

[**数 据 结 构**](javascript:setQuery('QueryName','%E6%95%B0%E6%8D%AE%E5%BA%93%E6%8A%80%E6%9C%AF%E5%8F%8A%E5%BA%94%E7%94%A8B'))

**第一次作业**

学院： 交通运输与物流学院

姓名： 刘欣豪

学号： 2020112921

专业： 交通工程

|  |
| --- |
| 作业得分： |
|  |

班级： 交通4班

指导老师： 葛乾

**选择题：**

1、A.数据元素 B.关系

2、B.元素和元素之间存在某种联系

3、A.逻辑

4、B.可行性、有穷性和确定性

5、C.执行时间正比于n2

6、A

7、C C

8、B

9、B

10、C

**判断题**

1、F 算法的好坏与算法的描述语言有关，**不受**到所使用的计算机影响

2、F 非空线性表头节点无前驱元素，最后一个元素无后继元素

3、T

4、F 前后移动错误，单链表只能一个方向移动

5、F 顺序栈指元素的存储结构相连，与元素大小无关

**填空题**

1. O（1）
2. min（n，m）
3. O（1） O（n）
4. q.prior=p.prior ， q.prior.next=q ， q.next=p ， p.prior=q
5. 1 n top1-top2=1

**简答题**

1、

主要考虑三点：**数据对象之间的关系**、**选用数据结构通用的时间复杂度**、**空间复杂度** 。

* 数据结构是指一组相互之间存在一种或多种特定逻辑关系的数据元素集合，考虑问题对象的关系是一对一还是其它复杂的网络结构。
* “时间”为一个算法从开始到执行完毕所需要的用时。当该问题对于时间要求很高，那么“时间”优先级别较高。
* “空间”是指执行完一个算法所需要消耗的内存大小。

2、

Ⅰ. 顺序表存储是将数据元素放到一块连续的内存存储空间，存取效率高，速度快。但是不可以动态增加长度

优点：存取速度高效

缺点：插入和删除比较慢

Ⅱ. 链表存储是在程序运行过程中动态的分配空间，只要存储器还有空间，就不会发生存储溢出问题

优点：插入和删除速度快，保留原有的物理顺序

缺点：查找速度慢

从其的存储优缺点来看，各自有各自的使用场景。如：很少的插入和删除操作但频繁的查找可以用顺序表存储，而频繁的插入和删除操作很少的查询就可以使用链表存储。

3、1到n范围内除2、3的所有整数

4、O()

5、O（n3）

**编程题**

1.

**样例输入与输出：**

输入样例Ⅰ：9 5 5 4 6 5 1 2（最大值在头部）

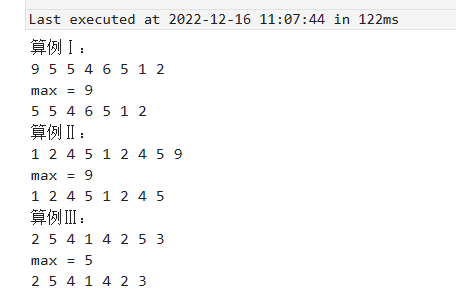
输出样例Ⅰ：5 5 4 6 5 1 2

输入样例Ⅱ：1 2 4 5 1 2 4 5 9（最大值在尾部）

输出样例Ⅱ：1 2 4 5 1 2 4 5

输入样例Ⅲ：2 5 4 1 4 2 5 3（最大值存在多个）

输出样例Ⅲ：2 5 4 1 4 2 3



**代码：**

1. **class** Node:
2. **def** \_\_init\_\_(self,value):
3. self.value = value
4. self.next = None
6. **class** List:
7. **def** \_\_init\_\_(self):
8. self.head = None
10. # 建立一个循环链表，头节点为-1
11. **def** create(self,data):
12. **if** len(data)<=0:
13. **return**
14. self.head = Node(-1)
15. t = self.head
16. **for** i **in** data:
17. t.next = Node(i) # 根据列表中数据建立循环链表
18. t = t.next
19. t.next = self.head # 将链表尾部与头部链接
21. **def** show(self):
22. **if** self.head.next == None:
23. **print**('empty!')
24. **return**
25. **print**(self.head.value,end=' ')
26. p = self.head
27. **while**(p.next!=self.head):
28. p = p.next
29. **print**(p.value,end=' ')
30. **print**(end='\n')
32. **def** fun1(self):
33. **if** self.head == None:
34. **print**('empty!')
35. **return**
36. p = self.head.next
37. mmax\_p = self.head
38. **while**(p.next!=self.head):
39. **if** mmax\_p.next.value<=p.next.value:
40. mmax\_p = p
41. p = p.next
42. **print**('max = %d'%mmax\_p.next.value)
43. mmax\_p.next = mmax\_p.next.next

