Experiment 4: (stack and queue)

Q1) Write a program using C to create a stack of numbers and perform using UDF:

(i) push operation (ii) pop operation (iii) display operation

```
#include <stdio.h>
#define MAX 50
int stack[MAX], top = -1;
void push(int element) {
   if (top == MAX - 1) {
      printf("Stack Overflow!\n");
   } else {
      stack[++top] = element;
      printf("%d pushed onto stack.\n", element);
   }
}
void pop() {
   if (top == -1) {
      printf("Stack Underflow!\n");
   } else {
      printf("%d popped from stack.\n", stack[top--]);
   }
}
void display() {
   if (top == -1) {
      printf("Stack is empty.\n");
   } else {
      printf("Stack elements: ");
      for (int i = 0; i <= top; i++) {
         printf("%d ", stack[i]);
      }
      printf("\n");
```

```
    }
}
int main() {
    int choice, element;
    do {
        printf("\n1. Push\n2. Pop\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
        case 1:
            printf("Enter element to push: ");
            scanf("%d", &element);
            push(element);
            break;
        case 2:
            pop();
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting program.\n");
            break;
        default:
            printf("Invalid choice!\n");
        }
    } while (choice != 4);

    return 0;
}
```

Q2) Write a C program to create a linear queue and perform the following operations using UDF:

(i) insertion ii) deletion and iii) Traversal

```
#include <stdio.h>
#define MAX 50
int queue[MAX], front = -1, rear = -1;
void insert(int element) {
   if (rear == MAX - 1) {
      printf("Queue Overflow!\n");
   } else {
      if (front == -1) {
         front = 0;
      }
      queue[++rear] = element;
      printf("%d inserted into queue.\n", element);
   }
}
void delete() {
   if (front == -1 || front > rear) {
      printf("Queue Underflow!\n");
   } else {
      printf("%d deleted from queue.\n", queue[front++]);
      if (front > rear) {
         front = rear = -1; // Reset queue if empty
      }
   }
}
void traverse() {
   if (front == -1 || front > rear) {
      printf("Queue is empty.\n");
   } else {
```

```
        printf("Queue elements: ");
        for (int i = front; i <= rear; i++) {
            printf("%d ", queue[i]);
        }
        printf("\n");
    }
}
int main() {
    int choice, element;
    do {
        printf("\n1. Insert\n2. Delete\n3. Traverse\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
        case 1:
            printf("Enter element to insert: ");
            scanf("%d", &element);
            insert(element);
            break;
        case 2:
            delete();
            break;
        case 3:
            traverse();
            break;
        case 4:
            printf("Exiting program.\n");
            break;
        default:
            printf("Invalid choice!\n");
        }
```

```
    } while (choice != 4);
    return 0;
}
```

Q3) Write a C program to create a circular queue and perform the following operations using UDF:

(i) insertion ii) deletion and iii) Traversal

```
#include <stdio.h>
#define MAX 50
int cqueue[MAX], front = -1, rear = -1;
void insert(int element) {
    if ((front == 0 && rear == MAX - 1) || (rear + 1 == front)) {
        printf("Circular Queue Overflow!\n");
    } else {
        if (front == -1) {
            front = rear = 0;
        } else if (rear == MAX - 1) {
            rear = 0;
        } else {
            rear++;
        }
        cqueue[rear] = element;
        printf("%d inserted into circular queue.\n", element);
    }
}
void delete() {
    if (front == -1) {
        printf("Circular Queue Underflow!\n");
    } else {
        printf("%d deleted from circular queue.\n", cqueue[front]);
        if (front == rear) {
```

```
            front = rear = -1;
        } else if (front == MAX - 1) {
            front = 0;
        } else {
            front++;
        }
    }
}
void traverse() {
    if (front == -1) {
        printf("Circular Queue is empty.\n");
    } else {
        printf("Circular Queue elements: ");
        int i = front;
        while (1) {
            printf("%d ", cqueue[i]);
            if (i == rear) {
                break;
            }
            i = (i + 1) % MAX;
        }
        printf("\n");
    }
}
int main() {
    int choice, element;
    do {
        printf("\n1. Insert\n2. Delete\n3. Traverse\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
```

```
        case 1:
            printf("Enter element to insert: ");
            scanf("%d", &element);
            insert(element);
            break;
        case 2:
            delete();
            break;
        case 3:
            traverse();
            break;
        case 4:
            printf("Exiting program.\n");
            break;
        default:
            printf("Invalid choice!\n");
        }
    } while (choice != 4);
    return 0;
}
```

