Stacks and Queues

Prof. Michael Tsai

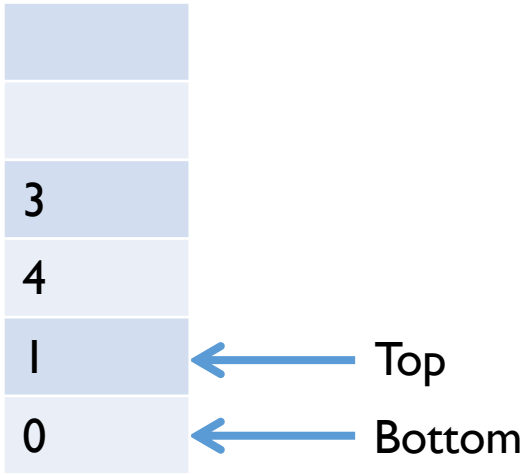
2017/02/21

Stack

- Stack represents an ordered and linear list

- "Stack of plates"

- Taken from and placed onto the top
- Pop: take an element
- Push: place an element
- Example →
- First in, ???? Out



A: Last

Therefore, Stack is a "First-In-Last-Out" (FILO) or "Last-In-First-Out" (LIFO) data structure.

Stack Operations

- So, what operations can be supported by a stack?
- What parameters do they need?

- Initialize a stack
- Push an element
- Pop an element
- Is it empty?
- Is it full?

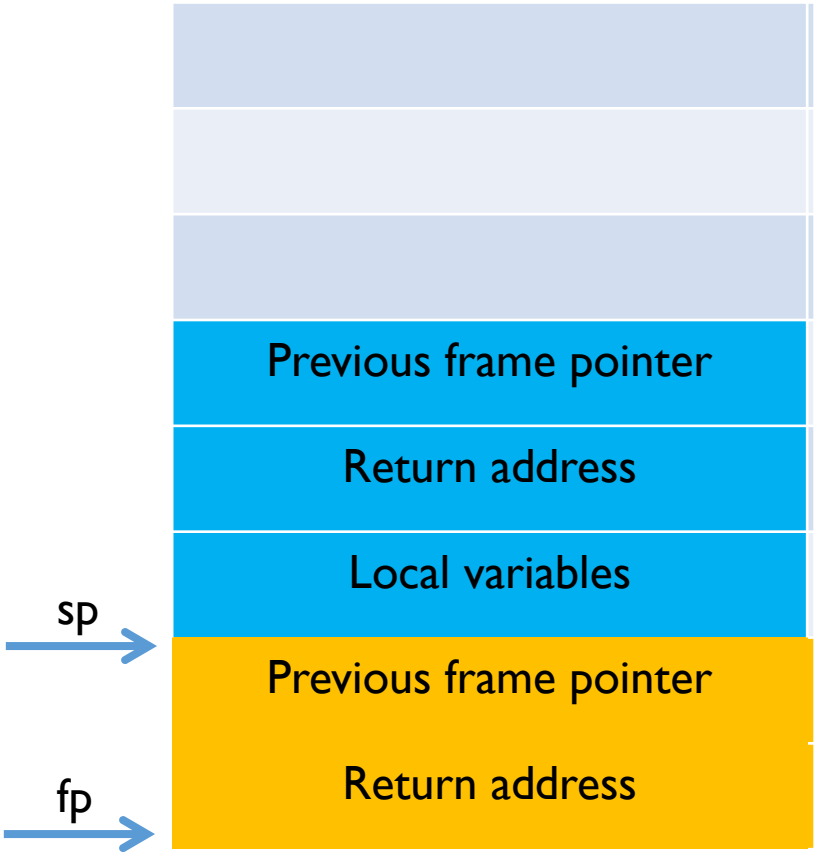
Stack Usage Example: System Stack

- A program uses a stack to store information about the calling function.

- What are stored?
- Activation record or stack frame
  - return address: where the calling function's next instruction is located.
  - previous frame pointer: the location of the last stack frame (in the stack)
  - local variables

- Example

- Recursive call?
- Stack overflow?



Stack Usage Example: Is it Palindrome?

