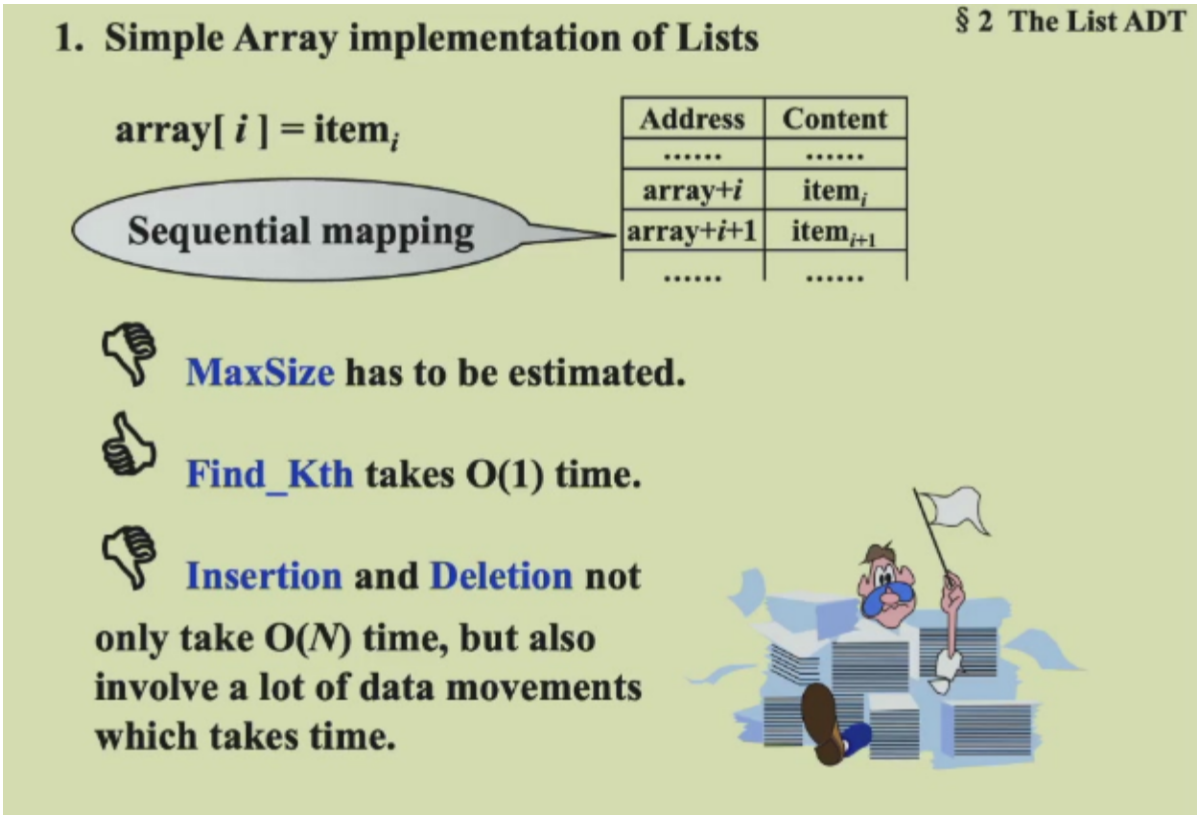
Lists, Stacks and Queues

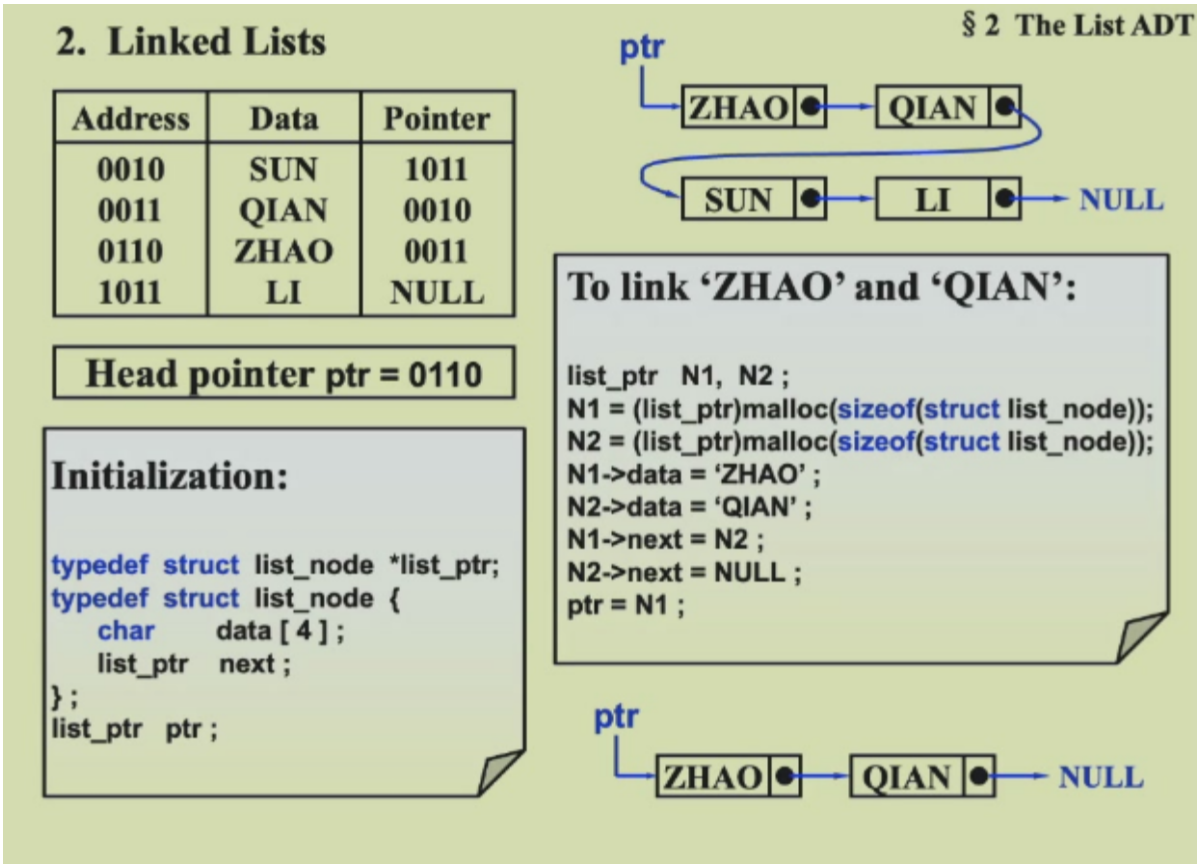
List ADT

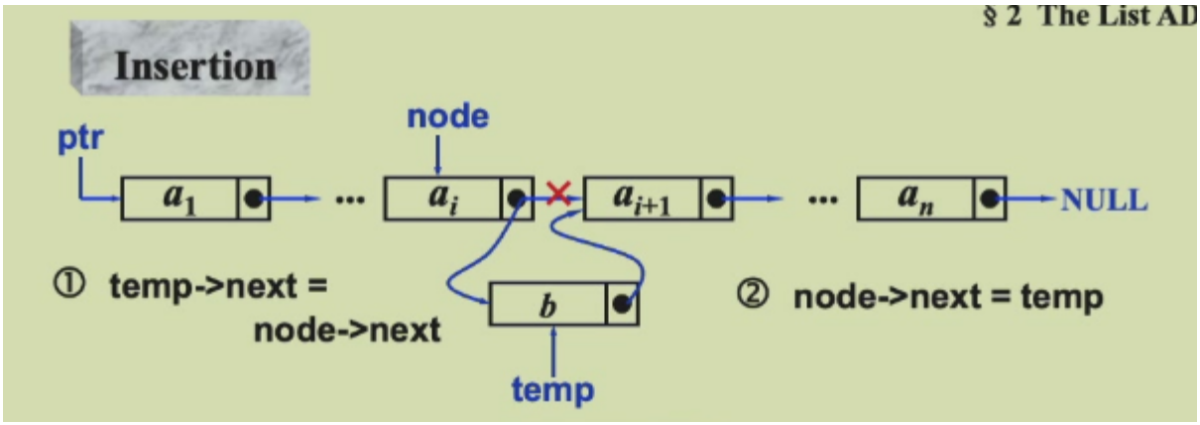
- Objects: item0, item1, ...
- Operations
  - length
  - print
  - making an empty list
  - finding kth item from the list
  - inserting a new item after kth item
  - deleting an item from the list
  - finding next of the current item
