第一部分 练习题及参考答案

1.1 第1章 绪论

扫一扫

习题+答案

1.2 第2章 线性表

扫一扫

习题+答案

1.3 第3章 栈和队列

扫一扫

习题+答案

1.4 第4章 串和数组

扫一扫

习题＋答案

1.5 第5章 递归

扫一扫

习题＋答案

1.6 第6章 树和二叉树

扫一扫

习题＋答案

1.7 第7章 图

扫一扫

习题＋答案

1.8 第8章 查找

扫一扫

习题+答案

1.9 第9章 排序

扫一扫

习题+答案

第二部分 上机实验题及参考答案

2.1 第1章 绪论

说明：本节所有上机实验题的程序文件位于 ch1 文件夹中。

2.1.1 上机实验题

1. 编写一个 Python 程序，求一元二次方程 $ax^2+bx+c=0$ 的根，并采用相关数据测试。

2. 求 1+(1+2)+(1+2+3)+…(1+2+3+…+n)之和有以下 3 种解法。

解法 1：采用两重迭代，依次求出(1+2+…+i)($1\leqslant i\leqslant n$)后累加。

解法 2：将解法简化为采用一重迭代实现求和。

解法 3：直接利用公式 $n(n+1)(n+2)/6$ 求和。

编写一个 Python 程序，利用上述 3 种解法求 $n=50\ 000$ 时的结果，并且给出各种解法的执行时间。

2.1.2 上机实验题参考答案

1. 解：设计 Exp1 类，它包含 a、b、c 和 ans 属性，其中 ans 列表存放求解结果；另外设计 solve()方法求方程的根，设计 disp()方法输出结果。对应的实验程序 Exp1.py 如下。

```
import math
class Exp1:
    def __init__(self,a1,b1,c1):                    #构造方法
        self.a=a1
        self.b=b1
        self.c=c1
        self.ans=[]
    def solve(self):                                #求方程的根
        d=self.b*self.b-4*self.a*self.c;
        if math.fabs(d)<=0.0001:                    #等于0
            x1=(-self.b+math.sqrt(d))/(2*self.a)
```

```
            self.ans.append(x1)
        elif (d>0):
            x1=(-self.b+math.sqrt(d))/(2*self.a)
            self.ans.append(x1)
            x2=(-self.b-math.sqrt(d))/(2*self.a)
            self.ans.append(x2)
    def disp(self):                                    #输出结果
        if len(self.ans)==0:
            print("无根")
        elif len(self.ans)==1:
            print("一个根为%.1f" %(self.ans[0]))
        else:
            print("两个根为%.1f和%.1f" %(self.ans[0],self.ans[1]))

#主程序
print("\n 测试 1")
a,b,c=2,-3,4
s=Exp1(a,b,c)
s.solve()
print("  a=%.1f,b=%.1f,c=%.1f " %(a,b,c),end=': ')
s.disp()

print("\n 测试 2")
a,b,c=1,-2,1
s=Exp1(a,b,c)
s.solve()
print("  a=%.1f,b=%.1f,c=%.1f " %(a,b,c),end=': ')
s.disp()

print("\n 测试 3")
a,b,c=2,-1,-1
s=Exp1(a,b,c)
s.solve()
print("  a=%.1f,b=%.1f,c=%.1f " %(a,b,c),end=': ')
s.disp()
```

上述程序的执行结果如图 2.1 所示。

```
测试1
  a=2.0, b=-3.0, c=4.0 : 无根

测试2
  a=1.0, b=-2.0, c=1.0 : 一个根为1.0

测试3
  a=2.0, b=-1.0, c=-1.0 : 两个根为1.0和-0.5
```

图 2.1　第 1 章实验题 1 的执行结果

2. 解：对应的实验程序 Exp2.py 如下。

```
import time
class Exp2:
    def solve1(self,n):                                #解法 1
        sum=0
        for i in range(1,n+1):
```

```
            for j in range(1,i+1):
                sum+=j;
        return sum
    def solve2(self,n):                                    #解法 2
        sum,sum1=0,0
        for i in range(1,n+1):
            sum1+=i
            sum+=sum1
        return sum
    def solve3(self,n):                                    #解法 3
        sum=n*(n+1)*(n+2)//6
        return sum

#主程序
n=50000
s=Exp2()
print("\n n=%d\n" %(n))
t1 = time.time()                                           #获取开始时间
print(" 解法 1 sum1=%d" %(s.solve1(n)))
t2 = time.time()                                           #获取结束时间
print(" 运行时间: %ds" %(t2-t1))
t1 = time.time()                                           #获取开始时间
print(" 解法 2 sum2=%d" %(s.solve2(n)))
t2 = time.time()                                           #获取结束时间
print(" 运行时间: %ds" %(t2-t1))
t1 = time.time()                                           #获取开始时间
print(" 解法 3 sum3=%d" %(s.solve3(n)))
t2 = time.time()                                           #获取结束时间
print(" 运行时间: %ds" %(t2-t1))
```