- If a word looks the same forward and backward.
- Examples:
  - Noon, civic, racecar, madam
  - Was it a cat I saw

- Q: how do we use a stack to check if a string is a palindrome?

Stack Usage Example: Balanced parentheses

- Q: how do we use a stack to check if a string has balanced parentheses (each '(' is matched with a ')' in the string.)?

Think about it!

How to implement a stack?

- First version: using array
- What else do we need except the array to store the data?
  - A variable to record the index of the top element.
- How to implement these operations?
  - Push
  - Pop
  - Empty?
  - Full?
  - Number of current elements?

```
struct ArrayStack {
	int top;
	int capacity;
	int *array;
};


struct ArrayStack *CreateStack() {
	struct ArrayStack *S = malloc(sizeof(struct
ArrayStack));
	if (!S) return NULL;
	S->capacity=4;
	S->top=-1;
	S->array=(int*)malloc(S->capacity*sizeof(int))
	if (!S->array) return NULL;
	return S;
}
```

```
int IsEmptyStack(struct ArrayStack *S) {
   return (S->top==-1);
}


int IsFullStack(struct ArrayStack *S) {
   return (S->top==S->capacity-1);
}
```

```
void Push(struct ArrayStack*S, int data) {
      if (IsFullStack(S))
            printf("Stack Overflow");
      else
            S->array[++S->top]=data;
}

int Pop(struct ArrayStack*S) {
      if (IsEmptyStack(S)) {
            printf("Stack is Empty");
            return 0;
      } else
            return (S->array[S->top--]);

}
```

What if it is full?

- `array=(int*)malloc(S->capacity*sizeof(int));`

- If we have more than "capacity" elements, then it is full!
- How do we enlarge the stack?
- Hint: realloc() to the rescue!

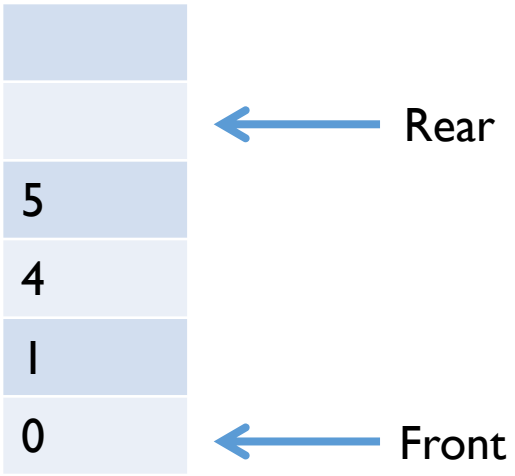
- realloc() can: (assume original size = n, new size = m)
  1. Allocate a new chunk of memory, sized m (larger)
  2. Move the content of the memory at the original location to the new location
  3. Give you the address of the new memory

- Q: How large should you realloc()?

Monitor Stack

Queue

- What is a queue?
- Queue is also an ordered and linear list
- But,
- Add from the **rear**, take from the Front
- Delete, or DeQueue: take an element
- Add, or EnQueue: add an element
- First-In-????-Out?