  - finding previous of the current item

Array Implementation

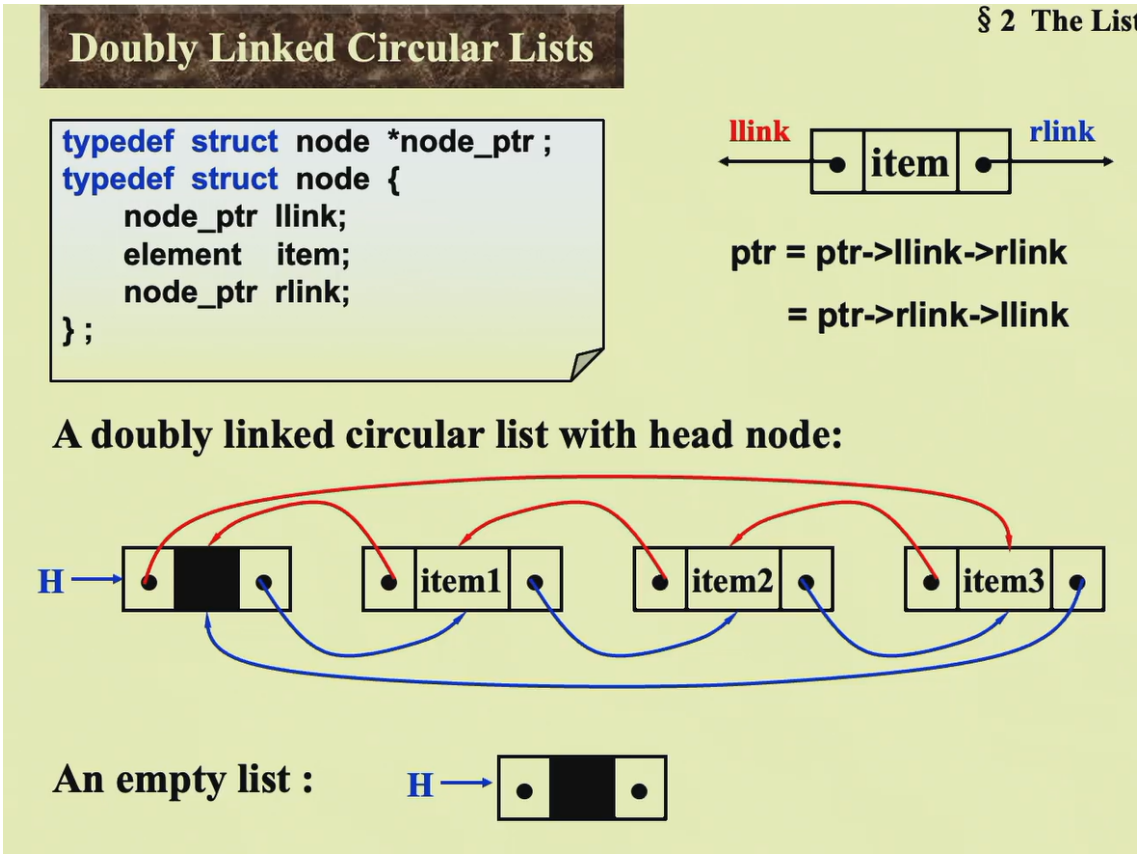


Linked lists





- 双向链表



set a dummy head

- 查找环：快慢指针

Applications

Two Applications

❋ **The Polynomial ADT**

Objects : $P(x) = a_1 x^{e1} + \cdots + a_n x^{en}$; a set of ordered pairs of $< e_i, a_i >$ where a_i is the coefficient and e_i is the exponent. e_i are nonnegative integers.

