1.Algorithms and Data Structures

算法所需的操作与算法所需的数据关系很大，且速度与数据的利用效率有关。

数据结构是用来储存和处理数据的方法。

2.Data Structure：Stacks（栈）

栈是一种**后进先出**的（**Last-In, First-Out，LIFO**）数据结构。

对象可以随时插入栈中，且只能直接关联到最后一个插入的对象，原理可类比查看浏览器中的历史记录。

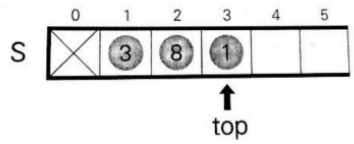
栈是一种**抽象数据类型（Abstract Data Type，ADT）**且支持以下方法：

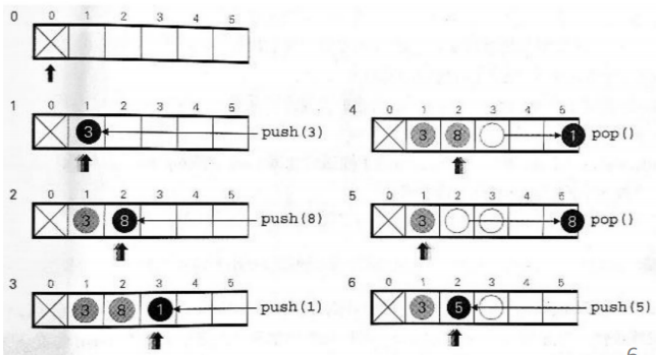
- push(Obj) : Insert object Obj onto the top of the stack. 将对象插入到栈的顶部。
- pop() : Remove (and return) the object from the top of the stack. An error occurs if the stack is empty. 把栈最顶部的对象从栈中移除并返回这个对象，若栈是空栈则报错。
- initialize() : initialize a stack. 将栈初始化。
- isEmpty() : returns a true if stack is empty, false otherwise. 栈为空则返回true，反之则返回false。
- isFull() : returns a true if stack is full, false otherwise. 同理

将对象压入（push）和弹出（pop）栈的方法：

```
PUSH(Obj)
1  ▷ Check to see if stack is full
2  if size() == N
3     then indicate stack-full error occurred
4  else t ← t + 1
5          S[t] ← Obj

POP()
1  ▷ Check to see if stack is empty
2  if isEmpty()
3     then indicate “stack empty” error occurred
4  else Obj ← S[t]
5          S[t] ← null
6          t ← t − 1
7  return Obj
```





栈的应用：

1.Important in run-time environments of modern procedural languages.

2.Evaluating arithmetic expressions can be performed using a stack if they are given using postfix notation (Reverse Polish notation (RPN), named for its developer Jan Łukasiewicz).

例如：将数组倒置

```
ReverseArray(Data: values[])
      // Push the values from the array onto the stack.
      Stack: stack = New Stack
      For i = 0 To <length of values> - 1
          stack.Push(values[i])
      Next i
      // Pop the items off the stack into the array.
      For i = 0 To <length of values> - 1
          values[i] = stack.Pop()
      Next i
End ReverseArray
```

另外，Reverse Polish notation：

在标准的算术表达式中，如 x+y xy 是operands +是addition operator ： infix notation 中缀表示法。

但是这种表示方法需要严格要求运算优先级

所以有：postfix notation 后缀表示法 操作的数优先于操作运算符 例如 x y +, x y z + * 或 x y + z *

► The postfix expression

x y w z / − *

gets translated into the expression

x * (y − w/z)