A: First Out

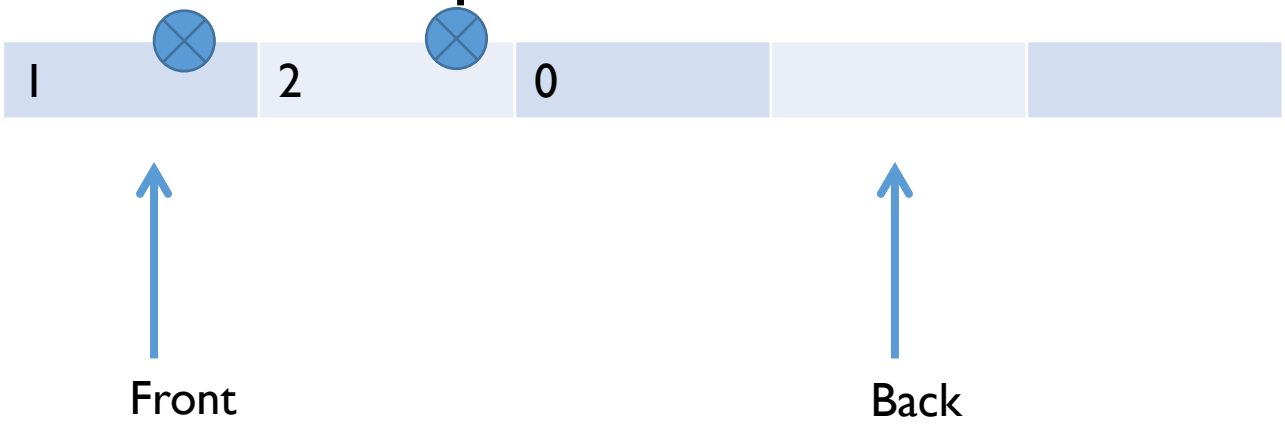
Queue

- So, what operations can be supported by a queue?
- What parameters do they need?

- Initialize a queue
- Add an element (EnQueue)
- Take an element (DeQueue)
- Is it empty?
- Is it full?

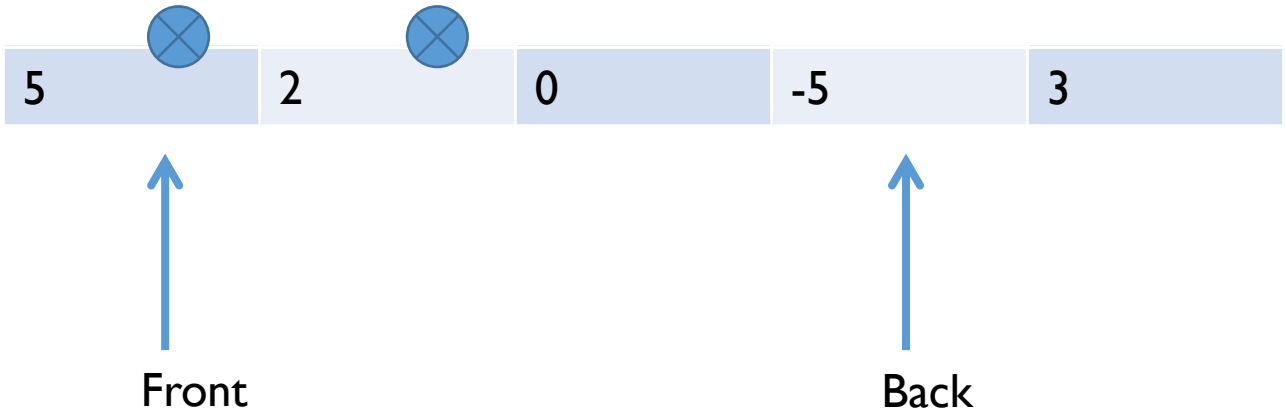
How do we implement a queue?

- Let's use array again
- Record front and back of the queue in the array (indices)
- But, here comes a problem:



- The queue is full! But space is wasted.
- How to solve this problem?

Solution: circular array



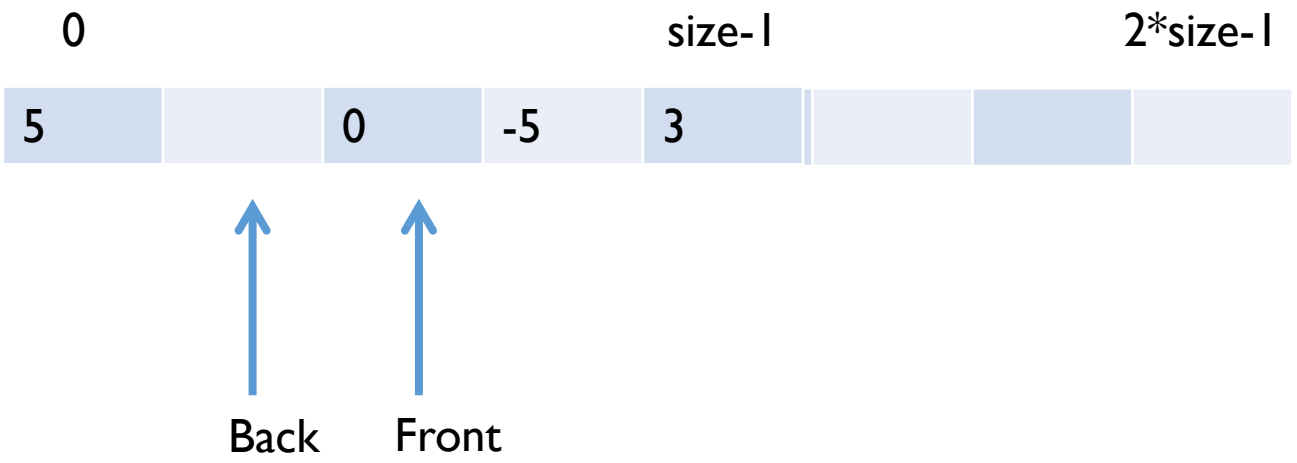
- The end of the array is assumed "connected" to the start of the array
- This makes sure that the number of elements which can be stored by a queue is the same as the size of the array