Operations:

- ☞ Finding degree, max $\{ e_i \}$, of a polynomial.
- ☞ Addition of two polynomials.
- ☞ Subtraction between two polynomials.
- ☞ Multiplication of two polynomials.
- ☞ Differentiation of a polynomial.

【Representation 1】

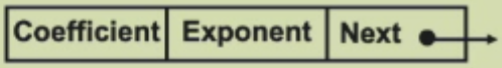
```
typedef struct {
    int    CoeffArray [ MaxDegree + 1 ] ;
    int    HighPower;
} *Polynomial ;
```

数组实现，计算简单，对稀疏数据不友好

【Representation 2】

Given: $A(x) = a_{m-1} x^{e_{m-1}} + \cdots + a_0 x^{e_0}$

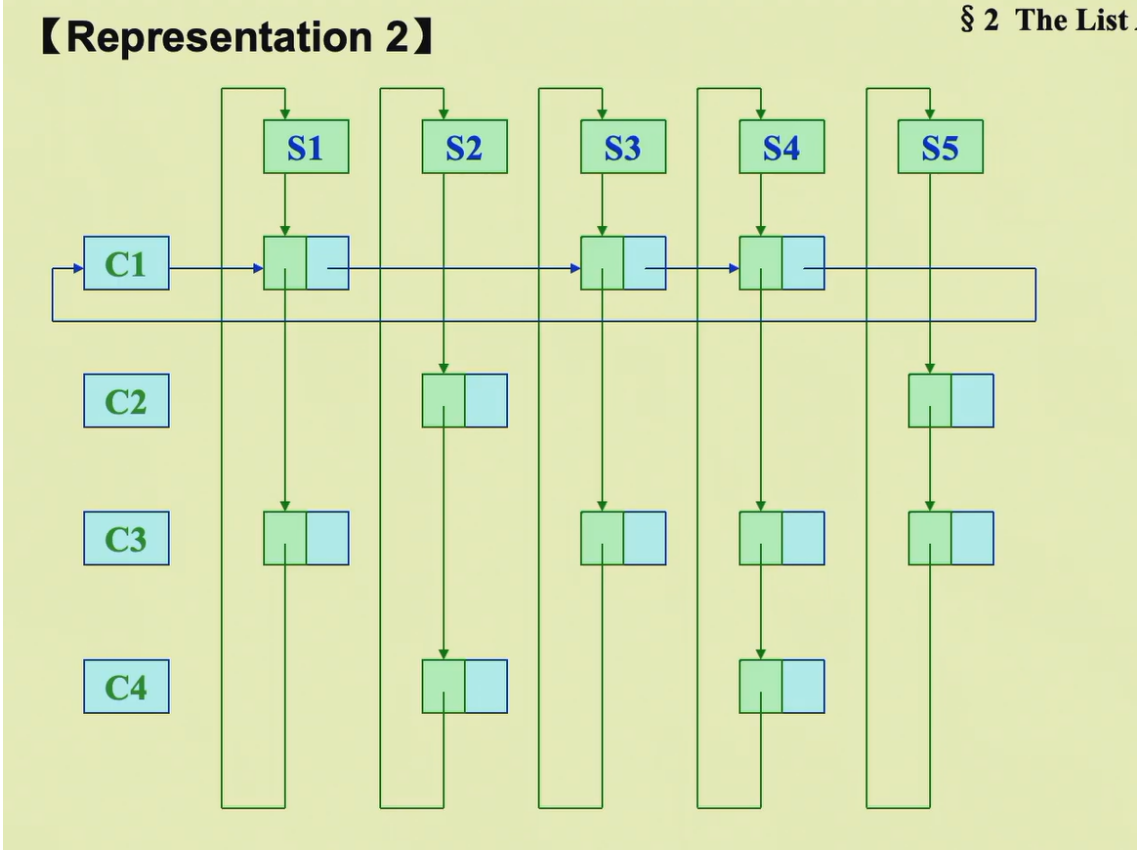
where $e_{m-1} > e_{m-2} > \cdots > e_0 \geq 0$ and $a_i \neq 0$ for $i = 0, 1, \cdots, m-1$.

We represent each term as a node 

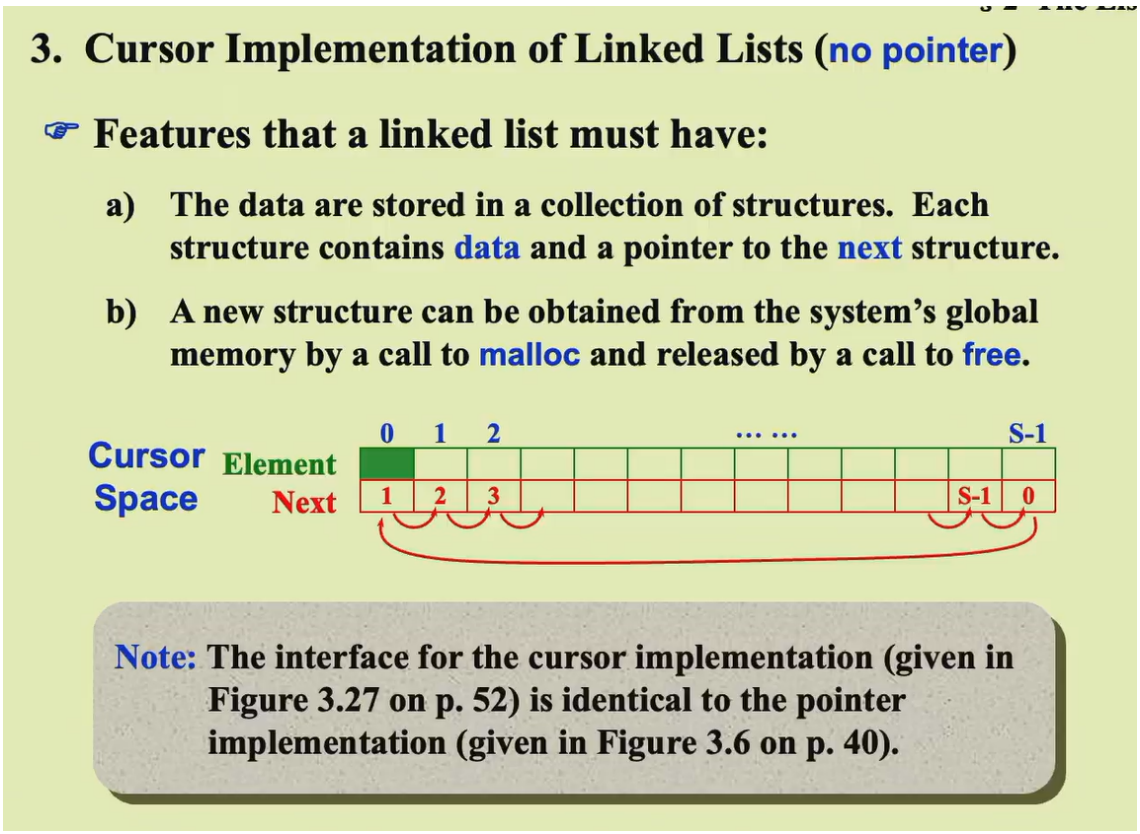
Declaration:

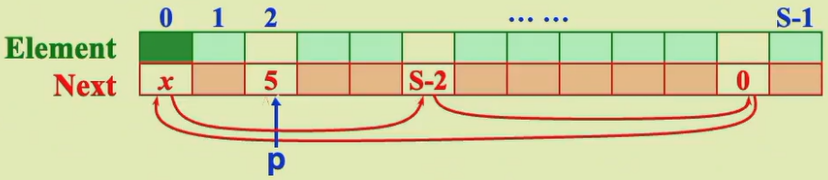
```
typedef  struct  poly_node   *poly_ptr;
struct  poly_node  {
    int      Coefficient ;  /* assume coefficients are integers */
    int      Exponent;
    poly_ptr   Next ;
};
typedef  poly_ptr   a ;    /* nodes sorted by exponent */
```

- multilist



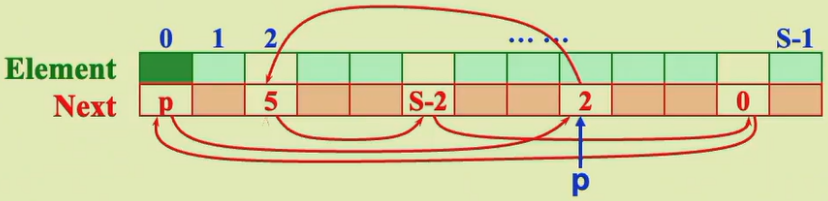
- Cursor Implementation of Linked Lists





malloc: p = CursorSpace[0].Next ;

CursorSpace[0].Next = CursorSpace[p].Next ;



free(p): CursorSpace[p].Next = CursorSpace[0].Next ;

CursorSpace[0].Next = p ;

Read operation implementations given in Figures 3.31-3.35

Note: The cursor implementation is usually significantly faster because of the lack of memory management routines.

```
position cursor_alloc( void )
{
  position p;
  p = CURSOR_SPACE[0].next;
  CURSOR_SPACE[0].next = CURSOR_SPACE[p].next;
  return p;
}

void cursor_free( position p)
{
  CURSOR_SPACE[p].next = CURSOR_SPACE[0].next;
  CURSOR_SPACE[0].next = p;
}
```

Figure 3.30 Routines: cursor-alloc and cursor-free

Slot	Element	Next
0	-	6
1	b	9
2	f	0
3	header	7
4	-	0
5	header	10
6	-	4
7	c	8
8	d	2
9	e	0
10	a	1

> 1. 整个数组模拟了一个global memory
> 2. 从0开始，形成的一个环就是freelist
> 3. alloc时，从这个环中删除一个元素
> 4. free时，将这个元素插入这个环

Stack ADT

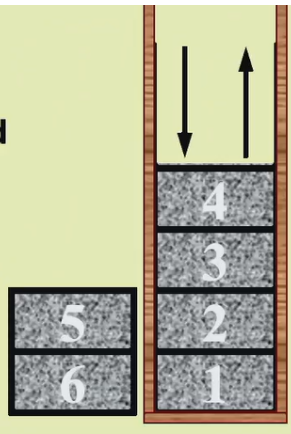
ADT

- Last in first out(LIFO)

1. ADT

A stack is a Last-In-First-Out (LIFO) list, that is, an ordered list in which insertions and deletions are made at the top only.

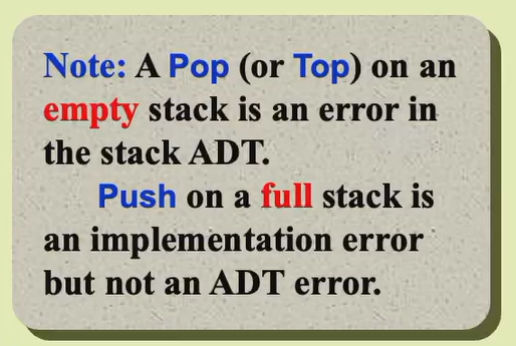
Objects: A finite ordered list with zero or more elements.



Operations:

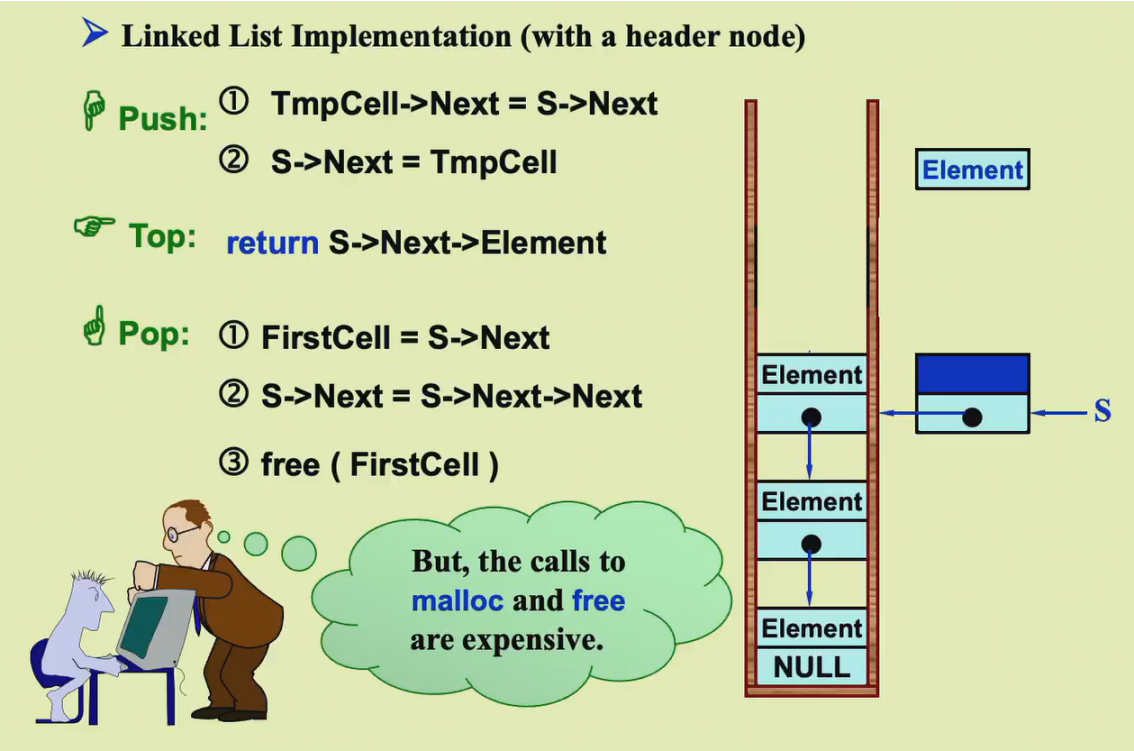
- Int IsEmpty(Stack S);
- Stack CreateStack( );
- DisposeStack(Stack S);
- MakeEmpty(Stack S);
- Push(ElementType X, Stack S);
- ElementType Top(Stack S);
- Pop(Stack S);

- operation only at top
- push: insert, pop: insert
- ADT error & implementation error