whereas

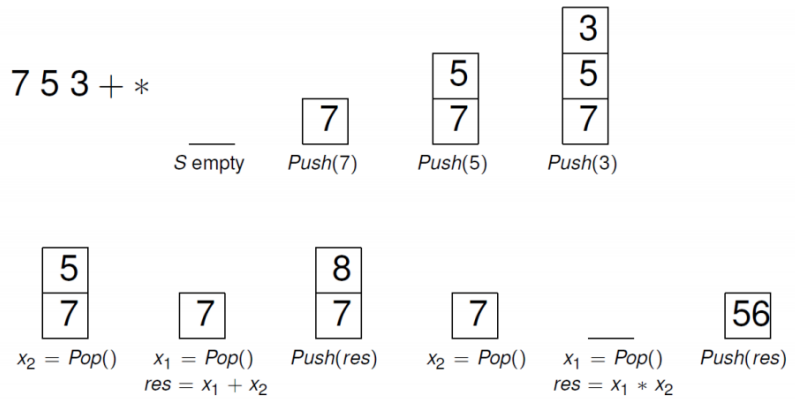
x y w /z − *

means

x * (y/w − z).

后缀表示法可以用栈来进行理解，例如 7 5 3 + *

首先将7,5,3依次推入栈中，如果遇到运算操作符就将栈最顶端两个数推出栈，对这两个数进行操作，然后将得到的结果压入栈中，继续上述操作。

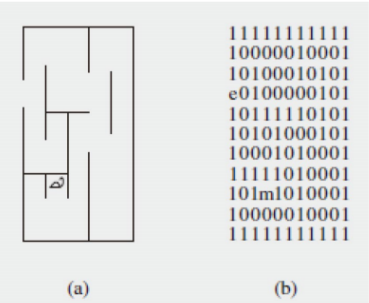


使用这种表示方式不需要用括号来表示运算优先级。

另外的例子：Exiting a maze

Problem : Exiting a Maze.

Consider the problem of a trapped mouse that tries to find its way to an exit in a maze.

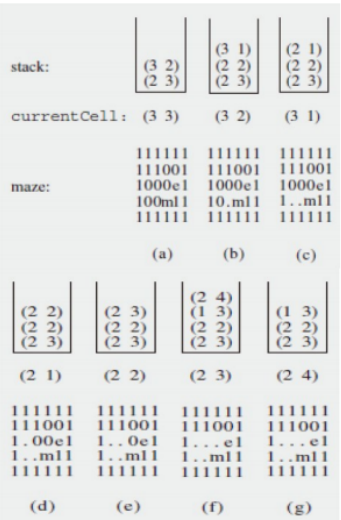


Write a program to solve the above problem by using stacks.

如上图所示，将迷宫简化为右边所示的图，迷宫壁用1代替，可走的道路用0代替，出口用e代替，老鼠用m代替，然后给迷宫的每行标注编号：

```
 012345
0111111
1111001
21000e1
3100m11
4111111
```

用一个有序数对表示对应的位置，如图所示m现在位于（3，3），然后找出与该点相邻的所有可以走的点的数对，并将这些数对压入栈，然后将最上方的选取为接下来要走的点，将该点弹出，然后压入与该点相邻的所有可以走的点（与上一步相同），同时将选取的点在迷宫中更新为．，然后继续上述步骤。



Write a program to solve the above problem by using stacks.

```
exitMaze()
initialize stack, exitCell, entryCell, currentCell = entryCell;
while currentCell is not exitCell
        mark currentCell as visited;
        push onto the stack the unvisited neighbors
of currentCell;
         If stack is empty
              failure;
          else pop off a cell from the stack and make it
currentCell;
success;
```

3.Data Structure：Queues（队列）

队列是一种先进先出的（**Fast-In, First-Out，FIFO**）数据结构，与现实生活中的队列类似。

对象可以在任何时候插入到队列的尾部，但只有最前面的对象可以被移除。

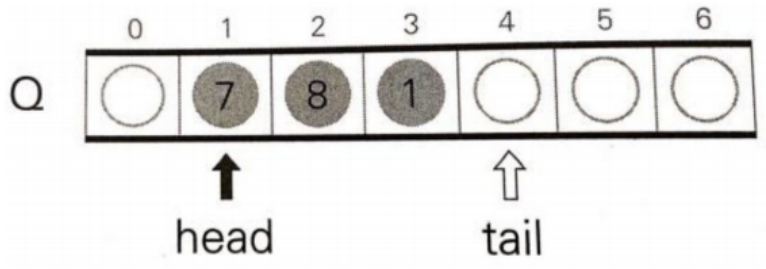
We say that elements enter the queue at the rear and are removed from the front.

