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

Laboratory 2

Title of the Laboratory Exercise: Stacks

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 ADT.

1. Aim and Objectives

   Aim

   - To develop stack 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

2. Pseudo Code

```
1.  Stack - Push
    If top > MAX
            Print Stack Overflow
            End
    Else
            Set top = top + 1
            Set stack[top] = value
    End

2.  Stack - Pop
    If top  < 0
            Print Stack Underflow
            End
    Else
            Set top = top - 1
            Return stack[top + 1]
    End

3.  Stack - Display
    Set i = 0
    While i < top
            Print stack[i]
            Set i = i +1;

    End
```

3. Implementation in C

```c
void push(Stack* mystack, void* data) {

    if (mystack -> top >= mystack -> MAX) {
        printf("\n*Stack Overflow Detected !*\n");
        return;
    }
    mystack -> data = realloc(mystack -> data, (mystack -> top + 2) * sizeof *(mystack -> data));
    if (mystack -> data != NULL) (mystack -> data)[(++mystack -> top)] = data;
    else printf("\n*cannot allocate memory !*\n");
}

void pop(Stack* mystack) {
    if (mystack -> top -1 < 0) {
        printf("\n*Stack Underflow detected !*\n");
        return;
    }
    mystack -> top--;
    /* Resize the data array as the elements are removed */
    mystack -> data = realloc(mystack -> data, (mystack -> top + 1) * sizeof *(mystack-> data));
    /* Print the removed data */
    printf("removed");
    ds((char*)(mystack->data)[mystack -> top+1]);
}

void display(Stack* mystack) {
    for (int i = mystack -> top  ; i >= 0 ; i--) ds(*(char**)(mystack->data + i))
}
```

Figure 1 Stack

Figure 2 Queue

4. Presentation of Results

```
--- STACKS USING DYNAMIC ALLOCATIONS ---
1.Push 2.Pop 3.Display 4.Exit
Enter your choice : 1
Enter your data : Satyajit Ghana

--- STACKS USING DYNAMIC ALLOCATIONS ---
1.Push 2.Pop 3.Display 4.Exit
Enter your choice : 3
DEBUG--**(char**)(mystack->data + i) : Satyajit Ghana*

--- STACKS USING DYNAMIC ALLOCATIONS ---
1.Push 2.Pop 3.Display 4.Exit
Enter your choice : 2
removed
DEBUG--*(char*)(mystack->data)[(mystack -> top)+1] : Satyajit Ghana*
```

<u>Explanation:</u>

The list of choices are presented to the user, which are push, pop, display and exist, the first three are the basic operations that can be performed on a stack. 1.Push is selected and then the data to be pushed is taken as terminal input, which is treated as a string, since the aim is to make a Stack ADT that can work with strings, then the push function is called, which checked if the stack is full or not, if not then the data is added to the stack, and the value of the top is incremented. Then the next option 3. Display is selected to display all the data stored in the stack, this calls the display function, which prints the data stored from 0 to top, if the value of top is less than 0 then the stack is empty and nothing is printed. Then the option 2. Pop is selected, this removes the data from the top of the stack, hence the value of top is decremented and the data that was deleted is printed on the screen. The whole logic to Pop being that the last element added is removed first, or Last-In-First-Out.

5. Conclusions

Backtracking is used in algorithms in which there are steps along some path (state) from some starting point to some goal. In all of these cases, there are choices to be made among a number of options. We need some way to remember these decision points in case we want/need to come back and try the alternative Again, stacks can be used as part of the solution. Recursion is another, typically more favored, solution, which is actually implemented by a stack.