【K03】展示后缀表达式计算过程的栈变化

```
#uuid_share#  4e746bfe-8d2d-42ca-af59-cb654a86adee  #
# DSA'21 课程上机作业
# 【H2】栈与队列编程作业
#
# 说明：为方便批改作业，请同学们在完成作业时注意并遵守下面规则：
# （1）直接在本文件中的*函数体内*编写代码，每个题目的函数后有调用语句用于检验
# （2）如果作业中对相关类有明确命名/参数/返回值要求的，请严格按照要求执行
# （3）有些习题会对代码的编写进行特殊限制，请注意这些限制并遵守
# （4）作业在 4 月 14 日 18:00 之前提交到 Canvas 系统
#

# ======= 1 中缀表达式求值 =======
# 通过把“中缀转后缀”和“后缀求值”两个算法功能集成在一起（非简单的顺序调用），
# 实现对中缀表达式直接求值，新算法还是从左到右扫描中缀表达式，
# 但同时使用两个栈，一个暂存操作符，一个暂存操作数，来进行求值。
#
# 创建一个函数，接受参数为一个字符串，即一个中缀表达式，
# 其中每个数字或符号间由一个空格隔开；
# 返回一个浮点数，即求值的结果。（支持 + - * / ^ 五种运算）
#   其中“ / ”定义为真除 True DIV，结果是浮点数类型
# 输入样例 1：
# ( 2 + 3 ) * 6 + 4 / 2
# 输出样例 1：
# 32.0
# 输入样例 2：
# 2 ^ 3 + 4 * 5 - 16 / 2
# 输出样例 2：
# 20.0
# 输入样例 3：
# ( 5 + 1 ) * 2 / 3 - 3 ^ ( 2 + 8 / 4 ) / 9 + 6
# 输出样例 3：
# 1.0


class Stack:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []
    def push(self, item):
        self.items.append(item)

    def pop(self):
        return self.items.pop()

    def peek(self):
        return self.items[len(self.items) - 1]

    def size(self):
        return len(self.items)

def calculate(s) -> float:
    #  请在此编写你的代码（可删除 pass 语句）
    prec = {}
    prec['^'] = 4
    prec['*'] = 3
    prec['/'] = 3
    prec['+'] = 2
    prec['-'] = 2
    prec['('] = 1
    numlist = [str(i) for i in range(1000000)]
    opStack = Stack()
    numStack = Stack()
    tokenList = s.split(' ')
    for token in tokenList:
        if token in numlist:
            numStack.push(token)
        elif token == '(':
            opStack.push(token)
        elif token == ')':
            topToken = opStack.pop()
            while topToken != '(':
                numTocal2 = numStack.pop()
                numTocal1 = numStack.pop()
                numStack.push(doMath(topToken,numTocal1,numTocal2))
                topToken = opStack.pop()
        elif token in '+-*/^':
            while (not opStack.isEmpty()) and (prec[opStack.peek()] >= prec[token]):
                topToken = opStack.pop()
                numTocal2 = numStack.pop()
                numTocal1 = numStack.pop()
                numStack.push(doMath(topToken, numTocal1, numTocal2))
            opStack.push(token)
    while not opStack.isEmpty():
        topToken = opStack.pop()
        numTocal2 = numStack.pop()
        numTocal1 = numStack.pop()
        doMath(topToken, numTocal1, numTocal2)
        numStack.push(doMath(topToken, numTocal1, numTocal2))
    return float(numStack.pop())
def doMath(op,op1,op2):
    if op == '*':
        return int(op1) * int(op2)
    elif op == '/':
        return int(op1) / int(op2)
    elif op == '+':
        return int(op1) + int(op2)
    elif op == '^':
        result = 1
        for i in range(int(op2)):
            result = result * int(op1)
        return result
    else:
        return int(op1) - int(op2)
    # 代码结束

# 调用检验
print("======== 1-calculate ========")
print(calculate("( 2 + 3 ) * 6 + 4 / 2"))
print(calculate("2 ^ 3 + 4 * 5 - 16 / 2"))
print(calculate("( 5 + 1 ) * 2 / 3 - 3 ^ ( 2 + 8 / 4 ) / 9 + 6"))

# ======= 2 基数排序 =======
# 实现一个基数排序算法，用于 10 进制的正整数从小到大的排序。
#
# 思路是保持 10 个队列(队列 0、队列 1......队列 9、队列 main)，开始，所有的数都在 main 队列，没有排序。
# 第一趟将所有的数根据其 10 进制个位(0~9)，放入相应的队列 0~9，全放好后，按照 FIFO 的顺序，将每个队列的数合并排到 main 队列。
# 第二趟再从 main 队列队首取数，根据其十位的数值，放入相应队列 0~9，全放好后，仍然按照 FIFO 的顺序，将每个队列的数合并排到 main 队列。
# 第三趟放百位，再合并；第四趟放千位，再合并。
# 直到最多的位数放完，合并完，这样 main 队列里就是排好序的数列了。
#
# 创建一个函数，接受参数为一个列表，为需要排序的一系列正整数，
# 返回排序后的数字列表。
# 输入样例 1：
# [1, 2, 4, 3, 5]
# 输出样例 1：
# [1, 2, 3, 4, 5]
# 输入样例 2：
# [8, 91, 34, 22, 65, 30, 4, 55, 18]
# 输出样例 2：
# [4, 8, 18, 22, 30, 34, 55, 65, 91]


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)


def radix_sort(lstTosort) -> list:
    # 请在此编写你的代码（可删除 pass 语句）
    Q0 = Queue()
    Q1 = Queue()
    Q2 = Queue()
    Q3 = Queue()
    Q4 = Queue()
    Q5 = Queue()
    Q6 = Queue()
    Q7 = Queue()
    Q8 = Queue()
    Q9 = Queue()
    Qmain = Queue()
    destination = len(str(max(lstTosort)))
    for i in lstTosort:
        Qmain.enqueue(str(i))
    for j in range(-1,-destination-1,-1):
        ct = 0
```

```
#拿出来,j 是代表位数
    while not Qmain.isEmpty() and ct < len(lstTosort):
        element = Qmain.dequeue()
        ct += 1
        if len(element) >= -j:
            if element[j] == '0':
                Q0.enqueue(element)
            elif element[j] == '1':
                Q1.enqueue(element)
            elif element[j] == '2':
                Q2.enqueue(element)
            elif element[j] == '3':
                Q3.enqueue(element)
            elif element[j] == '4':
                Q4.enqueue(element)
            elif element[j] == '5':
                Q5.enqueue(element)
            elif element[j] == '6':
                Q6.enqueue(element)