上述程序的执行结果如图 2.2 所示。解法 1 的时间复杂度为 $O(n^2)$，解法 2 的时间复杂度为 $O(n)$，解法 3 的时间复杂度为 $O(1)$。

图 2.2　第 1 章实验题 2 的执行结果

2.2　第 2 章　线性表

说明：本节所有上机实验题的程序文件位于 ch2 文件夹中。

2.2.1　基础实验题

1. 设计整数顺序表的基本运算程序，并用相关数据进行测试。
2. 设计整数单链表的基本运算程序，并用相关数据进行测试。

3. 设计整数双链表的基本运算程序，并用相关数据进行测试。
4. 设计整数循环单链表的基本运算程序，并用相关数据进行测试。
5. 设计整数循环双链表的基本运算程序，并用相关数据进行测试。

2.2.2 基础实验题参考答案

1. 解：顺序表的基本运算算法的设计原理参见《教程》中的 2.2.2 节。包含顺序表基本运算算法类 SqList 以及测试主程序的 Exp1-1.py 文件如下：

```
class SqList:                                  #顺序表类
    def __init__(self):                        #构造函数
        self.initcapacity=5;                   #初始容量设置为 5
        self.capacity=self.initcapacity        #容量设置为初始容量
        self.data=[None] * self.capacity       #设置顺序表的空间
        self.size=0                            #长度设置为 0

    def resize(self,newcapacity):              #改变顺序表的容量为 newcapacity
        assert newcapacity>=0                  #检测参数正确性的断言
        olddata=self.data
        self.data=[None] * newcapacity
        self.capacity=newcapacity
        for i in range(self.size):
            self.data[i]=olddata[i]

    def CreateList(self,a):                    #由数组 a 中的元素整体建立顺序表
        self.size=0
        for i in range(0,len(a)):
            if self.size==self.capacity:       #出现上溢出时
                self.resize(2 * self.size);    #扩大容量
            self.data[self.size]=a[i]
            self.size+=1                       #添加后元素个数增加 1

    def Add(self,e):                           #在线性表的末尾添加一个元素 e
        if self.size==self.capacity:
            self.resize(2 * self.size)         #顺序表空间满时倍增容量
        self.data[self.size]=e                 #添加元素 e
        self.size+=1                           #长度增 1

    def getsize(self):                         #返回长度
        return self.size

    def __getitem__(self,i):                   #求序号为 i 的元素
        assert 0<=i<self.size                  #检测参数 i 正确性的断言
        return self.data[i]

    def __setitem__(self,i,x):                 #设置序号为 i 的元素
        assert 0<=i<self.size                  #检测参数 i 正确性的断言
        self.data[i]=x

    def GetNo(self,e):                         #查找第一个为 e 的元素的序号
        i=0
        while i<self.size and self.data[i]!=e:  #查找元素 e
```

```
            i+=1
        if (i>=self.size):
            return -1;                                  #未找到时返回-1
        else:
            return i;                                   #找到后返回其序号

    def Insert(self,i,e):                               #在线性表中序号为 i 的位置插入元素 e
        assert 0<=i<=self.size                          #检测参数 i 正确性的断言
        if self.size==self.capacity:
            self.resize(2*self.size)                    #满时倍增容量
        for j in range(self.size,i,-1):                 #将 data[i]及后面的元素后移一个位置
            self.data[j]=self.data[j-1]
        self.data[i]=e                                  #插入元素 e
        self.size+=1                                    #长度增 1

    def Delete(self,i):                                 #在线性表中删除序号为 i 的元素
        assert 0<=i<=self.size-1                        #检测参数 i 正确性的断言
        for j in range(i,self.size-1):
            self.data[j]=self.data[j+1]                 #将 data[j]之后的元素前移一个位置
        self.size-=1                                    #长度减 1
        if self.capacity>self.initcapacity and self.size<=self.capacity/4:
            self.resize(self.capacity//2)               #满足要求时容量减半

    def display(self):                                  #输出顺序表
        for i in range(0,self.size):
            print(self.data[i],end=' ')
        print()

if __name__ == '__main__':
    L=SqList()
    print()
    print("建立空顺序表 L,其容量=%d" %(L.capacity))
    a=[1,2,3,4,5,6]
    print("  1～6 创建 L")
    L.CreateList(a)
    print("  L[容量=%d,长度=%d]: " %(L.capacity,L.getsize()),end='',L.display()
    print("  插入 6～10")
    for i in range(6,11):
        L.Add(i)
    print("  L[容量=%d,长度=%d]: " %(L.capacity,L.getsize()),end='',L.display()
    print("  序号为 2 的元素=%d" %(L[2]))
    print("  设置序号为 2 的元素为 20")
    L[2]=20
    print("  L[容量=%d,长度=%d]: " %(L.capacity,L.getsize()),end='',L.display()
    x=6
    print("  第一个值为%d 的元素序号=%d" %(x,L.GetNo(x)))
    n=L.getsize()
    for i in range(n-2):
        print("删除首元素")
        L.Delete(0)
        print("L[容量=%d,长度=%d]: " %(L.capacity,L.getsize()),end='',L.display()
```