46. **if** \_\_name\_\_ == "\_\_main\_\_":
48. test\_data1 = [9,5,5,4,6,5,1,2]
49. test\_data2 = [1,2,4,5,1,2,4,5,9]
50. test\_data3 = [2,5,4,1,4,2,5,3]
51. **print**('算例Ⅰ：')
52. L1 = List()
53. L1.create(test\_data1)
54. L1.show()
55. L1.fun1()
56. L1.show()
57. **print**('算例Ⅱ：')
58. L2 = List()
59. L2.create(test\_data2)
60. L2.show()
61. L2.fun1()
62. L2.show()
63. **print**('算例Ⅲ：')
64. L3 = List()
65. L3.create(test\_data3)
66. L3.show()
67. L3.fun1()
68. L3.show()

2、

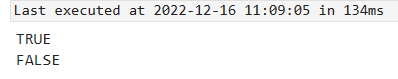
**样例输入与输出：**

输入样例Ⅰ：abc@cba

输出样例Ⅰ：TRUE

输入样例Ⅱ：abc@cca

输出样例Ⅱ：FALSE



**代码：**

1. **class** Stack:
2. **def** \_\_init\_\_(self):
3. self.data = []
5. **def** empty(self):
6. **if** len(self.data)==0:
7. **return** True
8. **return** False
10. **def** push(self,e):
11. self.data.append(e)
13. **def** create(self,data):
14. **if** type(data)!=str:
15. **print**('typeERROR')
16. **return**
17. **for** i **in** data:
18. self.push(i)
19. **def** pop(self):                  #元素出栈
20. **assert** **not** self.empty()     #检测栈为空
21. **return** self.data.pop()
23. **def** gettop(self):               #取栈顶元素
24. **assert** **not** self.empty()     #检测栈为空
25. **return** self.data[len(self.data)-1]
27. **def** fun1(self):
28. **assert** **not** self.empty()     #检测栈为空
29. p = self
30. j = []
31. **while** p.gettop()!='@':
32. j.append(p.pop())
33. p.pop()
34. **if** j==p.data:
35. **print**('TRUE')
36. **else**:
37. **print**('FALSE')
39. **if** \_\_name\_\_ =="\_\_main\_\_":
40. S1 = Stack()
41. str1 = 'abc@cba'
42. S1.create(str1)
43. S1.fun1()
45. str2 = 'abc@cca'
46. S2 = Stack()
47. S2.create(str2)
48. S2.fun1()

**大作业：**

**样例输入与输出：**

输入样例：

<body>

<center>

<h1> The Little Boat </h1>

</center>

<p> The storm tossed the little

boat like a cheap sneaker in an

old washing machine . The three

drunken f ishermen were used to

such treatment, of course, but

not the tree salesman, who even as

a stowaway now felt that he

had overpaid for the voyage. </p>

<ol>

<li> Will the salesman die? </li>

<li> What color is the boat? </li>

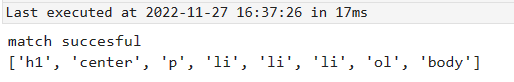
<li> And what about Naomi? </li></ol>

</body>

样例输出：

match succesful

['h1', 'center', 'p', 'li', 'li', 'li', 'ol', 'body']



**代码：**

1. **class** Stack:
2. **def** \_\_init\_\_(self):
3. self.data=[]
5. **def** empty(self):
6. **if** len(self.data)==0:
7. **return** True
8. **return** False
10. **def** push(self,c):
11. self.data.append(c)
13. **def** pop(self):
14. **assert** **not** self.empty()
15. **return** self.data.pop()
17. **def** gettop(self):
18. **assert** **not** self.empty()
19. **return** self.data[len(self.data)-1]
21. **def** match(txt):
22. s = Stack()
23. values = []
24. st = txt.find('<',0)
25. en = txt.find('>',0)
26. **while** st!=-1 **and** en!=-1:
27. temp = txt[st+1:en]
28. **if** temp.find('/')==-1 **and** len(temp)!=0:
29. s.push(temp)
30. **elif** temp.find('/')!=-1:
31. **if** s.gettop()==temp.strip('/'):
32. values.append(s.pop())
33. **else**:
34. **print**('not match')
35. **return**
36. st = txt.find('<',st+1)
37. en = txt.find('>',en+1)
38. **if** s.empty:
39. **print**('match succesful')
40. **print**(values)
42. **if** \_\_name\_\_ == '\_\_main\_\_':
43. with open('data.txt','r') as f:
44. txt = f.read()
45. f.close()
46. match(txt)