Lab3 Report

作者: 罗文杰
专业: 计算机科学与技术
学号: 3210102456

1 Introduction

After learning queue and stack in class, you are wondering if there is something more flexible. Maybe a list supporting pop and push on both sides sounds great.

We should maintain such a list, and support 4 operations:

1. `+ s` : push s to the left side
2. `-` : pop a char from the left side and print it
3. `[ s` : push s to the right side
4. `]` : pop a char from the right side and print it

Here, the list is empty at first, and s can be letters either lowercase or uppercase. If the list pops when empty, just print a _ as the result.

2 Array vs Linked List

A data structure is a way to store and organize data in a computer, so that it can be used efficiently.

Abstract Data Types (ADTs) are purely theoretical entities used to simplify the description of abstract algorithms, to classify and evaluate data structures, and to formally describe the type system of programming languages.

Array and Linked List are two ADTs. An array is a combination of numbers. The numbers stored in the same array must satisfy two conditions: they must be of the same type and the numbers must be stored consecutively in memory.

A linked list is a non-contiguous, non-sequential storage structure on a physical storage unit, where the logical order of data elements is achieved by the linking order of the pointers in the linked table. It consists of a series of nodes (each element in a linked table is called a node), which can be generated dynamically at runtime. Each node consists of two parts: a data field that stores the data elements and a pointer field that stores the address of the next node.

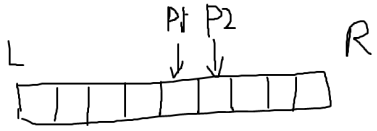
To build a stack, we can use either arrary or linked list, each of which has its advantages and disadvantages. Comparison between the two is as follws.

	Array	Linked list
Memory Boundaries	no borders	bounded
Memory distribution	continuous distribution	discontinuous distribution
Accessibility	random access available	visit in order
Capacity size	usually fixed	unsettled
Space occupied	small memory footprint	big memory footprint

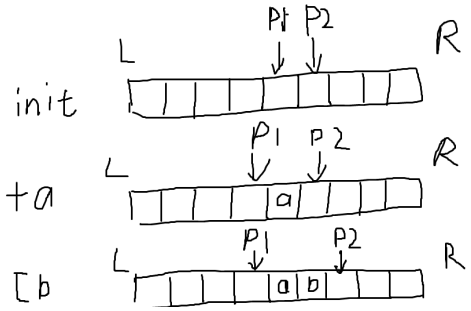
Given the ease of code implementation, we use arrays to complete the Flexible Stack architecture.

3 Logics of the Code

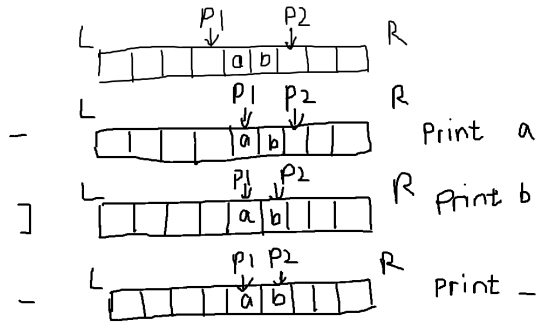
To make a stack can be PUSH or POP from two sides, we need two pointers pointing to the two top. (NOTE: L is low address and R is high address in my program)



When PUSH element into left side or right side, the corresponding pointer will store element's value into the memory, and move one space to left or right.



When POP element from left side or right side, The program will first check the difference between the addresses pointed to by the two pointers. If this difference is one, it means that the stack is empty and POP fails. If not, the corresponding pointer move one space and lord the element from memory.



4 Part of the Code

The code to implement PUSH and POP is shown below.

```
        ; R0 store the value needed to print
        ; R4 is the flag to decide wh left or right side the operated side
        ; R5 is the left pointer, R6 is the right pointer.
POP     NOT R1, R6
        ADD R1, R1, #2
        ADD R1, R5, R1
        BRz fail        ; Branch if stack is empty
        ADD R1, R4, #0
        BRz POLEFT      ; to left
        ADD R6, R6, #-1
        LDR R0, R6, #0  ; get the value
        BRnzp success
POLEFT  ADD R5, R5, #1
        LDR R0, R5, #0  ; get the value
        BRnzp success

PUSH    ADD R1, R4, #0
        BRz PULEFT      ; to left
        STR R0, R6, #0
        ADD R6, R6, #1
        BRnzp success
PULEFT  STR R0, R5, #0
        ADD R5, R5, #-1
success RET
fail    LD  R0, FLAG
        RET
FLAG    .FILL   x5F     ; _
```