Name: SATYAJIT GHANA CSE SEC-C Reg No: 17ETCS002159

Laboratory 3

Title of the Laboratory Exercise: Queues

Introduction and Purpose of Experiment

Queues are very important data structures used in many real time applications. This experiment introduces the development Queue ADT.

1. Aim and Objectives

   Aim

   - To develop Queue ADT and to use them for string applications

   Objectives

   At the end of this lab, the student will be able to

   - Design and develop and use queue and demonstrate its operations

2. Pseudo Codes

```
1. Queue - Enqueue
   If tail > MAX
         Print Queue Overflow
         End
   Else
         Set tail = tail + 1
         Set queue[tail] = value
   End

2. Queue - Dequeue
   If head  > tail or tail == -1
         Print Queue Underflow
         End
   Else
         Set head = head + 1
         Return queue[head - 1]
   End

3. Queue - Display
   Set i = head
   While i < tail
         Print queue[i]
         Set i = i +1;
   End
```

3. Implementation in C

```c
void enqueue(Queue* myqueue, void* data) {

    if (myqueue -> tail >= myqueue -> MAX) {
        printf("\n*Queue Overflow Detected !*\n");
        return;
    }
    myqueue -> data = realloc(myqueue -> data, (myqueue -> tail + 2) * sizeof *(myqueue -> data));
    if (myqueue -> data != NULL) {
        if (myqueue -> head == -1) {
            myqueue -> head = 0;
        }
        (myqueue -> data)[++myqueue -> tail] = data;
    } else {
        printf("\n*cannot allocate memory !*\n");
        return;
    }
}

void dequeue(Queue* myqueue) {
    if (myqueue -> head > myqueue -> tail) {
        printf("\n*Queue Underflow detected !*\n");
        return;
    }
    myqueue -> head ++;
    printf("removed"); ds((char*)(myqueue->data)[myqueue -> head - 1]); /* Remove the data */
}

void display(Queue* myqueue) {
    if (myqueue -> head > myqueue -> tail || myqueue -> head == -1) {
        printf("\nQueue is Empty\n");
        return;
    }
    for (int i = myqueue -> head  ; i <= myqueue -> tail ; i++) {
        ds(*(char**)(myqueue->data + i)) /* Display the data */
    }
}
```

Figure 1 Queue

4. Presentation of Results

```
--- QUEUES USING DYNAMIC ALLOCATIONS ---
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your choice : 1
Enter your data : Satyajit Ghana

--- QUEUES USING DYNAMIC ALLOCATIONS ---
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your choice : 3
DEBUG--**(char**)(myqueue->data + i) : Satyajit Ghana*

--- QUEUES USING DYNAMIC ALLOCATIONS ---
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your choice : 2
removed
DEBUG--*(char*)(myqueue->data)[myqueue -> head - 1] : Satyajit Ghana*
```

Explanation:

The list of choices are presented to the user, which are enqueuer, dequeue, display and exit. The first option 1. Enqueue is selected, the user is then asked for the data to input, the input is taken from the terminal as a String, since we are making a program for String Queue, the data taken is sent to enqueuer method, which adds it to the tail of the queue, the basic principle being that the first element is the head and the last element is the tail. Then the next option 3. Display is selected to view the current queue, which calls the display function, the elements from head to tail are displayed, i.e. the elements that are added first are displayed first and the elements added in the end are displayed at the end. Then the option 2. Dequeue is selected which removed the element from the head, hence incrementing the head pointer or the head data. The removed element is then displayed on the screen.

5. Conclusions

The simplest two search techniques are known as Depth-First Search (DFS) and Breadth-First Search (BFS). These two searches are described by looking at how the search tree (representing all the possible paths from the start) will be traversed.

Breadth-First Search with a Queue, in breadth-first search we explore all the nearest possibilities by finding all possible successors and enqueue them to a queue.