上述程序的执行结果如图 2.3 所示。

```
建立空顺序表L，其容量=5
1～6创建L
L[容量=10,长度=6]: 1 2 3 4 5 6
插入6～10
L[容量=20,长度=11]: 1 2 3 4 5 6 6 7 8 9 10
序号为2的元素=3
设置序号为2的元素为20
L[容量=20,长度=11]: 1 2 20 4 5 6 6 7 8 9 10
第一个值为6的元素序号=5
删除首元素
L[容量=20,长度=10]: 2 20 4 5 6 6 7 8 9 10
删除首元素
L[容量=20,长度=9]: 20 4 5 6 6 7 8 9 10
删除首元素
L[容量=20,长度=8]: 4 5 6 6 7 8 9 10
删除首元素
L[容量=20,长度=7]: 5 6 6 7 8 9 10
删除首元素
L[容量=20,长度=6]: 6 6 7 8 9 10
删除首元素
L[容量=10,长度=5]: 6 7 8 9 10
删除首元素
L[容量=10,长度=4]: 7 8 9 10
删除首元素
L[容量=10,长度=3]: 8 9 10
删除首元素
L[容量=5,长度=2]: 9 10
```

图 2.3　第 2 章基础实验题 1 的执行结果

2. 解：单链表的基本运算算法的设计原理参见《教程》中的 2.3.2 节。包含单链表基本运算算法类 LinkList 以及测试主程序的 Exp1-2.py 文件如下：

```
class LinkNode:                          #单链表结点类
    def __init__(self,data=None):        #构造函数
        self.data=data                   #data 属性
        self.next=None                   #next 属性

class LinkList:                          #单链表类
    def __init__(self):                  #构造函数
        self.head=LinkNode()             #头结点 head
        self.head.next=None

    def CreateListF(self,a):             #头插法：由数组 a 整体建立单链表
        for i in range(0,len(a)):        #循环建立数据结点 s
            s=LinkNode(a[i])             #新建存放 a[i]元素的结点 s
            s.next=self.head.next        #将 s 结点插入开始结点之前、头结点之后
            self.head.next=s

    def CreateListR(self,a):             #尾插法：由数组 a 整体建立单链表
        t=self.head                      #t 始终指向尾结点,开始时指向头结点
        for i in range(0,len(a)):        #循环建立数据结点 s
            s=LinkNode(a[i])             #新建存放 a[i]元素的结点 s
            t.next=s                     #将 s 结点插入 t 结点之后
            t=s
        t.next=None                      #将尾结点的 next 成员置为 None

    def geti(self,i):                    #返回序号为 i 的结点
        p=self.head
        j=-1
        while (j<i and p is not None):
            j+=1
            p=p.next
```

```
        return p

    def Add(self,e):                          #在线性表的末尾添加一个元素 e
        s=LinkNode(e)                         #新建结点 s
        p=self.head
        while p.next is not None:             #查找尾结点 p
            p=p.next
        p.next=s;                             #在尾结点之后插入结点 s

    def getsize(self):                        #返回长度
        p=self.head
        cnt=0
        while p.next is not None:             #找到尾结点为止
            cnt+=1
            p=p.next
        return cnt

    def __getitem__(self,i):                  #求序号为 i 的元素
        assert i>=0                           #检测参数 i 正确性的断言
        p=self.geti(i)
        assert p is not None                  #p 不为空的检测
        return p.data

    def __setitem__(self,i,x):                #设置序号为 i 的元素
        assert i>=0                           #检测参数 i 正确性的断言
        p=self.geti(i)
        assert p is not None                  #p 不为空的检测
        p.data=x

    def GetNo(self,e):                        #查找第一个为 e 的元素的序号
        j=0
        p=self.head.next
        while p is not None and p.data!=e:
            j+=1                              #查找元素 e
            p=p.next
        if p is None:
            return -1                         #未找到时返回-1
        else:
            return j                          #找到后返回其序号

    def Insert(self,i,e):                     #在线性表中序号为 i 的位置插入元素 e
        assert i>=0                           #检测参数 i 正确性的断言
        s=LinkNode(e)                         #建立新结点 s
        p=self.geti(i-1)                      #找到序号为 i-1 的结点 p
        assert p is not None                  #p 不为空的检测
        s.next=p.next                         #在 p 结点的后面插入 s 结点
        p.next=s

    def Delete(self,i):                       #在线性表中删除序号为 i 的位置的元素
        assert i>=0                           #检测参数 i 正确性的断言
        p=self.geti(i-1)                      #找到序号为 i-1 的结点 p
        assert p.next is not None             #p.next 不为空的检测
        p.next=p.next.next;                   #删除 p 结点的后继结点
```

```
    def display(self):                                    #输出线性表
        p=self.head.next
        while p is not None:
            print(p.data,end=' ')
            p=p.next;
        print()

if __name__ == '__main__':
    L=LinkList()
    print()
    print("  建立空单链表 L")
    a=[1,2,3,4,5,6]
    print("  1～6 创建 L")
    L.CreateListR(a)
    print("  L[长度=%d]: " %(L.getsize()),end='');L.display()
    print("  插入 6～10")
    for i in range(6,11):
        L.Add(i)
    print("  L[长度=%d]: " %(L.getsize()),end='');L.display()
    print("  序号为 2 的元素=%d" %(L[2]))
    print("  设置序号为 2 的元素为 20")
    L[2]=20
    print("  L[长度=%d]: " %(L.getsize()),end='');L.display()
    x=6
    print("  第一个值为%d 的元素序号=%d" %(x,L.GetNo(x)))
    n=L.getsize()
    for i in range(n-2):
        print("  删除首元素")
        L.Delete(0)
        print("  L[长度=%d]: " %(L.getsize()),end='');L.display()
```