- When is the queue empty?
- When is the queue full?
- Can we tell them apart?!

What if it is actually full?

- We use realloc() to obtain a large chunk of memory.

- Then, other work to be done?
- It does not look right after realloc().



- How do we fix this?

What's Next?

- Before next lecture (week 3), read the assigned sections in the textbook.
  (there will be related in-class activities)

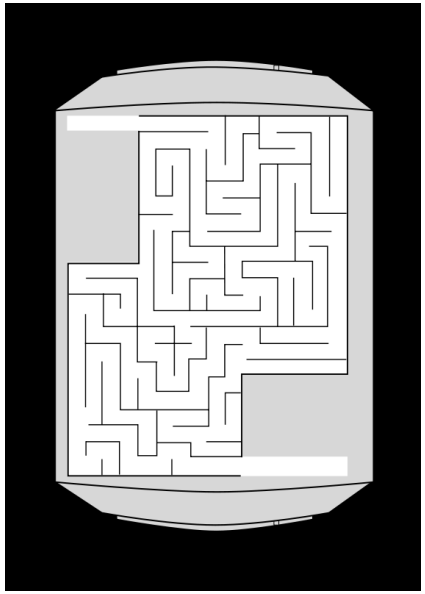
Applications of Stack & Queue (Reading assignment!)

接下來講一些應用:

- 計算機問題

- 迷宮問題

(回家自己看)



應用一: 計算機

- 不是普通的計算機
- 題目: 如果打入一串如1+2*3-5/(4+5)/5的算式, 請寫一個演算法來算出這串算式的結果.
- 怎麼寫呢? 好複雜T_T

先來看看算式長什麼樣子

- 1+2*3-5/(4+5)/5

- 裡面有:
- Operand – 1, 2, 3, 5, 4, 5, etc.
- Operator – + * - /
- 括號– (,)

- 特色1: 左到右(left-to-right associativity)
- 特色2:一般這種寫法叫做infix (operator夾在operand中間)
- 先後順序要operator去"比大小"
- 例如乘除是大, 加減是小, 那麼就先乘除後加減

別種寫法: postfix

- 把operator放到兩個operands的後面
- 例如 2+3*4 →2 3 4 * +
- a*b+5 → ?
- (1+2)*7 → ?
- a*b/c →?
- (a/(b-c+d))*(e-a)*c→?
- a/b-c+d*→?
- e-a*c→?

Postfix有什麼好處?

- 沒有括號
- 用stack幫忙就可以很容易地算出結果!

- 例子: 6 2 / 3 – 4 2 * +
- 從左邊讀過去
- 讀到6, 放6進stack (stack: 6)
- 讀到2, 放2進stack (stack: 6 2)
- 讀到/, 取兩個operands (6和2), 算6/2, 然後答案放回去stack (stack: 3)
- 讀到3, 放3進stack (stack: 3 3)
- 讀到-, 取兩個operands(3和3), 算3-3, 然後答案放回去stack (stack: 0)
- 依此類推….(請同學上來繼續完成☺)

剩下來的”小問題”: infix to postfix

- “小問題”: 怎麼把infix expression轉成postfix expression?

