Laboratory 6

Title of the Laboratory Exercise: Stacks and queues

Introduction and Purpose of Experiment

Stacks and queues are very important data structures used in many real time applications. This experiment introduces the development of stack and queue ADT and applying them together.

1. Aim and Objectives

   Aim

   - To develop stack and 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 stack and demonstrate its operations
   - Design and develop and use queue and demonstrate its operations

2. Experimental Procedure
   1. Analyse the problem statement
   2. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
   3. Implement the algorithm in C language
   4. Compile the C program
   5. Test the implemented program
   6. Document the Results
   7. Analyse and discuss the outcomes of your experiment

3. Calculations/Computations/Algorithms

Algorithms:

<u>Push:</u>

```
Step 1 : Start
Step 2 : Create a new node with the value to be inserted.
Step 3 : If the stack is empty, set the next of the new node to null.
Step 4 : If the stack is not empty, set the next of the new node to top.
Step 5 : Finally, increment the top to point to the new node.
Step 6 : Stop
```

Pop:

```
Step 1 : Start
Step 2 : If the stack is empty, terminate the method as it is stack underflow.
Step 3 : If the stack is not empty, increment the top to point to the next node.
Step 4 : Hence the element pointed to by the top earlier is now removed.
Step 5 : Stop
```

Enqueue:

```
Step 1 : Start
Step 2 : Create a new node with the value to be inserted.
Step 3 : If the queue is empty, then set both front and rear to point to newNode.
Step 4 : If the queue is not empty, then set next to the rear of the new node and the rear to point to the new node.
Step 5 : Stop
```

Dequeue:

```
Step 1 : Start
Step 2 : If the queue is empty, terminate the method.
Step 3 : If the queue is not empty, increment the front to point to the next node.
Step 4 : Finally, check if the front is null, then set rear to null also. This signifies an empty queue.
Step 5 : Stop
```

Implementation in C:

```c
#include <stdio.h>
#include "vector.h"
#include "input_helper.h"
#include "linked_list.h"

enum {
    STACK=1,
    QUEUE=2
```

```
};

void stack_menu();
void main_menu();
void queue_menu();

int main() {
    LinkedList* head = NULL;
    int choice;
    char* input;
    main_menu();
    choice = next_int();
    switch (choice) {
        case STACK: {
            while (1) {
                stack_menu();
                choice = next_int();
                switch (choice) {
                    case 1: {
                        printf("\nEnter your Data : ");
                        get_line(&input);
                        add_at_end_of_list(&head, newLink(input));
                    }
                        break;
                    case 2: {
                        LinkedList* deleted = remove_from_end_of_list(&head);
                        if (deleted != NULL) {
                            printf("\nDeleted : ");
                            print_link(deleted);
                        }
                    }
                        break;
                    case 3: {
                        print_list(head);
                    }
                        break;
                    case 4:
                        return 0;
                    default:
                        printf("\nInvalid Choice !\n");
                }
            }
        }
        case QUEUE: {
            while (1) {
                queue_menu();
                choice = next_int();
                switch (choice) {
                    case 1: {
                        printf("\nEnter your Data : ");
                        get_line(&input);
                        add_at_end_of_list(&head, newLink(input));
                    }
                        break;
                    case 2: {
                        LinkedList* deleted = remove_from_beg_of_list(&head);
                        if (deleted != NULL) {
                            printf("\nDeleted : ");
                            print_link(deleted);
```

```c
                        }
                    }
                        break;
                    case 3: {
                        print_list(head);
                    }
                        break;
                    case 4:
                        return 0;
                    default:
                        printf("\nInvalid Choice !\n");
                }
            }
        }
        default:
            printf("\nWrong Choice !\n");
            break;
    }
}

void stack_menu() {
    printf("\n---------- Stacks using Linked Lists ----------------------\n"
           "1.\tPush\n"
           "2.\tPop\n"
           "3.\tDisplay\n"
           "4.\tExit\n"
           "\nYour Choice : ");
}

void queue_menu() {
    printf("\n---------- Queues using Linked Lists ----------------------\n"
           "1.\tEn-queue\n"
           "2.\tDe-queue\n"
           "3.\tDisplay\n"
           "4.\tExit\n"
           "\nYour Choice : ");
}

void main_menu() {
    printf("\n-----------STACK AND QUEUES USING LINKED LISTS------\n"
           "1.\tStacks\n"
           "2.\tQueues\n"
           "\nYour Choice : ");
}
```

4. Presentation of Results

```
-----------STACK AND QUEUES USING LINKED LISTS------
1.     Stacks
2.     Queues

Your Choice : 1
1
```

```
---------- Stacks using Linked Lists ----------------------
1.    Push
2.    Pop
3.    Display
4.    Exit

Your Choice : 1
10
1
10

Enter your Data :
---------- Stacks using Linked Lists ----------------------
1.    Push
2.    Pop
3.    Display
4.    Exit

Your Choice : 1
1

Enter your Data : 123
123

---------- Stacks using Linked Lists ----------------------
1.    Push
2.    Pop
3.    Display
4.    Exit

Your Choice : 3
3

DEBUG--*(char*)(head -> data) : 10*

DEBUG--*(char*)(head -> data) : 123*

---------- Stacks using Linked Lists ----------------------
1.    Push
2.    Pop
3.    Display
4.    Exit

Your Choice : 2
2

Deleted :
DEBUG--*(char*)(link -> data) : 123*

---------- Stacks using Linked Lists ----------------------
1.    Push
2.    Pop
3.    Display
4.    Exit

Your Choice : 3
```

```
3

DEBUG--*(char*)(head -> data) : 10*

---------- Stacks using Linked Lists ----------------------
1.    Push
2.    Pop
3.    Display
4.    Exit

Your Choice :
```

5. Analysis and Discussions

A stack is a linear data structure which allows the adding and removing of elements in a particular order. New elements are added at the top of the stack. If we want to remove an element from the stack, we can only remove the top element from the stack. Since it allows insertion and deletion from only one end and the element to be inserted last will be the element to be deleted first, it is called the 'Last in First Out' data structure (LIFO). Stack can be implemented using both arrays and linked lists. The limitation, in the case of an array, is that we need to define the size at the beginning of the implementation. This makes our stack static. It can also result in “stack overflow” if we try to add elements after the array is full. So, to alleviate this problem, we use a linked list to implement the stack so that it can grow in real time. The time complexity of Push and Pop are O(1).

A queue is also a linear data structure where insertions and deletions are performed from two different ends. A new element is added from the rear of the queue and the deletion of existing elements occurs from the front. Since we can access elements from both ends and the element inserted first will be the one to be deleted first, the queue is called the 'First in First Out' data structure (FIFO). Similar to stack, the queue can also be implemented using both arrays and linked lists. But it also has the same drawback of limited size. Hence, we will be using a linked list to implement the queue.

6. Conclusions

Stack can be used to implement back/forward button in the browser. Undo feature in the text editors are also implemented using Stack. It is also used to implement recursion. Call and return mechanism for a method uses Stack. It is also used to implement backtracking.

CPU scheduling in Operating system uses Queue. The processes ready to execute and the requests of CPU resources wait in a queue and the request is served on first come first serve basis.

Data buffer - a physical memory storage which is used to temporarily store data while it is being moved from one place to another is also implemented using Queue.