上述程序的执行结果如图 2.4 所示。

```
建立空单链表L
1～6创建L
L[长度=6]: 1 2 3 4 5 6
插入6～10
L[长度=11]: 1 2 3 4 5 6 6 7 8 9 10
序号为2的元素=3
设置序号为2的元素为20
L[长度=11]: 1 2 20 4 5 6 6 7 8 9 10
第一个值为6的元素序号=5
删除首元素
L[长度=10]: 2 20 4 5 6 6 7 8 9 10
删除首元素
L[长度=9]: 20 4 5 6 6 7 8 9 10
删除首元素
L[长度=8]: 4 5 6 6 7 8 9 10
删除首元素
L[长度=7]: 5 6 6 7 8 9 10
删除首元素
L[长度=6]: 6 6 7 8 9 10
删除首元素
L[长度=5]: 6 7 8 9 10
删除首元素
L[长度=4]: 7 8 9 10
删除首元素
L[长度=3]: 8 9 10
删除首元素
L[长度=2]: 9 10
```

图 2.4　第 2 章基础实验题 2 的执行结果

3. 解：双链表的基本运算算法的设计原理参见《教程》中的 2.3.4 节。包含双链表基本运算算法类 DLinkList 以及测试主程序的 Exp1-3.py 文件如下：

```
class DLinkNode:                          #双链表结点类
    def __init__(self,data=None):         #构造函数
        self.data=data                     #data 属性
        self.next=None                     #next 属性
        self.prior=None                    #prior 属性

class DLinkList:                          #双链表类
    def __init__(self):                    #构造函数
        self.dhead=DLinkNode()             #头结点 dhead
        self.dhead.next=None
        self.dhead.prior=None

    def CreateListF(self,a):               #头插法：由数组 a 整体建立双链表
        for i in range(0,len(a)):          #循环建立数据结点 s
            s=DLinkNode(a[i])              #新建存放 a[i]元素的结点 s,将其插入表头
            s.next=self.dhead.next         #修改 s 结点的 next 成员
            if self.dhead.next!=None:      #修改头结点的非空后继结点的 prior
                self.dhead.next.prior=s
            self.dhead.next=s              #修改头结点的 next
            s.prior=self.dhead             #修改 s 结点的 prior

    def CreateListR(self,a):               #尾插法：由数组 a 整体建立双链表
        t=self.dhead                       #t 始终指向尾结点,开始时指向头结点
        for i in range(0,len(a)):          #循环建立数据结点 s
            s=DLinkNode(a[i])              #新建存放 a[i]元素的结点 s
            t.next=s                       #将 s 结点插入 t 结点之后
            s.prior=t
            t=s
        t.next=None                        #将尾结点的 next 成员置为 None

    def geti(self,i):                      #返回序号为 i 的结点
        p=self.dhead
        j=-1
        while (j<i and p is not None):
            j+=1
            p=p.next
        return p

    def Add(self,e):                       #在线性表的末尾添加一个元素 e
        s=DLinkNode(e)                     #新建结点 s
        p=self.dhead
        while p.next is not None:          #查找尾结点 p
            p=p.next
        p.next=s;                          #在尾结点之后插入结点 s
        s.prior=p

    def getsize(self):                     #返回长度
        p=self.dhead
        cnt=0
        while p.next is not None:          #找到尾结点为止
```

```
        cnt+=1
        p=p.next
    return cnt

def __getitem__(self,i):                    #求序号为 i 的元素
    assert i>=0                             #检测参数 i 正确性的断言
    p=self.geti(i)
    assert p is not None                    #p 不为空的检测
    return p.data