Implementation

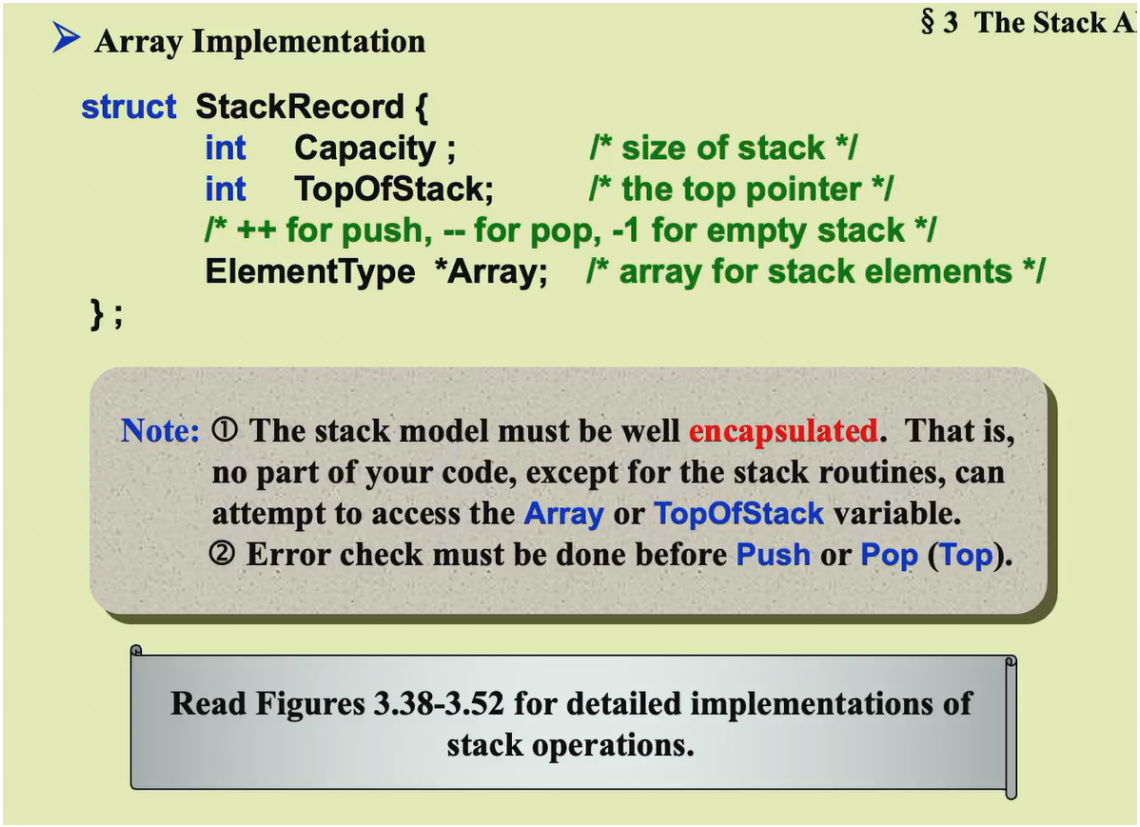
- Linked list implementation



  - Recycle bin

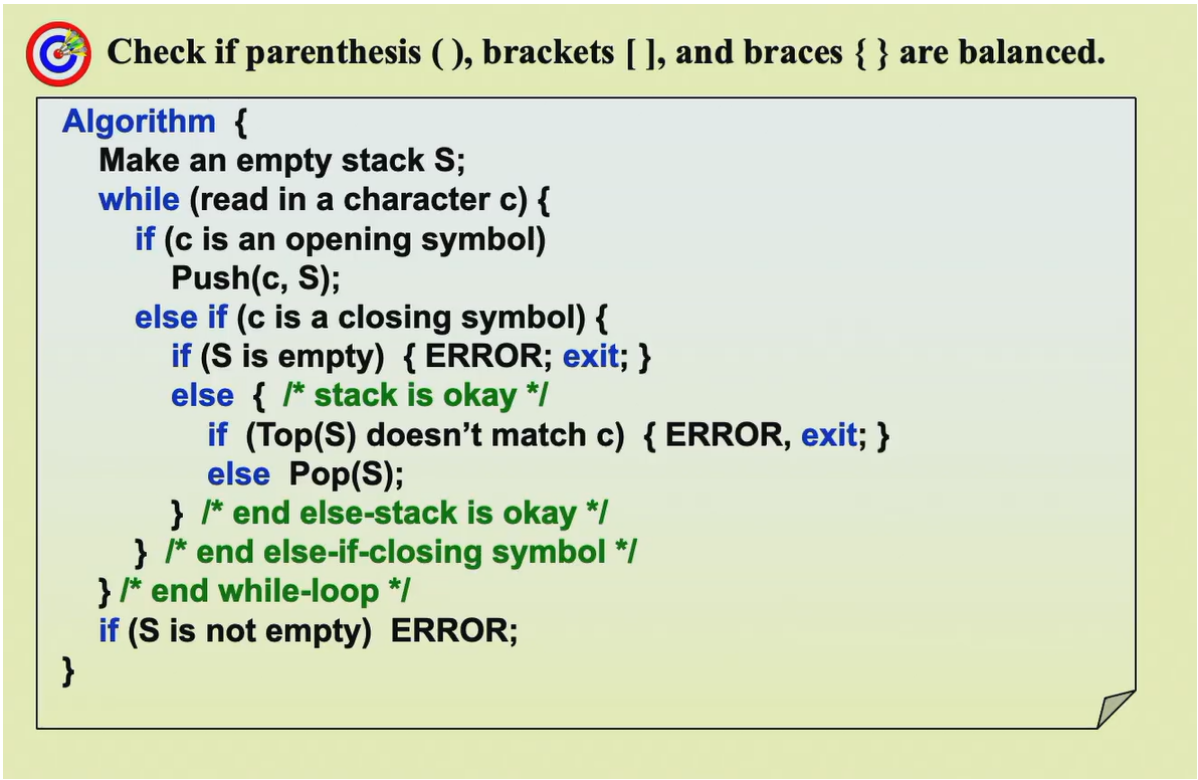
    维护另一个Stack: Recycle bin，在本栈pop后不执行free，push到rb里。要执行push时，从rb里pop出来一个。