- 第一種方法: (適合紙上談兵)
- 1. 把整個expression
- 2. 把所有的operator都移到operand的後面去, 方便去除所有括號
- 3. 去除所有括號

- 但是實作上要怎麼做呢? (不希望做兩次)

又是一個可以利用stack解決的問題

- 方法:
- 1. 從左至右讀取expression
- 2. 碰到operand就直接輸出
- 3. 碰到operator時比較stack頂上的operator和目前讀到的operator哪一個比較"大" (precedence)
  - 如果目前讀到的operator**比較大**, 就把operator放到stack裡
  - 如果**一樣大**或者現在讀到的operator**比較小**, 就一直把stack裡面的operator拿出來印出來, 一直到stack是空的或者現在的operator比stack頂的operator大

- 為什麼可以這樣做?
- 裡面括號需要先印出, 但是卻是後讀到
- 比較外面的先用stack記起來, FILO, 由內而外

例子一

- 請一位同學來解說 怎麼把a+b*c利用前述方法轉換成postfix ☺

- 那麼, 如果碰到括號怎麼辦?

括號

- 括號內的有優先性

- 因此當碰到右括號的時候, 就立刻把stack中的東西一直拿出來直到碰到左括號為止
- 舉例: a*(b+c)
- 輸出的內容, stack內容(最右邊為top)
- a
- a, *
- a, *(
- ab, *(
- ab, *(+
- abc, *(+
- abc+*

還有一些小問題

- 小問題1: 左括號的"大小"到底是多少?
- 碰到左括號一定要放進去stack →此時要是最大的
- 左括號的下一個operator一定要可以放進去→此時要是最小的

- 結論:
- 左括號有兩種"大小"
- 在stack內的時候優先性為最小
- 在stack內的時候優先性為最大
- 其他operator的優先性則不管在stack內外都一樣

還有一些小問題

- Q: 如何確保stack空的時候, 第一次碰到的operator可放進去?
- A: 在stack底部放入一個虛擬operator帶有最低的優先性
- Q: 如何確保讀到expression最後的時候, 可以把stack裡面未取出的operator都拿出來?
- A: 字串最後可以加一個優先性最低的虛擬operator

例題

- a+(b*c/(d-f)+e)

- 用白板解說 (請同學? :P)

答案: abc*df-/e++

最後來一個有趣的迷宮問題吧

- 迷宮: 0是路, 1是牆壁. 每一部可以往上、下、左、右和四個斜角方向走一步.
- 問題: 怎麼找出一條路從(0,0)走到(7,7) ? (不一定要最短)
- 提示

	0	1	2	3	4	5	6	7
0	0	1	1	1	1	0	1	1
1	0	0	0	0	0	0	0	1
2	1	1	1	1	1	1	1	0
3	1	0	0	0	0	0	0	1
4	0	1	1	1	1	1	1	1
5	0	0	0	1	1	0	1	0
6	0	1	0	1	0	1	0	1
7	0	1	1	0	1	1	1	0

走迷宮的時候, 人要怎麼走?

- 最重要的時候, 是碰到岔路的時候
- 先記起來, 選其中一條走走看
- 如果碰壁了 (一直沒有走到終點), 就退回最後一次碰到的岔路, 換另外一條岔路
- 關鍵字: 最後一次碰到的岔路 (不是最先碰到的)
- 所以是先進後出→使用stack
- 關鍵字: 換另外一條岔路
- 要記得上一次走過哪一條路了

一些細節

- Q: 那麼, stack裡面要存什麼呢?

- A:
- “岔路”的地方的
- 座標, 也就是(row, col)
- 試過那些岔路了(試到八個方向的哪個方向了)

- Q: 要怎麼預防繞圈圈? (永遠出不來)
- A: 標示所有已經走過的地方, 走過就不用再走了
- <注意> 這是因為不用找”最短”的路

讓我們來寫algorithm

- 把(row_start, col_start, 第一個方向) 放入stack
- while(stack不是空的) {
  - 從stack拿出一組岔路點(row, col, dir)
    - while(還有別的dir還沒試) {
      - 將(row, col)往dir方向移動, 得到(row_n, col_n, dir).
      - 如果(row_n, col_n)就是終點, 則結束
      - 如果(row_n, col_n)不是牆壁且沒有來過 {
        - 標示(row_n, col_n)來過了
        - 把(row, col, dir的下一個方向)放入stack
        - row=row_n; col=col_n; dir=第一個方向;
      - }
    - }
- }