def __setitem__(self,i,x):                  #设置序号为 i 的元素
    assert i>=0                             #检测参数 i 正确性的断言
    p=self.geti(i)
    assert p is not None                    #p 不为空的检测
    p.data=x

def GetNo(self,e):                          #查找第一个为 e 的元素的序号
    j=0
    p=self.dhead.next
    while p is not None and p.data!=e:
        j+=1                                #查找元素 e
        p=p.next
    if p is None:
        return -1                           #未找到时返回-1
    else:
        return j                            #找到后返回其序号

def Insert(self,i,e):                       #在线性表中序号为 i 的位置插入元素 e
    assert i>=0                             #检测参数 i 正确性的断言
    s=DLinkNode(e)                          #建立新结点 s
    p=self.geti(i-1)                        #找到序号为 i-1 的结点 p
    assert p is not None                    #p 不为空的检测
    s.next=p.next                           #修改 s 结点的 next
    if p.next!=None:                        #修改 p 结点的非空后继结点的 prior
        p.next.prior=s
    p.next=s                                #修改 p 结点的 next
    s.prior=p                               #修改 s 结点的 prior

def Delete(self,i):                         #在线性表中删除序号为 i 的位置的元素
    assert i>=0                             #检测参数 i 正确性的断言
    p=self.geti(i)                          #找到序号为 i 的结点 p
    assert p is not None                    #p 不为空的检测
    p.prior.next=p.next                     #修改 p 结点的前驱结点的 next
    if p.next!=None:                        #修改 p 结点的非空后继结点的 prior
        p.next.prior=p.prior

def display(self):                          #输出线性表
    p=self.dhead.next
    while p is not None:
        print(p.data,end='')
        p=p.next;
    print()
```

```
if __name__ == '__main__':
    L=DLinkList()
    print()
    print("  建立空双链表 L")
    a=[1,2,3,4,5,6]
    print("  1～6 创建 L")
    L.CreateListR(a)
    print("  L[长度=%d]: " %(L.getsize()),end='');L.display()
    print("  插入 6～10")
    for i in range(6,11):
        L.Add(i)
    print("  L[长度=%d]: " %(L.getsize()),end='');L.display()
    print("  序号为 2 的元素=%d" %(L[2]))
    print("  设置序号为 2 的元素为 20")
    L[2]=20
    print("  L[长度=%d]: " %(L.getsize()),end='');L.display()
    x=6
    print("  第一个值为%d 的元素序号=%d" %(x,L.GetNo(x)))
    n=L.getsize()
    for i in range(n-2):
        print("  删除首元素")
        L.Delete(0)
        print("  L[长度=%d]: " %(L.getsize()),end='');L.display()
```

上述程序的执行结果如图 2.5 所示。

```
建立空双链表L
1～6创建L
L[长度=6]: 1 2 3 4 5 6
插入6～10
L[长度=11]: 1 2 3 4 5 6 6 7 8 9 10
序号为2的元素=3
设置序号为2的元素为20
L[长度=11]: 1 2 20 4 5 6 6 7 8 9 10
第一个值为6的元素序号=5
删除首元素
L[长度=10]: 2 20 4 5 6 6 7 8 9 10
删除首元素
L[长度=9]: 20 4 5 6 6 7 8 9 10
删除首元素
L[长度=8]: 4 5 6 6 7 8 9 10
删除首元素
L[长度=7]: 5 6 6 7 8 9 10
删除首元素
L[长度=6]: 6 6 7 8 9 10
删除首元素
L[长度=5]: 6 7 8 9 10
删除首元素
L[长度=4]: 7 8 9 10
删除首元素
L[长度=3]: 8 9 10
删除首元素
L[长度=2]: 9 10
```