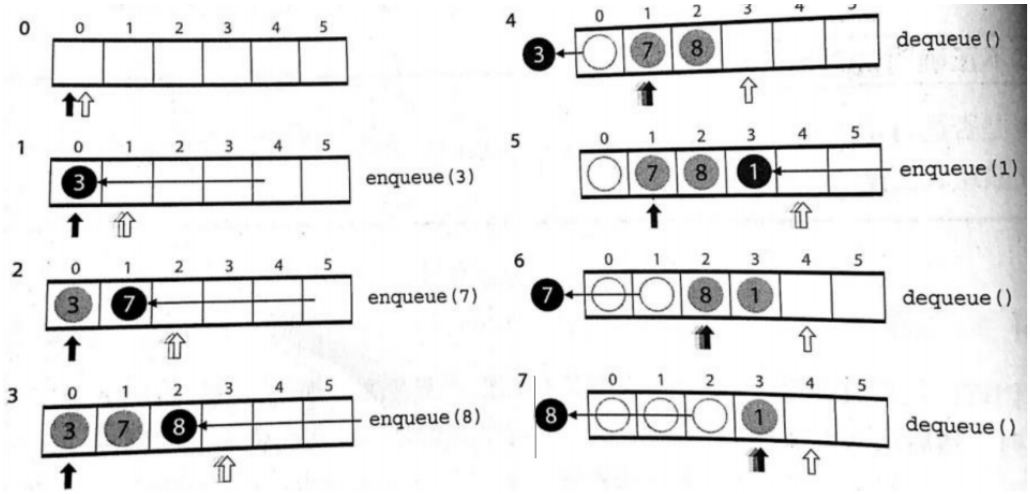
队列也是一种ADT，支持以下的方法：

- enqueue(Obj): inserts object Object the rear of the queue. 队列尾部插入对象。
- dequeue(): removes and returns the object from the front of the queue. An error occurs if the queue is empty. 移除并返回队列前方的对象，空队列将报错。

除去以上两个基础的方法，还有：

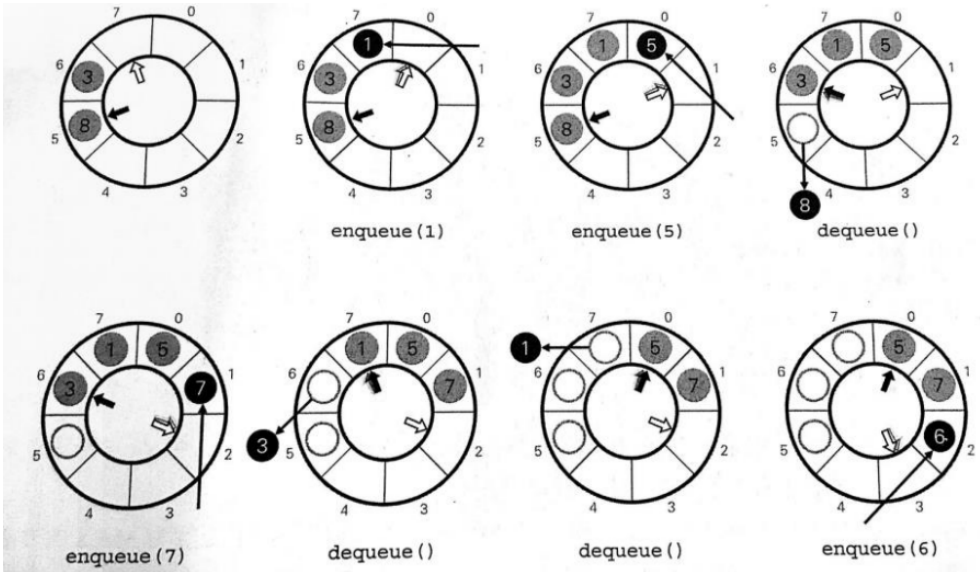
- size(): Return the number of objects in the queue. 返回队列中对象的数量。
- isEmpty(): Returns true if queue is empty, and false otherwise. 判断队列是否为空。
- isFull(): Returns true if queue is full, and false otherwise. 判断队列是否已满。
- front(): Return, but do not remove, the object at the front of the queue. An error is returned if the queue is empty. 返回但并不移除队列前方的对象，空队列将报错。





