        do {
            printf("%d ", current->data);
            current = current->next;
        } while (current != front);

        printf("\n");
}

void isEmpty() {
        if (front == NULL) {
            printf("Circular queue is empty.\n");
        } else {
            printf("Circular queue is not empty.\n");
        }
}

void size() {
        int count = 0;
        if (front == NULL) {
            printf("Circular queue size : 0\n");
        } else {
            struct Node* current = front;
            do {
                count++;
                current = current->next;
            } while (current != front);

           printf("Circular queue size : %d\n", count);
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
15
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
16
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :

1
Enter element :
17
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
3
Elements in the circular queue : 15 16 17
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
5
Circular queue size : 3
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
2
Deleted value = 15
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
2
Deleted value = 16
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
2
Deleted value = 17
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
3
Circular queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
4
Circular queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
5
Circular queue size : 0
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
6

Test Case - 2
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
2
Circular queue is underflow.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
5

```
Circular queue size : 0
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
4
Circular queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
3
Circular queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
143
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
153
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
163
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
173
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
3
Elements in the circular queue : 143 153 163 173
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
2
Deleted value = 143
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
2
Deleted value = 153
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
5
Circular queue size : 2
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
```

4
Circular queue is not empty.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
6