            elif element[j] == '7':
                Q7.enqueue(element)
            elif element[j] == '8':
                Q8.enqueue(element)
            elif element[j] == '9':
                Q9.enqueue(element)
        else:
            Qmain.enqueue(element) #当位数不够的时候,较小,放回去
    #放回去
    while not Q0.isEmpty():
        Qmain.enqueue(Q0.dequeue())
    while not Q1.isEmpty():
        Qmain.enqueue(Q1.dequeue())
    while not Q2.isEmpty():
        Qmain.enqueue(Q2.dequeue())
    while not Q3.isEmpty():
        Qmain.enqueue(Q3.dequeue())
    while not Q4.isEmpty():
        Qmain.enqueue(Q4.dequeue())
    while not Q5.isEmpty():
        Qmain.enqueue(Q5.dequeue())
    while not Q6.isEmpty():
        Qmain.enqueue(Q6.dequeue())
    while not Q7.isEmpty():
        Qmain.enqueue(Q7.dequeue())
        while not Q8.isEmpty():
            Qmain.enqueue(Q8.dequeue())
        while not Q9.isEmpty():
            Qmain.enqueue(Q9.dequeue())
    result = []
    while not Qmain.isEmpty():
        result.append(Qmain.dequeue())
    result = [int(x) for x in result]
    return result
# 代码结束

# 调用检验
print("======== 2-radix_sort ========")
print(radix_sort([1, 2, 4, 3, 5]))
print(radix_sort([8, 91, 34, 22, 65, 30, 4, 55, 18]))

# ======= 3 HTML 标记匹配 =======
# 实现扩展括号匹配算法，用来检查 HTML 文档的标记是否匹配。
# HTML 标记应该成对、嵌套出现，
# 开标记是<tag>这种形式，闭标记是</tag>这种形式。
#
# 创建一个函数，接受参数为一个字符串，为一个 HTML 文档中的内容，
# 返回 True 或 False，表示该字符串中的标记是否匹配。
# 输入样例 1：
# <html> <head> <title> Example </title> </head> <body> <h1>Hello, world</h1>
</body> </html>
# 输出样例 1：
# True
# 输入样例 2：
# <html> <head> <title> Test </title> </head> <body> <p>It's just a test.</p> <p>And
this is for False.<p> </body> </html>
# 输出样例 2：
# False


def HTMLMatch(string) -> bool:
    # 请在此编写你的代码（可删除 pass 语句）
    string = string.split(' ')
    string = ''.join(x for x in string)
    #print(string)
    strStack = Stack()
    checkTheStack = Stack()
    #首先把他们合成 基本元素
    #ct 是一个计数器
    global ct
    ct = 0
    while ct <= len(string)-1:
        if string[ct] == '<':
            element = ''
            tag = True
            while tag and ct < len(string):
                if string[ct] != '>':
                    element += string[ct]
                    #print(element)
                    ct += 1
                else:
                    element += string[ct]
                    ct +=1
                    strStack.push(element)
                    tag = False
              #print(strStack.content())
        else:
            ct += 1
    while not strStack.isEmpty():
        element_check = strStack.peek()
        if element_check[1] == '/':
            element_check = element_check[0] + element_check[2:]
            #print(element_check)
            checkTheStack.push(element_check)
            #print(checkTheStack.content())
            strStack.pop()
        else:
            #print(checkTheStack.content())
            #print(strStack.content())
            if not checkTheStack.isEmpty():
                if checkTheStack.pop() == strStack.pop():
                    pass
            else:
                return False
    if checkTheStack.isEmpty():
        return True
    else:
        return False
    #  代码结束


#  调用检验
print("======== 3-HTMLMatch ========")
print(
    HTMLMatch(
        "<html> <head> <title>Example</title> </head> <body> <h1>Hello, world</h1>
</body> </html>"
    ))
print(
    HTMLMatch(
        "<html> <head> <title> Test </title> </head> <body> <p>It's just a test.</p>
<p>And this is for False.<p> </body> </html>"
    ))


class Node():
    def __init__(self, initdata=None):
        self.data = initdata
        self.next = None
        self.prev = None

    def getData(self):
        return self.data

    def getNext(self):
        return self.next

    def getPrev(self):
        return self.prev

    def setData(self, newdata):
        self.data = newdata

    def setNext(self, newnext):
        self.next = newnext

    def setPrev(self, newprev):
        self.prev = newprev


# ======== 4  链表实现栈和队列  ========
#  用链表实现 ADT Stack 与 ADT Queue 的所有接口
class LinkStack(object):
    #  请在此编写你的代码（可删除 pass 语句）
    class Node(object):
        def __init__(self, data=None,prev=None):
            self.data = data
```

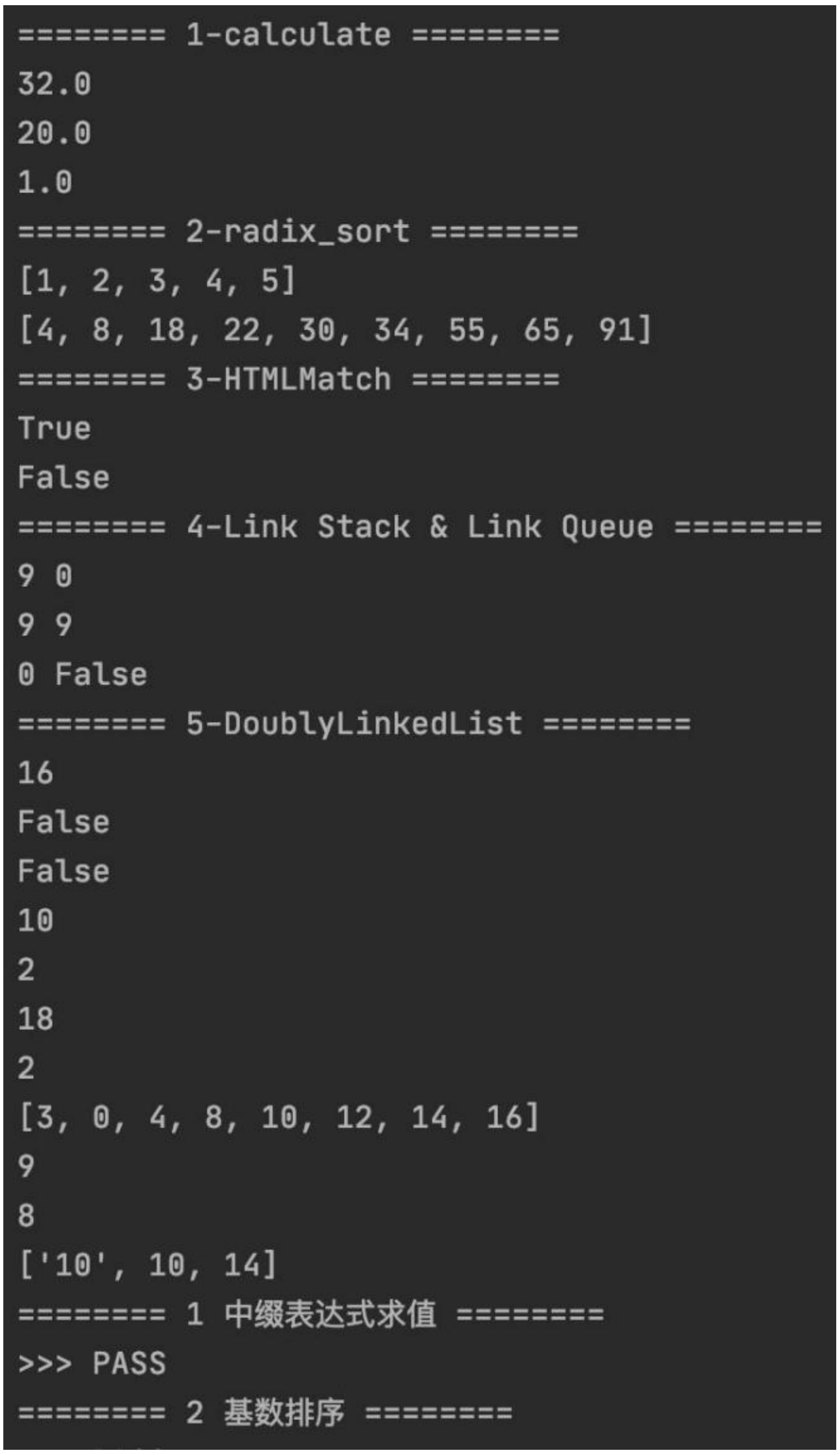
```
                self.prev = prev
        def __init__(self):
            self.top = None
            self.len = 0
        def isEmpty(self):
            return self.len == 0
        def size(self):
            return self.len
            # 获取栈顶元素
        def peek(self):
            return self.top.data
            # 压栈
        def push(self, data):
            newnode = self.Node(data)
