选10、链表的类实现

1、面向过程与面向对象

比如要写一个五子棋程序，可以从两个角度寻求解决问题的方法

一种方法是考虑整个流程，从数据输入、逻辑判断、输出等方面逐一编写相关代码。这便是面向过程的。

还有一种方法是，将一个五子棋游戏分为：黑白双方（负责输入），棋盘系统（用于呈现），规则系统（用于判定），这种方法是将整个游戏看作一个个的对象来处理，称为面向对象。

2、类与对象

在面向对象中，最重要的概念就是类与对象。

类是对某一类具有相似属性事物的抽象，对象是类的实例化。

比如我们都称为人类，而具体某个个体，则是人类的一个实例化对象。

在一个类中，包含着属性与方法。比如人的身高和颜值都是属性，人会笑，笑是一种方法。

说白了，属性是变量，方法是函数。

3、python中类的实现

在Python中，定义一个类，其基本语法为：

def <类名称>:

[属性变量]

def init(self[,初始化参数]): #构造函数，在类进行实例化时就被调用执行

`````

def func1(self[,参数])): #成员函数，即某个方法。

`````

在具体代码中，可以定义一个类的实例化对象。如：

model=<类名称>([初始化参数])

其实在Python中，许多变量本身就是通过类来实现的，比如list

4、链表的类实现：

```
class linkstruct:
    data=[]
    head=-1
    length=0
    def __init__(self):
        self.data=[]
        self.head=-1
        self.length=0
```

```
    def add(self,p,n): #在数据节点p后插入n
        q=k=self.head
        if self.length==0:
            self.data.append([n,self.head])
            self.head=0
        else:
            while k!=-1:
                if self.data[k][0]!=p:
                    q=k
                    k=self.data[k][1]
                else:
                    self.data.append([n,self.data[k][1]])
                    self.data[k][1]=len(self.data)-1
                    break
            if k==-1:
                self.data.append([n,-1])
                self.data[q][1]=len(self.data)-1
        self.length+=1
```

```
    def add(self,p,n): #在数据节点p后插入n
        q=k=self.head
        if self.length==0:
            self.data.append([n,self.head])
            self.head=0
        else:
            while k!=-1:
                if self.data[k][0]!=p:
                    q=k
                    k=self.data[k][1]
                else:
                    self.data.append([n,self.data[k][1]])
                    self.data[k][1]=len(self.data)-1
                    break
            if k==-1:
                self.data.append([n,-1])
                self.data[q][1]=len(self.data)-1
        self.length+=1
```

```
    def show(self):
        res=[]
        k=self.head
        while k!=-1:
            res.append(self.data[k][0])
            k=self.data[k][1]
        return res
    def delete(self,n):
        k=self.head
        while k!=-1:
            if self.data[k][0]!=n:
                q=k
                k=self.data[k][1]
            else:
                if k==self.head:
                    self.head=self.data[k][1]
                    self.data[head]=None
                else:
                    self.data[q][1]=self.data[k][1]
                    self.data[k]=None
                self.length-=1
                break
```

调用：

```
ls=linkstruct()
ls.add(6,1)
ls.add(6,3)
ls.add(6,5)
ls.add(1,2)
ls.add(2,3)
print(ls.show())
ls.delete(5)
print(ls.show())
```

```
[1, 2, 3, 2, 3, 1, 3, 3, 5]
[1, 2, 3, 2, 3, 1, 3, 3]
```