Experiment 5: (searching and sorting)

Q1) Write a program to implement binary search on array elements using UDF

```
#include <stdio.h>


int binarySearch(int arr[], int size, int key) {
    int low = 0, high = size - 1, mid;


    while (low <= high) {
        mid = (low + high) / 2;
        if (arr[mid] == key) {
```

```
            return mid;
        } else if (arr[mid] < key) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return -1;
}

int main() {
    int arr[50], size, key, result;

    printf("Enter number of elements: ");
    scanf("%d", &size);
    printf("Enter elements in ascending order: ");
    for (int i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }

    printf("Enter the element to search: ");
    scanf("%d", &key);

    result = binarySearch(arr, size, key);

    if (result == -1) {
        printf("Element not found.\n");
    } else {
        printf("Element found at index %d (0-based index).\n", result);
```

```
    }

    return 0;
}
```

Q2) write a program to implement selection sort on a given list of array elements.

```
#include <stdio.h>

void selectionSort(int arr[], int size) {
    for (int i = 0; i < size - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < size; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        int temp = arr[minIndex];
        arr[minIndex] = arr[i];
        arr[i] = temp;
    }
}

int main() {
    int arr[50], size;

    printf("Enter number of elements: ");
    scanf("%d", &size);
    printf("Enter elements: ");
```

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

    selectionSort(arr, size);

    printf("Sorted array: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

Q3) Write a program to input a string and sort the alphabets in ascending order using bubble sort.

```
#include <stdio.h>
#include <string.h>
void bubbleSort(char str[]) {
    int len = strlen(str);
    for (int i = 0; i < len - 1; i++) {
        for (int j = 0; j < len - i - 1; j++) {
            if (str[j] > str[j + 1]) {
                char temp = str[j];
                str[j] = str[j + 1];
                str[j + 1] = temp;
            }
```

```
        }
    }
}

int main() {
    char str[50];

    printf("Enter a string: ");
    scanf("%s", str);

    bubbleSort(str);

    printf("Sorted string: %s\n", str);
    return 0;
}
```

Experiment 6: (sorting and merging)

Q1) Write a program to input elements into two arrays A[5] and B[5]. Input the elements in ascending order and then merge their values into a resultant array C[10] in sorted manner using UDF.

```
#include <stdio.h>

void mergeArrays(int a[], int b[], int c[], int sizeA, int sizeB) {
    int i = 0, j = 0, k = 0;

    while (i < sizeA && j < sizeB) {
        if (a[i] < b[j]) {
            c[k++] = a[i++];
        } else {
            c[k++] = b[j++];
```

```
        }
    }
    while (i < sizeA) {
        c[k++] = a[i++];
    }
    while (j < sizeB) {
        c[k++] = b[j++];
    }
}

int main() {
    int a[5], b[5], c[10];

    printf("Enter 5 elements for array A in ascending order: ");
    for (int i = 0; i < 5; i++) {
        scanf("%d", &a[i]);
    }

    printf("Enter 5 elements for array B in ascending order: ");
    for (int i = 0; i < 5; i++) {
        scanf("%d", &b[i]);
    }

    mergeArrays(a, b, c, 5, 5);

    printf("Merged and sorted array: ");
    for (int i = 0; i < 10; i++) {
        printf("%d ", c[i]);
    }
```

```
    printf("\n");

    return 0;
}
```

Q2) Write a program to implement insertion sort on a given list of array elements.

```
#include <stdio.h>
void insertionSort(int arr[], int size) {
    for (int i = 1; i < size; i++) {
        int key = arr[i];
        int j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

int main() {
    int arr[50], size;

    printf("Enter number of elements: ");
    scanf("%d", &size);
    printf("Enter elements: ");
    for (int i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
```

```
    }

    insertionSort(arr, size);

    printf("Sorted array: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

Q3) Write a C program to implement quick sort to a given list of integers to sort in ascending order.

```
#include <stdio.h>

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pivot = arr[high];
        int i = low - 1;

        for (int j = low; j < high; j++) {
            if (arr[j] <= pivot) {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
```

```
        }
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;

        int partitionIndex = i + 1;
        quickSort(arr, low, partitionIndex - 1);
        quickSort(arr, partitionIndex + 1, high);
    }
}

int main() {
    int arr[50], size;

    printf("Enter number of elements: ");
    scanf("%d", &size);
    printf("Enter elements: ");
    for (int i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }

    quickSort(arr, 0, size - 1);

    printf("Sorted array: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
```

```
    return 0;
}
```