图 2.5　第 2 章基础实验题 3 的执行结果

4. 解：循环单链表的基本运算算法的设计原理参见《教程》中的 2.3.6 节。包含循环单链表基本运算算法类 CLinkList 以及测试主程序的 Exp1-4.py 文件如下：

```
class LinkNode:                          #循环单链表结点类
    def __init__(self,data=None):        #构造函数
```

```
        self.data=data                          #data 属性
        self.next=None                          #next 属性

class CLinkList:                                #循环单链表类
    def __init__(self):                         #构造函数
        self.head=LinkNode()                    #头结点 head
        self.head.next=self.head                #构成循环

    def CreateListF(self,a):                    #头插法：由数组 a 整体建立循环单链表
        for i in range(0,len(a)):               #循环建立数据结点 s
            s=LinkNode(a[i])                    #新建存放 a[i]元素的结点 s
            s.next=self.head.next               #将 s 结点插入开始结点之前、头结点之后
            self.head.next=s

    def CreateListR(self,a):                    #尾插法：由数组 a 整体建立循环单链表
        t=self.head                             #t 始终指向尾结点,开始时指向头结点
        for i in range(0,len(a)):               #循环建立数据结点 s
            s=LinkNode(a[i]);                   #新建存放 a[i]元素的结点 s
            t.next=s                            #将 s 结点插入 t 结点之后
            t=s
        t.next=self.head                        #将尾结点的 next 改为指向头结点

    def geti(self,i):                           #返回序号为 i 的结点
        p=self.head                             #首先 p 指向头结点
        j=-1                                    #头结点的编号看成-1
        while (j<i):
            j+=1
            p=p.next
            if p==self.head: break
        return p

    def Add(self,e):                            #在线性表的末尾添加一个元素 e
        s=LinkNode(e)                           #新建结点 s
        p=self.head
        while p.next!=self.head:                #查找尾结点 p
            p=p.next
        p.next=s;                               #在尾结点之后插入结点 s
        s.next=self.head

    def getsize(self):                          #返回长度
        p=self.head
        cnt=0
        while p.next!=self.head:                #找到尾结点为止
            cnt+=1
            p=p.next
        return cnt

    def __getitem__(self,i):                    #求序号为 i 的元素
        assert i>=0                             #检测参数 i 正确性的断言
        p=self.geti(i)
        assert p!=self.head                     #p 不为头结点的检测
        return p.data

    def __setitem__(self,i,x):                  #设置序号为 i 的元素
```

```
        assert i>=0                                  #检测参数 i 正确性的断言
        p=self.geti(i)
        assert p!=self.head                          #p 不为头结点的检测
        p.data=x

    def GetNo(self,e):                               #查找第一个为 e 的元素的序号
        j=0
        p=self.head.next                             #首先 p 指向首结点
        while p!=self.head and p.data!=e:
            j+=1                                     #查找元素 e
            p=p.next
        if p==self.head:
            return -1                                #未找到时返回-1
        else:
            return j                                 #找到后返回其序号

    def Insert(self,i,e):                            #在线性表中序号为 i 的位置插入元素 e
        assert i>=0                                  #检测参数 i 正确性的断言
        s=LinkNode(e)                                #建立新结点 s
        if (i==0):                                   #插入作为首结点
            s.next=self.head.next
            self.head.next=s
        else:
            p=self.geti(i-1)                         #找到序号为 i-1 的结点 p
            assert p!=self.head                      #p 不为头结点的检测
            s.next=p.next                            #在 p 结点后面插入 s 结点
            p.next=s

    def Delete(self,i):                              #在线性表中删除序号为 i 的位置的元素
        assert i>=0                                  #检测参数 i 正确性的断言
        p=self.geti(i-1)                             #找到序号为 i-1 的结点 p
        assert p.next!=self.head                     #p.next 不为头结点的检测
        p.next=p.next.next;                          #删除 p 结点的后继结点

    def display(self):                               #输出线性表
        p=self.head.next                             #首先 p 指向首结点
        while p!=self.head:
            print(p.data,end=' ')
            p=p.next
        print()

if __name__ == '__main__':
    L=CLinkList()
    print()
    print("  建立循环单链表 L")
    a=[1,2,3,4,5,6]
    print("  1～6 创建 L")
    L.CreateListR(a)
    print("  L[长度=%d]: " %(L.getsize()),end='');L.display()
    print("  插入 6～10")
    for i in range(6,11):
        L.Add(i)
    print("  L[长度=%d]: " %(L.getsize()),end='');L.display()
```

```
    print("  序号为 2 的元素＝%d" %(L[2]))
    print("  设置序号为 2 的元素为 20")
    L[2]＝20
    print("  L[长度＝%d]: " %(L.getsize()),end＝''),L.display()
    x＝6
    print("  第一个值为%d 的元素序号＝%d" %(x,L.GetNo(x)))
    n＝L.getsize()
    for i in range(n－2):
        print("  删除首元素")
        L.Delete(0)
        print("  L[长度＝%d]: " %(L.getsize()),end＝''),L.display()
```