Moving to the end of the array
->out of memory

环形队列（Circular Queue）：



多程序设计实现了有限形式的并行且允许多个任务和线程，完成这一目的不允许有一个单独的线程独自占用CPU：One solution is to use a queue to allocate CPU time to threads in a round robin protocol.

4.Data Structure：List（列表）

列表是一组对象的集合，每个对象都被储存在一个节点（node）中，包含有一个数据区域和一个指向列表中下一个元素的指针（pointer）。

数据可以插入列表中的任何位置（插入一个新节点并且重新指定指针）。

列表分为单链表和双链表（singly– or doubly–linked）。

A list ADT supports: referring, update (both insert and delete) aswell as searching methods.

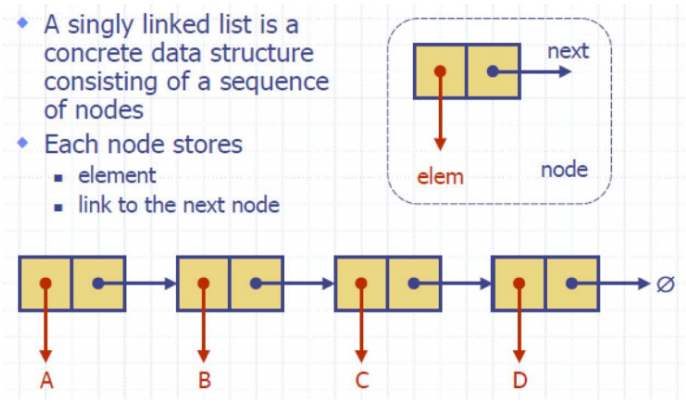
列表支持以下的方法：

- first(): Return position of first element; error occurs if list S is empty. 返回第一个元素的位置，空列表报错。
- last(): Return the position of the last element; error occurs if list S is empty. 返回最后一个元素的位置，空列表报错。
- isFirst(p): Return true if element p is first item in list, false otherwise. 判断元素是否为列表中第一个元素。
- isLast(p): Return true is element p is last element in list, false otherwise. 判断元素是否为列表中最后一个元素。
- before(p): Return the position of the element in S preceding the one at position p; error if p is first element. 返回列表中p前面的元素，如果p是第一个则报错。
- after(p): Return the position of the element in S following the one at position p; error if p is last element. 返回列表中p后面的元素，如果p是最后一个则报错。

列表的更新方法：

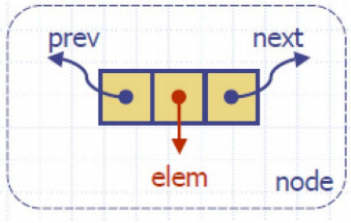
- replaceElement(p,e): p - position, e - element.
- swapElements(p,q): p,q - positions.
- insertFirst(e): e - element.
- insertLast(e): e - element.
- insertBefore(p,e): p - position, e - element.
- insertAfter(p,e): p - position, e - element.
- remove(p): p - position.