            lastTop = self.top
            self.top = newnode
            newnode.prev = lastTop
            self.len += 1
        def pop(self):
            lastData = self.top.data
            nowNode = self.top.prev
            self.top.prev = self.top
            self.top = nowNode
            self.len -= 1
            return lastData
        # 代码结束
class LinkQueue(object):
        # 请在此编写你的代码（可删除 pass 语句）
        class Node(object):
            def __init__(self, data=None,next=None):
                self.data = data
                self.next = next
        def __init__(self):
            self.first = None
            self.last = None
            self.len = 0
        def isEmpty(self):
            return self.len == 0
        def enqueue(self,data):
            newnode = self.Node(data)
            if self.isEmpty():
                self.last = newnode
                self.first = self.last
                self.len += 1
            else:
                self.last.next = newnode
                self.last = newnode
                self.len += 1
        def dequeue(self):
            dataToout = self.first.data
            self.first = self.first.next
            self.len -= 1
            return dataToout
        def size(self):
            return self.len
        # 代码结束

# 检验
print("======== 4-Link Stack & Link Queue ========")
s = LinkStack()
q = LinkQueue()
for i in range(10):
        s.push(i)
        q.enqueue(i)
print(s.peek(), q.dequeue())    # 9 0
print(s.pop(), q.size())    # 9 9
while not s.isEmpty():
        s.pop()
print(s.size(), q.isEmpty())    # 0 False


# ======== 5 双链无序表 ========
# 实现双向链表版本的 UnorderedList，接口同 ADT UnorderedList
# 包含如下方法：isEmpty, add, search, size, remove, append, index, pop, insert, __len__,
__getitem__
# 用于列表字符串表示的__str__方法 (注：__str__里不要使用 str(), 用 repr()代替
# 用于切片的__getitem__方法
# 在节点 Node 中增加 prev 变量，引用前一个节点
# 在 UnorderedList 中增加 tail 变量与 getTail 方法，引用列表中最后一个节点
# 选做：DoublyLinkedList(iterable) -> new DoublyLinkedList initialized from iterable's
items
class DoublyLinkedList(object):
        # 请在此编写你的代码（可删除 pass 语句）
        class Node(object):
            def __init__(self,data = None):
                self.data = data
                self.next = None
                self.prev = None
            def getData(self):
                return self.data
            def getNext(self):
                return self.next
            def getPrev(self):
                return self.prev
            def setData(self, newdata):
                self.data = newdata
            def setNext(self, newnext):
                self.next = newnext
            def setPrev(self, newprev):
                self.prev = newprev
        def __init__(self,data=None):
            self.head = None
            self.tail = None
            self.length = 0
            self.itercount=0
            self.iterNode = None
            if data != None:
                for i in data:
                    self.append(i)
        def isEmpty(self):
            return self.head == None
        def search(self, item):
            tag = self.head
            while tag != None:
                if tag.data == item:
                    return True
                else:
                    tag = tag.next
            return False

        def copy(self):
            dlst = DoublyLinkedList()
            current_Node = self.head
            while current_Node != None:
                dlst.append(current_Node.getData())
                current_Node = current_Node.getNext()
            return dlst

        def __iter__(self):
            return self.copy()

        def __next__(self):
            if self.itercount == 0:
                self.iterNode = self.head
            if self.iterNode == None:
                self.itercount = 0
                self.iterNode = None
                raise StopIteration()
            else:
                x = self.iterNode.getData()
                self.iterNode = self.iterNode.getNext()
                self.itercount += 1
                return x

        def size(self):
            tag = self.head
            ct = 0
            while tag != None:
                ct += 1
                tag = tag.next
            return ct

        def remove(self,item):
            tag = self.head
            found = False
            while not found:
                if tag == None:
                    raise ValueError("%s not found"%item)
                elif tag.getData() != item:
                    tag = tag.getNext()
                else:
                    found=True
                    prev = tag.getPrev()
                    next = tag.getNext()
                    if prev != None:
                        prev.setNext(next)
                    else:
                        self.head = next
                    if next != None:
                        next.setPrev(prev)
                    else:
                        self.tail = prev
            self.length -= 1

        def append(self, item):
            newnode = self.Node(item)
            if self.isEmpty():
                self.head = newnode
                self.tail = newnode
                self.length += 1
            else:
                tag = self.head
                while tag.next != None:
                    tag = tag.next
                tag.next = newnode
                newnode.prev = tag
                self.tail = newnode
                self.length += 1

        def add(self,item):
            tag = self.Node(item)
            tag.setNext(self.head)
            if self.head != None:
                self.head.setPrev(tag)
            else:
                self.tail = tag
            self.head = tag
            self.length += 1

        def __eq__(self,input):
            equal = True
            if type(input) != DoublyLinkedList or self.length != input.length:
                equal = False
            else:
                tag1 = self.head
                tag2 = input.head
                while equal and tag1 != None:
                    if tag1.getData() == tag2.getData():
                        tag1 = tag1.getNext()
                        tag2 = tag2.getNext()
                    else:
                        equal = False
            return equal

        def index(self,item):
            if self.isEmpty():
                return item +'is not in list'
            else:
                tag = self.head
                ct = 0
                while tag.next != None:
                    if tag.data == item:
                        return ct
                    else:
                        ct += 1
                        tag = tag.next
                item = str(item)+' '
                return item + 'is not in list'

        def pop(self,position = None):
            if position == None:
                if self.tail==None:
                    raise IndexError("pop from empty DoublyLinkedList")
                output = self.tail.getData()
                self.tail=self.tail.getPrev()
                if self.tail != None:
                    self.tail.setNext(None)
                else:
                    self.head = None
            else:
                currentNode = self.head
                currentid = 0
                while currentid<position:
                    currentNode = currentNode.getNext()
                    currentid += 1
                    if currentNode == None:
                        raise IndexError("pop index out of range")
                output = currentNode.getData()
                prev = currentNode.getPrev()
                next = currentNode.getNext()
                if prev != None:
                    prev.setNext(next)
                else:
                    self.head = next
                if next != None:
                    next.setPrev(prev)
                else:
                    self.tail = prev
            self.length -= 1
            return output

        def insert(self, id, item):
            tag = self.head
            ct = 0
```

```
        newNode = self.Node(item)
        while ct < id:
            tag = tag.getNext()
            ct += 1
            if tag == None:
                newNode.setPrev(self.tail)
                self.tail.setNext(newNode)
                self.tail = newNode
                break
        else:
            prevNode = tag.getPrev()
            newNode.setNext(tag)
            newNode.setPrev(prevNode)
            if prevNode != None:
                prevNode.setNext(newNode)
            else:
                self.head = newNode
        self.length += 1
    def __len__(self):
        tag = self.head
        ct = 0
        while tag:
            tag = tag.next
            ct += 1
        return ct

    def __str__(self):
        tag = self.head
        ct = 0
        resultToshow = []
        while ct < self.__len__():
            resultToshow.append(tag.data)
            tag = tag.next
            ct += 1
        return repr(resultToshow)
    def getTail(self):
        return self.tail

    def __getitem__(self,key):
        if isinstance(key,int):
            if key >= 0:
                currentNode = self.head
                currentid = 0
                while currentid < key:
                    currentNode = currentNode.getNext()
                    currentid += 1
                    if currentNode == None:
                        raise IndexError("DoublyLinkedList index out of range")
                output = currentNode.getData()
            else:
                currentNode = self.tail
                currentid=-1
                while currentid > key:
                    currentNode = currentNode.getPrev()
                    currentid -= 1
                    if currentNode == None:
                        raise IndexError("DoublyLinkedList index out of range")
                output = currentNode.getData()
            return output
        if isinstance(key,slice):
            newslice = DoublyLinkedList()
            start,stop,step=key.start,key.stop,key.step
            if step == None:
                step = 1
            elif step == 0:
                raise ValueError("slice step cannot be zero")
            if start == None:
                if step > 0:
                    start = 0
                else:
                    start = self.length-1
            elif start < 0:
                start += self.length
            if stop == None:
                if step > 0:
                    stop = self.length
                else:
                    stop =- 1
            elif stop < 0:
                stop += self.length
            if step > 0:
                if stop > self.length:
                    stop = self.length
                if start < 0:
                    start = 0
                if start < stop:
                    currentid = 0
                    currentNode = self.head
                    while currentid < start:
                        currentNode = currentNode.getNext()
                        currentid += 1
                    newslice.append(currentNode.getData())
                    while currentid < stop-step:
                        for i in range(step):
                            currentNode = currentNode.getNext()
                            currentid += 1
                        newslice.append(currentNode.getData())
            if step < 0:
                if start > self.length:
                    start = self.length
                if stop < 0:
                    stop = -1
                if start > stop:
                    currentid = 0
                    currentNode = self.head
                    while currentid < start:
                        currentNode = currentNode.getNext()
                        currentid += 1
                    newslice.append(currentNode.getData())
                    while currentid > stop-step:
                        for i in range(-step):
                            currentNode = currentNode.getPrev()
                            currentid -= 1
                        newslice.append(currentNode.getData())
            return newslice
        else:
            raise TypeError("invalid argument type")
    # 代码结束

print("======== 5-DoublyLinkedList ========")
mylist = DoublyLinkedList()
for i in range(0, 20, 2):
    mylist.append(i)
mylist.add(3)
mylist.remove(6)
print(mylist.getTail().getPrev().getData())   # 16
print(mylist.isEmpty())   # False
print(mylist.search(5))   # False
print(mylist.size())   # 10
print(mylist.index(2))   # 2
print(mylist.pop())   # 18
print(mylist.pop(2))   # 2
print(mylist)   # [3, 0, 4, 8, 10, 12, 14, 16]
mylist.insert(3, "10")
print(len(mylist))   # 9
print(mylist[4])   # 8
print(mylist[3:8:2])   # ['10', 10, 14]


#  代码结束
#  检验
print("======== 5-DoublyLinkedList ========")
mylist = DoublyLinkedList()
for i in range(0, 20, 2):
    mylist.append(i)
mylist.add(3)
mylist.remove(6)
print(mylist.getTail().getPrev().getData())   # 16
print(mylist.isEmpty())   # False
print(mylist.search(5))   # False
print(mylist.size())   # 10
print(mylist.index(2))   # 2
print(mylist.pop())   # 18
print(mylist.pop(2))   # 2
print(mylist)   # [3, 0, 4, 8, 10, 12, 14, 16]
mylist.insert(3, "10")
print(len(mylist))   # 9
print(mylist[4])   # 8
print(mylist[3:8:2])   # ['10', 10, 14]

#uuid_share#   dad110f3-7cd9-4957-ba45-9e2a5165a722   #
# DSA'21 H2  随机测试样例 P1

from random import randrange, shuffle, choice
from sys import stderr, stdout
print_bak = globals().get('print_bak', print)   #  使用另行备份的 print 函数

from collections import Counter


class Node:   #  检测规范调用接口
    invalid_key = Counter()

    def __init__(self, initdata=None):
        self.__data = initdata
        self.__next = None
```

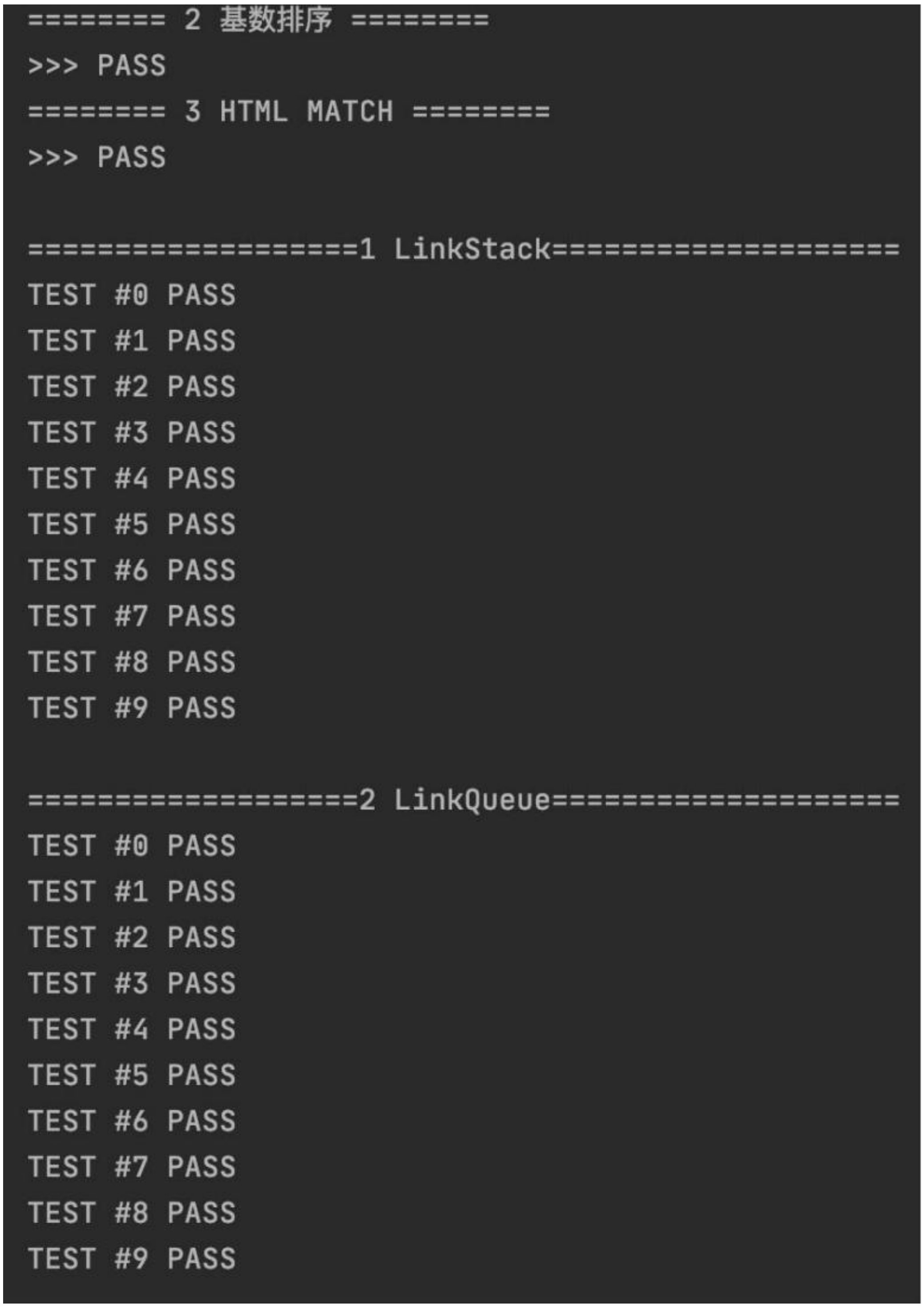
```
            self.__prev = None

        for k in ('data', 'next', 'prev'):
            exec(
                f'''@property
def {k}(self):
        self.invalid_key['get_'+'{k}']+=1
        return getattr(self,'get'+'{k.capitalize()}')()
@{k}.setter