上述程序的执行结果如图 2.6 所示。

```
建立循环单链表L
1～6创建L
L[长度=6]: 1 2 3 4 5 6
插入6～10
L[长度=11]: 1 2 3 4 5 6 6 7 8 9 10
序号为2的元素=3
设置序号为2的元素为20
L[长度=11]: 1 2 20 4 5 6 6 7 8 9 10
第一个值为6的元素序号=5
删除首元素
L[长度=10]: 2 20 4 5 6 6 7 8 9 10
删除首元素
L[长度=9]: 20 4 5 6 6 7 8 9 10
删除首元素
L[长度=8]: 4 5 6 6 7 8 9 10
删除首元素
L[长度=7]: 5 6 6 7 8 9 10
删除首元素
L[长度=6]: 6 6 7 8 9 10
删除首元素
L[长度=5]: 6 7 8 9 10
删除首元素
L[长度=4]: 7 8 9 10
删除首元素
L[长度=3]: 8 9 10
删除首元素
L[长度=2]: 9 10
```

图 2.6　第 2 章基础实验题 4 的执行结果

5. 解：循环双链表的基本运算算法的设计原理参见《教程》中的 2.3.6 节。包含循环双链表基本运算算法类 CDLinkList 以及测试主程序的 Exp1-5.py 文件如下：

```
class DLinkNode:                          #循环双链表结点类
    def __init__(self,data＝None):        #构造函数
        self.data＝data                   #data 属性
        self.next＝None                   #next 属性
        self.prior＝None                  #prior 属性

class CDLinkList:                         #循环双链表类
    def __init__(self):                   #构造函数
        self.dhead＝DLinkNode()           #头结点 dhead
        self.dhead.next＝self.dhead
        self.dhead.prior＝self.dhead

    def CreateListF(self,a):              #头插法：由数组 a 整体建立循环双链表
        for i in range(0,len(a)):         #循环建立数据结点 s
            s＝DLinkNode(a[i])            #新建存放 a[i]元素的结点 s,将其插入表头
            s.next＝self.dhead.next       #修改 s 结点的 next 成员
```

```
        self.dhead.next.prior=s          #修改头结点的后继结点的 prior
        self.dhead.next=s                #修改头结点的 next
        s.prior=self.dhead               #修改 s 结点的 prior

    def CreateListR(self,a):             #尾插法：由数组 a 整体建立循环双链表
        t=self.dhead                     #t 始终指向尾结点,开始时指向头结点
        for i in range(0,len(a)):        #循环建立数据结点 s
            s=DLinkNode(a[i])            #新建存放 a[i]元素的结点 s
            t.next=s                     #将 s 结点插入 t 结点之后
            s.prior=t
            t=s
        t.next=self.dhead                #将尾结点的 next 改为指向头结点
        self.dhead.prior=t               #将头结点的 prior 置为 t

    def geti(self,i):                    #返回序号为 i 的结点
        p=self.dhead                     #首先 p 指向头结点
        j=-1
        while (j<i):
            j+=1
            p=p.next
            if p==self.dhead: break
        return p

    def Add(self,e):                     #在线性表的末尾添加一个元素 e
        s=DLinkNode(e)                   #新建结点 s
        p=self.dhead
        while p.next!=self.dhead:        #查找尾结点 p
            p=p.next
        p.next=s;                        #在尾结点 p 之后插入结点 s
        s.prior=p
        s.next=self.dhead
        self.dhead.prior=s

    def getsize(self):                   #返回长度
        p=self.dhead
        cnt=0
        while p.next!=self.dhead:        #找到尾结点为止
            cnt+=1
            p=p.next
        return cnt

    def __getitem__(self,i):             #求序号为 i 的元素
        assert i>=0                      #检测参数 i 正确性的断言
        p=self.geti(i)
        assert p!=self.dhead             #p 不为头结点的检测
        return p.data

    def __setitem__(self,i,x):           #设置序号为 i 的元素
        assert i>=0                      #检测参数 i 正确性的断言
        p=self.geti(i)
        assert p!=self.dhead             #p 不为头结点的检测
        p.data=x
```

```
    def GetNo(self,e):                          #查找第一个为 e 的元素的序号
        j=0
        p=self.dhead.next                       #首先 p 指向首结点
        while p!=self.dhead and p.data!=e:
            j+=1                                #查找元素 e
            p=p.next
        if p==self.dhead:
            return -1                           #未找到时返回-1
        else:
            return j                            #找到后返回其序号

    def Insert(self,i,e):                       #在线性表中序号为 i 的位置插入元素 e
        assert i>=0                             #检测参数 i 正确性的断言
        s=DLinkNode(e)                          #建立新结点 s
        if (i==0):                              #插入作为首结点
            p=self.dhead
        else:
            p=self.geti(i-1)                    #找到序号为 i-1 的结点 p
            assert p!=self.dhead                #p 不为头结点的检测
        s.next=p.next                           #在 p 结点之后插入 s 结点
        p.next.prior=s
        p.next=s
        s.prior=p

    def Delete(self,i):                         #在线性表中删除序号为 i 的位置的元素
        assert i>=0                             #检测参数 i 正确性的断言
        p=self.geti(i-1)                        #找到序号为 i-1 的结点 p
        assert p.next!=self.head                #p.next 不为头结点的检测
        p.next=p.next.next;                     #删除 p 结点的后继结点

    def Delete(self,i):                         #在线性表中删除序号为 i 的位置的元素
        assert i>=0                             #检测参数 i 正确性的断言
        p=self.geti(i)                          #找到序号为 i 的结点 p
        assert p!=self.dhead                    #p 不为头结点的检测
        p.prior.next=p.next                     #删除 p 结点
        p.next.prior=p.prior

    def display(self):                          #输出线性表
        p=self.dhead.next
        while p!=self.dhead:
            print(p.data,end=' ')
            p=p.next;
        print()

if __name__ == '__main__':
    L=CDLinkList()
    print()
    print("  建立循环双链表 L")
    a=[1,2,3,4,5,6]
    print("  1～6 创建 L")
    L.CreateListR(a)
    print("  L[长度=%d]: " %(L.getsize()),end=''),L.display()
    print("  插入 6～10")
```

```
for i in range(6,11):
    L.Add(i)
print("  L[长度=%d]: " %(L.getsize()),end='',L.display()
print("  序号为 2 的元素=%d" %(L[2]))
print("  设置序号为 2 的元素为 20")
L[2]=20
print("  L[长度=%d]: " %(L.getsize()),end='',L.display()
x=6
print("  第一个值为%d 的元素序号=%d" %(x,L.GetNo(x)))
n=L.getsize()
for i in range(n-2):
    print("  删除首元素")
    L.Delete(0)
    print("  L[长度=%d]: " %(L.getsize()),end='',L.display()
```