- Array Implementation



Applications

Balancing Symbols

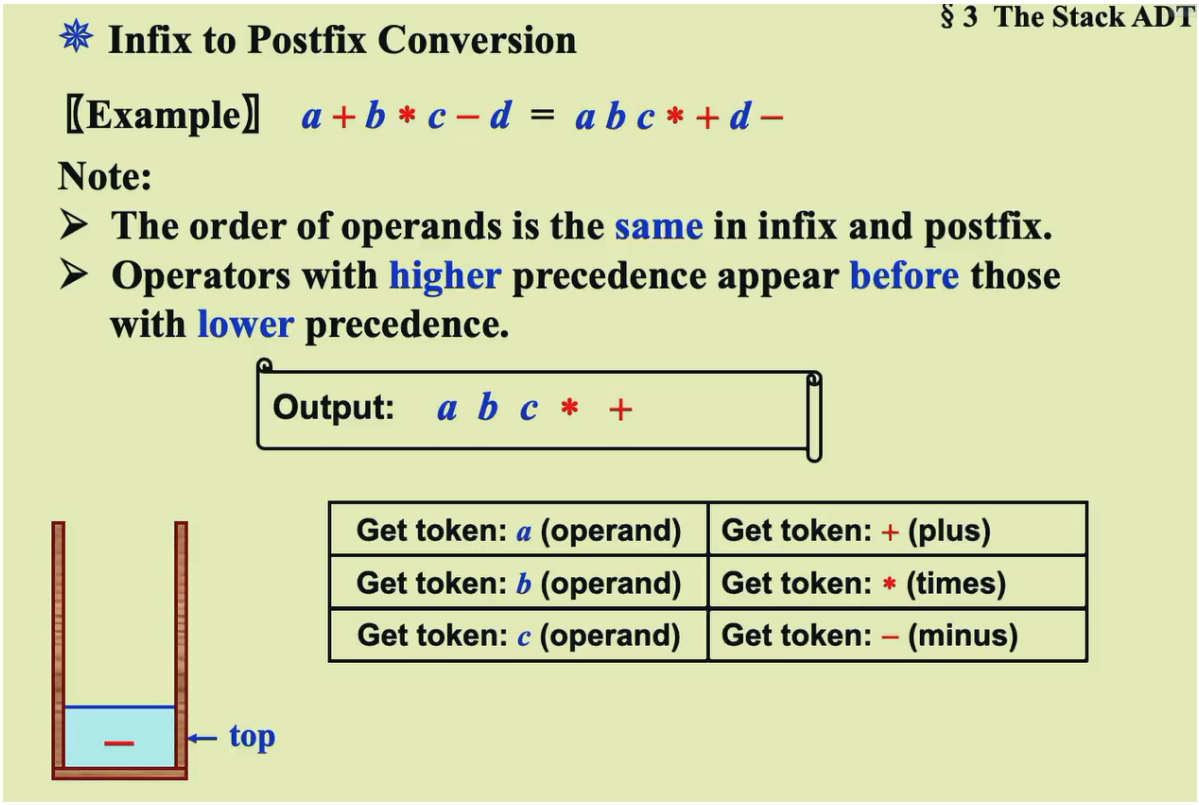


Postfix Evaluation

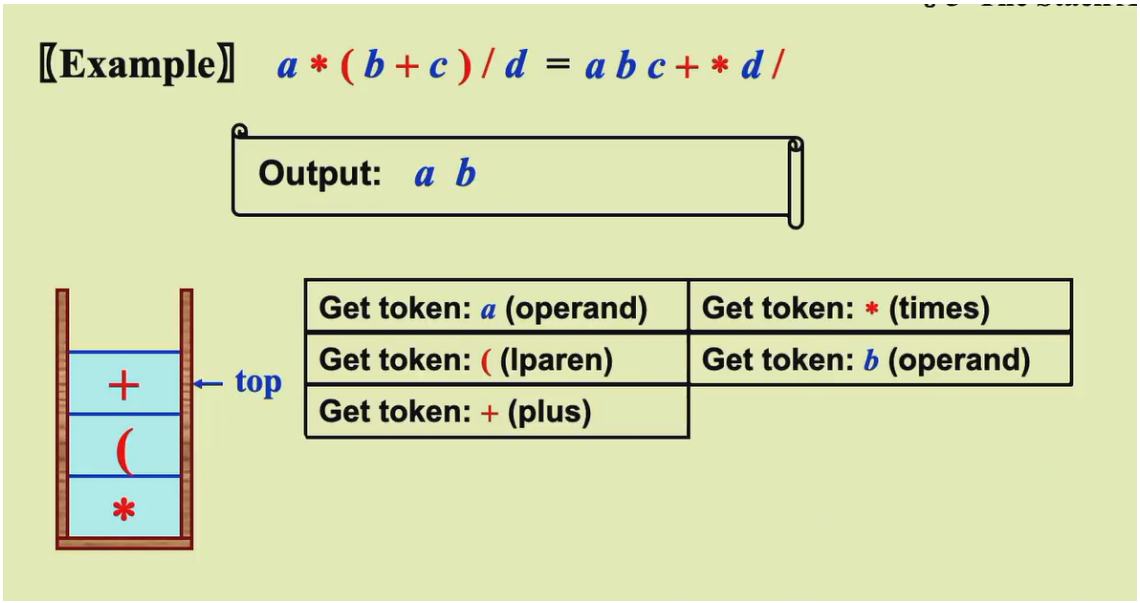
- 遇到operand入栈，遇到operator从栈中取数计算
- 计算后的结果继续push进去
- 结束后pop得到答案
- error: 出现pop空栈

Infix to Postfix Conversion

- 遇到operand输出，栈维护operator
- 若operator<=栈顶，将栈顶pop出来
- 若operator大于栈顶，operator入栈



- 有括号的情况，左括号不进stack优先级最高，左括号进stack后优先级最低
- 遇到右括号pop至左括号



Solutions:

① Never pop a (from the stack except when processing a) .

② Observe that when (is not in the stack, its precedence is the highest; but when it is in the stack, its precedence is the lowest. Define in-stack precedence and incoming precedence for symbols, and each time use the corresponding precedence for comparison.

Note: $a\ b - c -$ will be converted to $a\ b - c -$. However, 2^2^3 (2^{2^3}) must be converted to 2 2 3 ^ ^, not 2 2 ^ 3 ^ since exponentiation associates right to left.

Function Calls

Function Calls -- System Stack

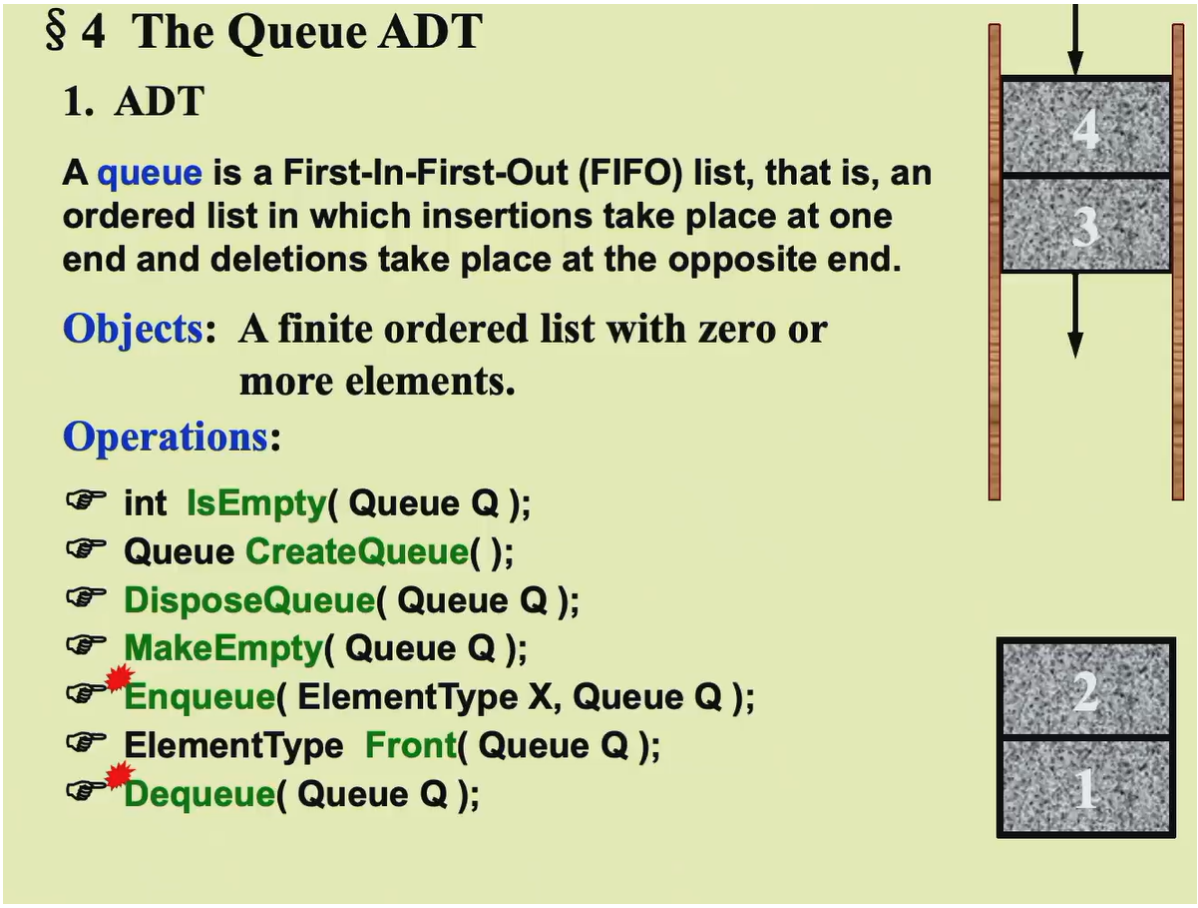

```
void  PrintList ( List L )
{
   if ( L != NULL ) {
      PrintElement ( L->Element );
      PrintList( L->next );
   }
} /* a bad use of recursion */
```

```
void  PrintList ( List L )
{
top:  if ( L != NULL ) {
         PrintElement ( L->Element );
         L = L->next;
         goto top; /* do NOT do this */
      }
} /* compiler removes recursion */
```