def {k}(self,val):
        self.invalid_key['set_'+'{k}']+=1
        getattr(self,'set'+'{k.capitalize()}')(val)''', globals(), locals())

        def getData(self):
            return self.__data

        def getNext(self):
            return self.__next

        def getPrev(self):
            return self.__prev

        def setData(self, newdata):
            self.__data = newdata

        def setNext(self, newnext):
            self.__next = newnext

        def setPrev(self, newprev):
            self.__prev = newprev


# ======= 1  中缀表达式求值  =======
def gen_exp():
        pool = ['@']
        op_expand = randrange(7)
        op_group = randrange(4)
        ops = [''] * op_expand + ['()'] * op_group
        shuffle(ops)

        has_pow = False
        for op in ops:
            tmp = list(i for i, x in enumerate(pool) if x == '@')
            tmp = choice(tmp)
            if op:    # add brackets
                pool.insert(tmp + 1, ')')
                pool.insert(tmp, '(')
                tmp += 1
            # expand expr
            op = '+-*/^'[randrange(5)]
            if op == '^':
                if has_pow:
                    op = '+-*/'[randrange(4)]
                else:
                    has_pow = True
            pool.insert(tmp + 1, '@')
            pool.insert(tmp + 1, op)

        for i, x in enumerate(pool):
            if x == '@':
                pool[i] = str(randrange(12))

        expr = ' '.join(pool)

        try:   # 有效性检查
            res = eval(expr.replace('^', '**'))
            res1 = eval(expr.replace('/', '//').replace('^', '**'))
            ress = (res, res1)
            assert all(1e-8 < abs(x) < 1e8 and type(x).__name__ != 'complex'
                       and str(x) != 'nan' for x in ress)
            return expr, (res, res1)
        except:
            return gen_exp()


print_bak("======== 1  中缀表达式求值  ========")
try:
        cases = [
            ('114514', (114514, 114514)),
        ]
        for test in range(20):
            res1 = None
            if test < len(cases):
                expr, ress = cases[test]
            else:
                expr, ress = gen_exp()
            res1 = calculate(expr)
            if not any(
                        abs((r - res1) / r) < 1e-6 if r else r == res1 for r in ress):
                    raise AssertionError('参考答案: %s  或  %s' % ress)
        print_bak('>>> PASS')
except Exception as e:
        print_bak(f'''调用: calculate({repr(expr)})''', file=stderr)
        print_bak(f'输出: {repr(res1)}', file=stderr)
        print_bak(f'{type(e).__name__}: {e}', file=stderr)

# ======= 2  基数排序  =======
print_bak("======== 2  基数排序  ========")
try:
        cases = []
        for test in range(20):
            res1 = None
            if test < len(cases):
                lst = cases[test]
            else:
                lst = [randrange(1000) for i in range(randrange(10, 50))]
            res = sorted(lst)
            res1 = radix_sort(lst[:])
            assert res == res1, f'''参考答案: {res}'''
        print_bak('>>> PASS')
except Exception as e:
        print_bak(f'''调用: radix_sort({repr(lst).replace(' ','')})''', file=stderr)
        print_bak(f'输出: {repr(res1)}', file=stderr)
        print_bak(f'{type(e).__name__}: {e}', file=stderr)

# ======= 3 HTML =======


def gen_xml(make_invalid=False):
        pool = []
        n_pairs = randrange(1, 15)
        from string import ascii_letters, ascii_lowercase
        text_pool = ascii_letters + "0123546789" + ' ' * 50

        for i in range(n_pairs):
            tag = ''.join(choice(ascii_lowercase) for i in range(randrange(1, 5)))
            tmp = randrange(len(pool) + 1)
            pool.insert(tmp, f'</{tag}>')
            pool.insert(tmp, f'<{tag}>')

        if make_invalid:
            for i in range(3):
                if not pool:
                        break
                pool.pop(randrange(len(pool)))
        pool.append('</html>')

        res = ['<html>']
        for node in pool:
            res.append(''.join(choice(text_pool) for i in range(randrange(10))))
            res.append(node)

        return ''.join(res)


ref_match = lambda s: (lambda l: not any(
        map(lambda t: t[1] != l.pop()[1] if t[0] else l.append(t),
            __import__('re').findall('<(/?)(.*?)>', s))) and not l)([])

print_bak("======== 3 HTML MATCH ========")
try:
        cases = [
            ('<html></html>', True),
            ('<html>', False),
            ('</html>', False),
        ]
        for test in range(20):
            res1 = None
            if test < len(cases):
                expr, res = cases[test]
            else:
                expr = gen_xml(test % 2)
