《大学计算机基础》（常规班）实验指导书

# 实验4 问题的描述——数据结构

* 注意：每个实验任务需要创建单独创建Python文件，命名规则是：d\_实验任务尾号.py
* 例：本次实验的任务4-1~4-3应该创建3个py文件，文件名分别是：d\_1.py~ d\_3.py
* 每次实验后，需按照模板完成实验报告，并提交（本次实验报告命名规则是：

“实验4\_学号姓名.docx”,例：“实验4\_12051211王一”）。

## 1. 实验目的

（1）了解如何使用简单和较为复杂的数据结构描述问题。

## 2. 实验任务

### 实验任务4-1 学生信息统计

基于实验3-1中的表格，建立一个类，并编写代码对其进行测试。要求调用该类能完成以下问题：

（1）能够构造任意一名学生的成绩信息，包括姓名、学号和六科成绩。

（2）提供类方法计算某学生六科成绩的平均分。

（3）提供类方法求某学生总成绩的等级。总成绩等级按照平均分，定义如下：

|  |  |
| --- | --- |
| 平均分x | 等级 |
| 90<=x<=100 | A |
| 80<=x<90 | B |
| 70<=x<80 | C |
| 60<=x<70 | D |
| x<60 | F |

（4）提供类方法打印某学生的成绩信息，输出格式如下：

**姓名(学号：XXX) 语XX 数XX 英XX 物XX 化XX 生XX 平均分XX 等级X**

（5）编写测试程序，利用实验3-1表格中的数据，对你编写的类进行测试。

**要求：运行程序后，输入实验3-1表格中不同的数据，验证在各种情况下你的程序是否正确；并将程序运行结果截图后，粘贴到实验报告中。**

**实验目的：**

（1）理解类的定义；

（2）会定义类方法；

（3）会调用所定义的类。

**实验指导：**

1. 定义学生类，包含姓名、学号、各科成绩等属性：

①构造函数传入姓名、学号及各科成绩等参数；

②按照题目要求编写计算平均分、等级和输出的成员函数。

1. 在主函数中利用列表储存类的实例，对每个实例进行测试。

**参考代码：**

#4-1.py

#本练习要求：定义一个类，完成指定功能。

class Student: #定义一个学生类

def \_\_init\_\_(self,name,num,chn,math,eng,phy,chem,bio): #定义构造函数

self.name=name

self.number=num

self.chn=chn

self.math=math

self.eng=eng

self.phy=phy

self.chem=chem

self.bio=bio

def aver\_score(self): #计算平均分（请自行补充完整）

……

return score

def aver\_grade(self): #计算等级（请自行补充完整）

if self.aver\_score()>=90 and self.aver\_score()<=100: #若平均分在90~100，则为A

grade="A"

……

return grade

def print\_student(self): #按照格式输出学生信息（请自行补充完整）

……

if \_\_name\_\_=="\_\_main\_\_":#测试开始（请自行补充完整）

Stu\_List=[] #采用列表保存类的实例

……

程序运行结果如下：



### 实验任务4-2 大整数加法的实现

编写一个程序，实现两大整数的加法（两整数具有相同的位数）。注意不可直接使用加法表达式。

**要求：多次运行程序，验证输入不同的数值，程序能否得到正确统计结果；并将程序运行结果截图后，粘贴到实验报告中。**

**实验目的：**

学习使用合适的数据结构来解决相关问题。

**实验指导：**

1. 问题分析

实现两大整数相加，可模拟做竖式的方法，从个位开始相加，满10则向前进位。因为在输入数据时我们通常由高位到低位输入，而在这里使用数据时需由低位到高位取出，很显然满足“后进先出”的规则，可采用栈来完成此题目。

1. 具体步骤

①第一个加数数据按照输入顺序（从高位到低位）入栈1，此时栈顶为最低位；

②第二个加数数据按照输入顺序（从高位到低位）入栈2，此时栈顶为最低位；

③将栈1、栈2均弹出栈顶，并作加法，考虑进位，计算结果入栈3，这时栈3正好是低位入栈；

④将栈3出栈，正好是由高位到低位的计算结果。

**注：实现输入的数字按照由高位到低位入栈，可采取将输入数字的字符串每一位与字符”0”的ASCII码值作差的方法**，如：

str = “123”

list=[]

for c in str

list.append(ord(c)-ord(“0”))

则列表list = [1,2,3]

1. 如果你对列表的属性和方法比较熟悉，可以将列表当作堆栈使用。此时会用到列表的append()、pop()方法。这两个方法具体使用的例子如下：

>>> st = [3, 4, 5]

>>> st.append(6)

>>> st.append(7)

>>> st

[3, 4, 5, 6, 7]

>>> st.pop()

7

>>> st

[3, 4, 5, 6]

>>> st.pop()

6

>>> st.pop()