Queue ADT

ADT

First In First Out (FIFO)

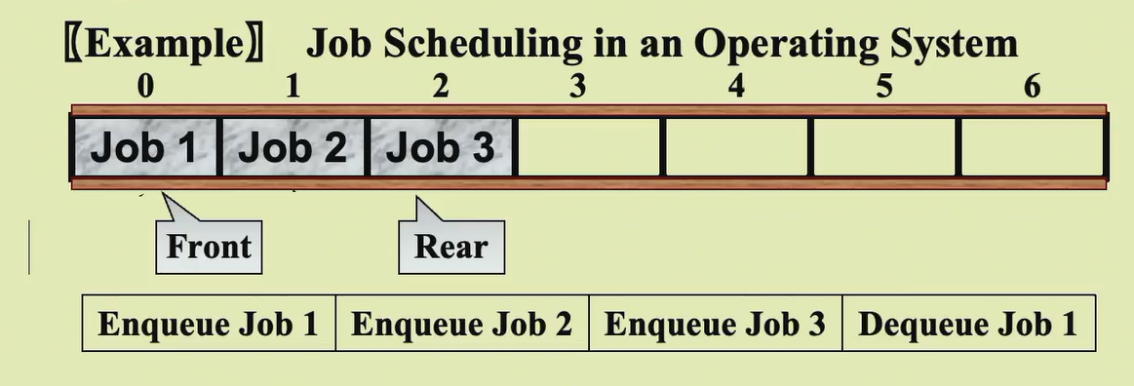


Implementation

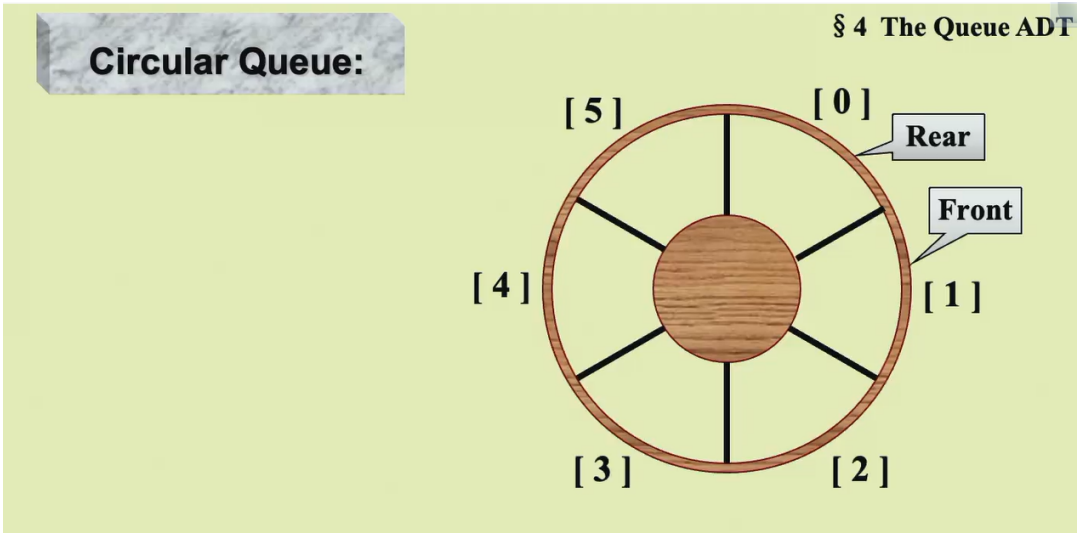
- Linked list

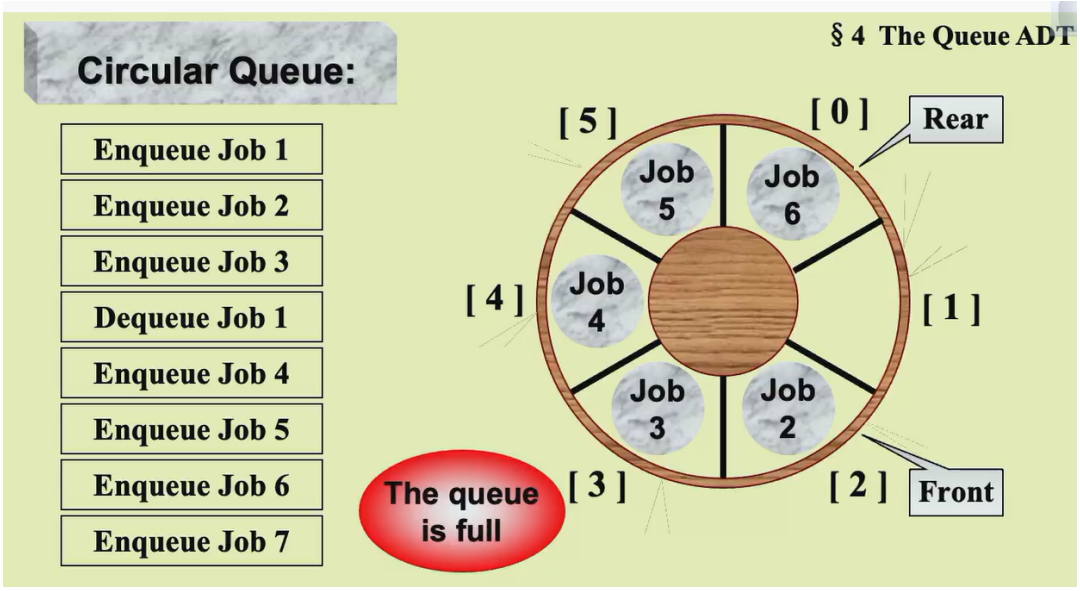
  本身就是队列

- Array



  - Enqueue: rear++
  - Dequeue: front++
  - 如何解决队尾问题：Circular Queue





否则无法区分满or空

- Method 1: 空一格
- Method 2: 加一个size字段