                res = ref_match(expr)
            res1 = HTMLMatch(expr)
            assert res == res1, f'''参考答案: {res}'''
        print_bak('>>> PASS')
except Exception as e:
        print_bak(f'''调用: HTMLMatch({repr(expr)})''', file=stderr)
        print_bak(f'输出: {repr(res1)}', file=stderr)
        print_bak(f'{type(e).__name__}: {e}', file=stderr)

# SESSDSA20 H3  随机测试样例 P2
LINE_WIDTH = 50
N_TESTS = 10
N_OPS = 20
```

```
from collections import deque
from random import randrange, choice
from sys import stderr

if 'ref ds':

    class ref_node:
        def __init__(self, lst, ind):
            self.lst = lst
            self.ind = ind

        def getData(self):
            return self.lst[self.ind]

        def getNext(self):
            return ref_node(self.lst, self.ind + 1)

        def getPrev(self):
            return ref_node(self.lst, self.ind - 1)

        def setData(self, newdata):
            self.lst[self.ind] = newdata

        def __eq__(self, other):
            try:
                return self.getData() == other.getData()
            except:
                return False

    ref_node.__str__ = lambda self: 'Node(%r)' % self.getData()
    Node.__str__ = Node.__repr__ = ref_node.__repr__ = ref_node.__str__

    class ref_list:
        isEmpty = lambda self: not self.lst
        add = lambda self, item: self.lst.insert(0, item)
        search = lambda self, item: item in self.lst
        size = __len__ = lambda self: len(self.lst)

        def __init__(self, ref_type, it=None):
            self.ref_type = ref_type
            self.lst = []
            if it:
                for i in it:
                    self.lst.append(i)
        def getTail(self):
            assert self.size() > 0
            return ref_node(self.lst, len(self) - 1)

        def __getitem__(self, arg):
            res = self.lst[arg]
            if isinstance(arg, slice):
                res = ref_list(self.ref_type, res)
            return res

        def __eq__(self, other):
            try:
                if len(self) != len(other):
                    return False
                tmp = [(self[i], other[i]) for i in range(len(self))]
                return all(i == j for i, j in tmp)
            except:
                return False

        __str__ = __repr__ = lambda self: f'{self.ref_type.__name__}({self.lst})'

    class ref_deque(deque):
        push = deque.append
        peek = lambda self: self[-1]
        enqueue = deque.append
        dequeue = deque.popleft
        isEmpty = lambda self: not bool(self)
        size = deque.__len__


def test(i, t_lst, r_lst, op_write, op_read):
    print_bak('TEST #%d' % i, end='')
    _SIZE = 0
    passed = True
    ops = []
    params = []

    def get(param):
        if param == 'num':
            return randrange(N_OPS)
        elif param == 'numstr':
            if randrange(2):
                return randrange(N_OPS)
            return str(randrange(10))
        elif param == 'len':
            return randrange(_SIZE)
        elif param == '-len':
            return randrange(_SIZE) - _SIZE
        elif param == 'slice':
            a = randrange(_SIZE - 1)
            b = randrange(a, _SIZE + 1)
            return slice(a, b, randrange(1, 10))

def one_check(op):
    ref_exec = True
    op = op.split()

    try:
        params.clear()
        params.extend(map(get, op[1:]))
        r_ref = getattr(r_lst, op[0])(*params)
    except:
        ref_exec = False

    if ref_exec:
        r_test = getattr(t_lst, op[0])(*params)
        if r_ref != None:
            assert r_ref == r_test, '输出: %r;\n 应该输出: %r' % (r_test, r_ref)
            if isinstance(r_ref, ref_list):
                assert type(
                    r_test) == r_ref.ref_type, '输出类型错误: %s;\n 应为: %s' % (
                        type(r_test).__name__, r_ref.ref_type.__name__)

        ops.append((op[0], *params))

def output(op):
    func = op[0]
    params = ','.join(map(repr, op[1:]))
    return '%s(%s)' % (func, params)

try:
    for i in range(N_OPS):
        # write one
        curr_op = choice(op_write)
        one_check(curr_op)

        # update size
```

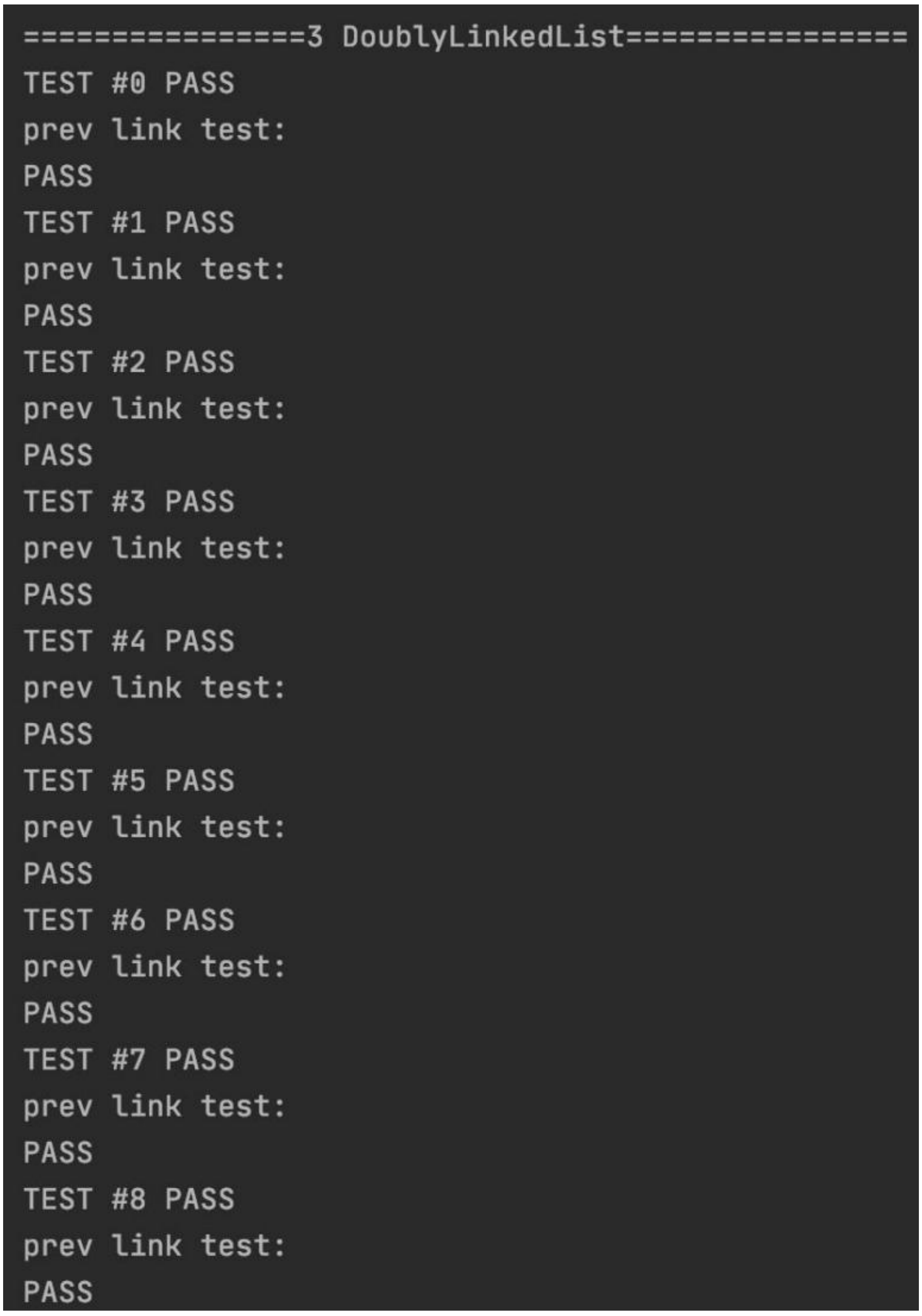
```
            _SIZE = len(r_lst)

            # read one
            curr_op = choice(op_read)
            one_check(curr_op)

        print_bak(' PASS')
    except Exception as e:
        print_bak('\n 出错的操作:', output((curr_op.split()[0], *params)))
        print_bak('历史操作:', ','.join(map(output, ops)))
        print_bak('报错: (%s: %s)' % (type(e).__name__, str(e)), file=stderr)
        try:
            print_bak('LAST LISTS'.center(LINE_WIDTH, '.'))