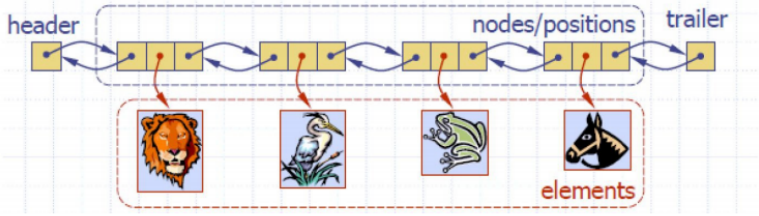
A node in a singly–linked list stores a next link pointing to next element in list (null if element is last element). 单链表的节点储存指向下一个元素的指针。



双链表除此之外还有一个指向上一个元素的指针。

- A doubly linked list provides a natural implementation of the List ADT
- Nodes implement Position and store:
  - element
  - link to the previous node
  - link to the next node
- Special trailer and header nodes



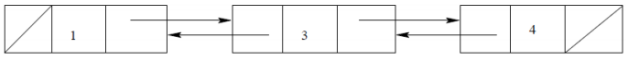


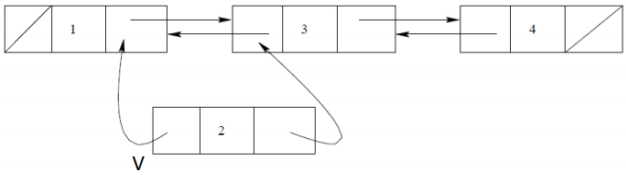
From now on, we will concentrate on doubly-linked lists.

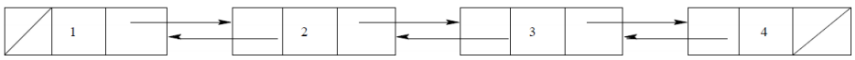
插入元素的例子：

How to insert an element?

Example: *insertAfter(1,e)*



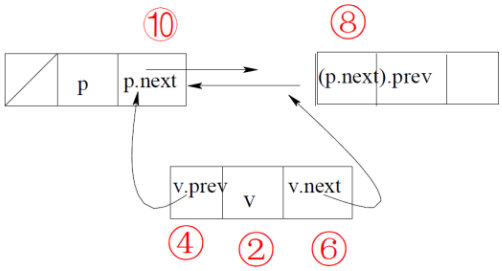




Pseudo-code for *insertAfter(p,e)* :

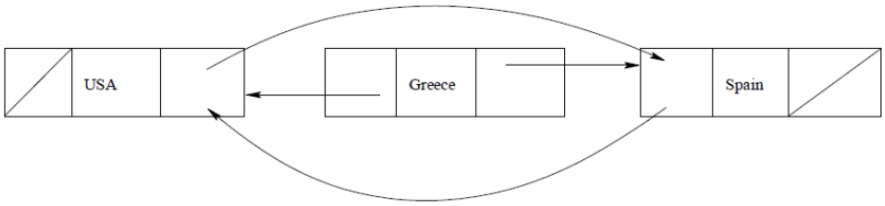
INSERTAFTER(*p,e*)

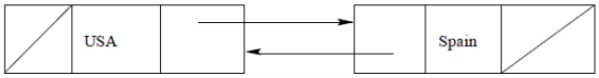
```
    //Create a new node v
2   v.element ← e
    //Link v to its predecessor
4   v.prev ← p
    //Link v to its successor
6   v.next ← p.next
    //Link p's old successor to v
8   (p.next).prev ← v
    //Link p to its new successor v
10  p.next ← v
11  return v
```



移除元素的例子：

Example: *remove(2)*





The pseudo-code for *remove(p)*:

REMOVE(*p*)

```
    //Assign a temporary variable to hold return value
2   t ← p.element
    //Unlink p from list
4   (p.prev).next ← p.next
5   (p.next).prev ← p.prev
    //invalidate p
7   p.prev ← null
8   p.next ← null
9   return t
```