5

>>> st

[3, 4]

你也可以用课本所给出的stack类来完成此问题，类的定义如下：

#Stack.py

class Stack:

#创建一个空栈

def \_\_init\_\_(self):

self.items=[]

#元素入栈

def push(self,item):

self.items.append(item)

#元素出栈

def pop(self):

return self.items.pop()

#返回栈顶的元素

def peek(self):

return self.items[len(self.items)-1]

#判断栈是否为空

def isEmpty(self):

return self.items==[]

#返回栈的大小

def size(self):

return len(self.items)

**参考代码：**

#4-2.py

#利用栈实现大整数加法

#栈的实现

……

#主函数入口

if \_\_name\_\_=="\_\_main\_\_":

#输入两个加数

……

#建立建立加数栈1、加数栈2和结果栈

……

#加数1和加数2按照由高位到低位分别入栈

……

tmp=0

while …… : #栈1和栈2均非空时执行循环

tmp += …… #取栈1和栈2的栈顶相加，存入临时变量tmp

…… #栈1弹出栈顶

…… #栈2弹出栈顶

…… #将tmp除以10的余数压入结果栈

tmp = …… #取tmp除以10的商，结果只能为0和1,1代表有进位

if tmp: #最高位相加后若有进位则将进位结果压入结果栈

……

#结果输出

……

程序运行结果如下：



### 实验任务4-3 重复密钥加密法的实现

凯撒加密法是一种简单的消息编码方式，它是按照字母表将消息中的每个字母移动常量的k位。但是这种方式极易破解，因为字母的移动只有26种可能。

因此，为了解决这一问题，我们提出了重复密钥加密法：这时不是将每个字母移动常数位，而是利用一个密钥值列表，将各个字母移动不同的位数。如果消息比密钥值长，可以从头再使用这个密钥值列表。

请编写一个程序，实现重复密钥加密法，并进行解密，以验证方法的正确性。为简化问题，约定移动的范围在整个ASCII码表内，密钥值可为正、负、零，加密时将原字符的ASCII码值加上密钥值即可，规定密钥值的绝对值不超过15。

**要求：多次运行程序，验证输入不同的原文，程序能否得到正确密文和解密结果；并将程序运行结果截图后，粘贴到实验报告中。**

**实验目的：**

学习使用合适的数据结构来解决相关问题。

**实验指导：**

1. 问题分析

加密（或解密）本身的方法非常简单，只需将原文（或密文）的ASCII码值加上（或减去）对应的密钥值即可，问题的重点在于密钥的重复性。按照题目要求，每使用完一个密钥值，则将其放在密钥值列表的最后，实现循环使用。由此可知其满足“先进先出”的规则，可使用队列来方便地完成此题目。

1. 具体步骤

①给出密钥列表和待加密的原文；

②建立两个队列，分别储存一份密钥，模拟加密者使用一份密钥，解密者使用一份密钥；

③依次取一个密钥值，对原文加密，再将密钥值放回队列；

④输出加密后的密文，再按照同样的方法进行解密，输出解密后的原文，进行比较。

1. 如果你对列表的属性和方法比较熟悉，可以将列表当作队列使用。此时会用到列表的append()、pop()方法。这两个方法具体使用的例子如下：

>>> queue = ['a', 'b', 'c']

>>> queue.append('d')

>>> queue.append('e')

>>> queue

['a', 'b', 'c', 'd', 'e']

>>> queue.pop(0)

'a'

>>> queue

['b', 'c', 'd', 'e']

>>> queue.pop(0)

'b'

>>> queue

['c', 'd', 'e']

你也可以用课本所给出的queue类来完成此问题，类的定义如下：

#queue\_class.py

class Queue:

def \_\_init\_\_(self):

self.items = []

def isEmpty(self):

return self.items == []

def enqueue(self,item):

self.items.insert(0,item)

def dequeue(self):

return self.items.pop()

def size(self):

return len(self.items)

**参考代码：**

#4-3.py

#利用队列实现重复密钥加密法

#队列的实现

……

#主函数入口

if \_\_name\_\_=='\_\_main\_\_':

…… #密钥列表

…… #输入待加密原文

str\_encode="" #密文字符串

str\_decode="" #解密字符串

#将密钥值分别储存在加密队列和解密队列中

……

#加密过程实现，包括从队列中取密钥值、进行加密和将用完的密钥值放入队列

……

#解密过程实现，包括从队列中取密钥值、进行解密和将用完的密钥值放入队列

……

#输出过程

print("加密后的密文为：%s" %str\_encode)

print("解密后的原文为：%s" %str\_decode)

**注意：以上代码结构适用于利用编写好的队列类来实现队列。若选择用列表实现队列，则可直接在主程序中将列表用作队列，无需主程序前的队列实现部分。**

程序运行结果如下：