上述程序的执行结果如图 2.7 所示。

```
建立循环双链表L
1～6创建L
L[长度=6]: 1 2 3 4 5 6
插入6～10
L[长度=11]: 1 2 3 4 5 6 6 7 8 9 10
序号为2的元素=3
设置序号为2的元素为20
L[长度=11]: 1 2 20 4 5 6 6 7 8 9 10
第一个值为6的元素序号=5
删除首元素
L[长度=10]: 2 20 4 5 6 6 7 8 9 10
删除首元素
L[长度=9]: 20 4 5 6 6 7 8 9 10
删除首元素
L[长度=8]: 4 5 6 6 7 8 9 10
删除首元素
L[长度=7]: 5 6 6 7 8 9 10
删除首元素
L[长度=6]: 6 6 7 8 9 10
删除首元素
L[长度=5]: 6 7 8 9 10
删除首元素
L[长度=4]: 7 8 9 10
删除首元素
L[长度=3]: 8 9 10
删除首元素
L[长度=2]: 9 10
```

图 2.7　第 2 章基础实验题 5 的执行结果

2.2.3　应用实验题

1. 编写一个简单的学生成绩管理程序，每个学生记录包含学号、姓名、课程和分数成员，采用顺序表存储，完成以下功能：

① 屏幕显示所有学生记录。

② 输入一个学生记录。

③ 按学号和课程删除一个学生记录。

④ 按学号排序并输出所有学生记录。

⑤ 按课程排序，对于一门课程，学生按分数递减排序。

2. 编写一个实验程序实现以下功能：

① 从文本文件 xyz.in 中读取 3 行整数，每行的整数递增排列，两个整数之间用一个空格分隔，全部整数的个数为 n，这 n 个整数均不相同。