            print_bak('参考列表:', r_lst)
            print_bak('测试列表:', t_lst)
        except Exception as e:
            print_bak('打印报错 (%s: %s)' % (type(e).__name__, str(e)),
                      file=stderr)
        print_bak('END'.center(LINE_WIDTH, '.'))


def test_code(title, code):
    print_bak(title, end=':\n')
    # print_bak('Code'.center(LINE_WIDTH, '.'))
    # print_bak(code)
    try:
        exec(code, globals())
    except Exception as e:
        print_bak('报错 (%s: %s)' % (type(e).__name__, str(e)), file=stderr)


def prev_iter(lst):
    node = lst.getTail()
    res = []
    for i in range(len(lst)):
        res.append(node.getData())
        node = node.getPrev()
    return res


def safe_iter(lst):
    lst_iter = iter(lst)
    try:
        for i in range(len(lst)):
            yield next(lst_iter)
    except Exception as e:
        yield '报错 (%s: %s)' % (type(e).__name__, str(e))
    try:
        not_end = next(lst_iter)
        yield 'NOT END'
    except:
        pass


# push pop peek
print_bak('\n' + "1 LinkStack".center(LINE_WIDTH, '='))
for i in range(N_TESTS):
    test(i, LinkStack(), ref_deque(), (
        'push num',
        'pop',
    ), (
        'isEmpty',
        'peek',
        'size',
    ))

# enqueue dequeue
print_bak('\n' + "2 LinkQueue".center(LINE_WIDTH, '='))
for i in range(N_TESTS):
    test(i, LinkQueue(), ref_deque(), (
        'enqueue num',
        'dequeue',
    ), (
        'isEmpty',
        'size',
    ))

# getTail
print_bak('\n' + "3 DoublyLinkedList".center(LINE_WIDTH, '='))
for i in range(N_TESTS):
    l1 = DoublyLinkedList()
    l2 = ref_list(DoublyLinkedList)
    test(i, l1, l2, (
        'append numstr',
        'add numstr',
        'insert len numstr',
        'pop len',
        'pop',
        'remove numstr',
    ), (
        'isEmpty',
        'search numstr',
        'size',
        '__len__',
        'index numstr',
        '__getitem__ len',
        '__getitem__ slice',
        'getTail',
    ))
    test_code(
        'prev link test', r'''r1=prev_iter(l1)
r2=l2[::-1]
if r1==r2:
    print_bak('PASS')
else:
    print_bak('双链表倒序结果: ',r1,file=stderr)
    print_bak('参考结果: ',r2,file=stderr)''')

comment = '''
注：prev link test 用于测试双链表反向连接情况
以上为必做内容，以下为选做内容
'''
try:
    from browser import document
    target = document['py_stdout']
    target.innerHTML += f'<span style="color:blue">{comment}</span>'
except ImportError:
    print_bak(comment, file=stderr)

# Additional



def print_helper(text, cond):
    print_bak(text, end=' ')
    print_bak(cond, file=stdout if cond else stderr)



print_bak('\n' + "Ex DoublyLinkedList".center(LINE_WIDTH, '='))
test_code(
    '__eq__+__iter__ test', '''lst=DoublyLinkedList(range(5))
print_bak('lst:',lst)
print_helper('lst==DoublyLinkedList(range(5)) -> T:',lst==DoublyLinkedList(range(5)))
print_helper('lst!=DoublyLinkedList(range(6)) -> T:',lst!=DoublyLinkedList(range(6)))
print_helper('lst!=list(range(5)) -> T:',lst!=list(range(5)))
print_helper('lst!=None -> T:',lst!=None)
print_helper('lst==DoublyLinkedList(lst) -> T:',lst==DoublyLinkedList(safe_iter(lst)))
print_helper('多 iter 测试  -> T:',
    [
        (x,y) for x in safe_iter(lst) for y in safe_iter(lst)
    ]==[
        (x,y) for x in range(5) for y in range(5)
    ])''')
test_code(
    '-slice test', '''lst='DoublyLinkedList(range(50))'
print_bak('list:',lst)
lst=eval(lst)
all_pass=1
for i in range(20):

sli=slice(randrange(-100,100),randrange(-100,100),randrange(1,10)*(randrange(2)*2-1))
    l1=list(lst[sli])
    l2=list(range(50)[sli])
    if l1!=l2:
        all_pass=0
        print_bak('FAIL:',sli,l1,l2,file=stderr)
        print_bak('RESULT:',l1,file=stderr)
        print_bak('SHOULD BE:',l2,file=stderr)
if all_pass:
    print_bak('PASS')''')

if Node.invalid_key:
    print_bak('非法调用:', dict(Node.invalid_key), file=stderr)
```



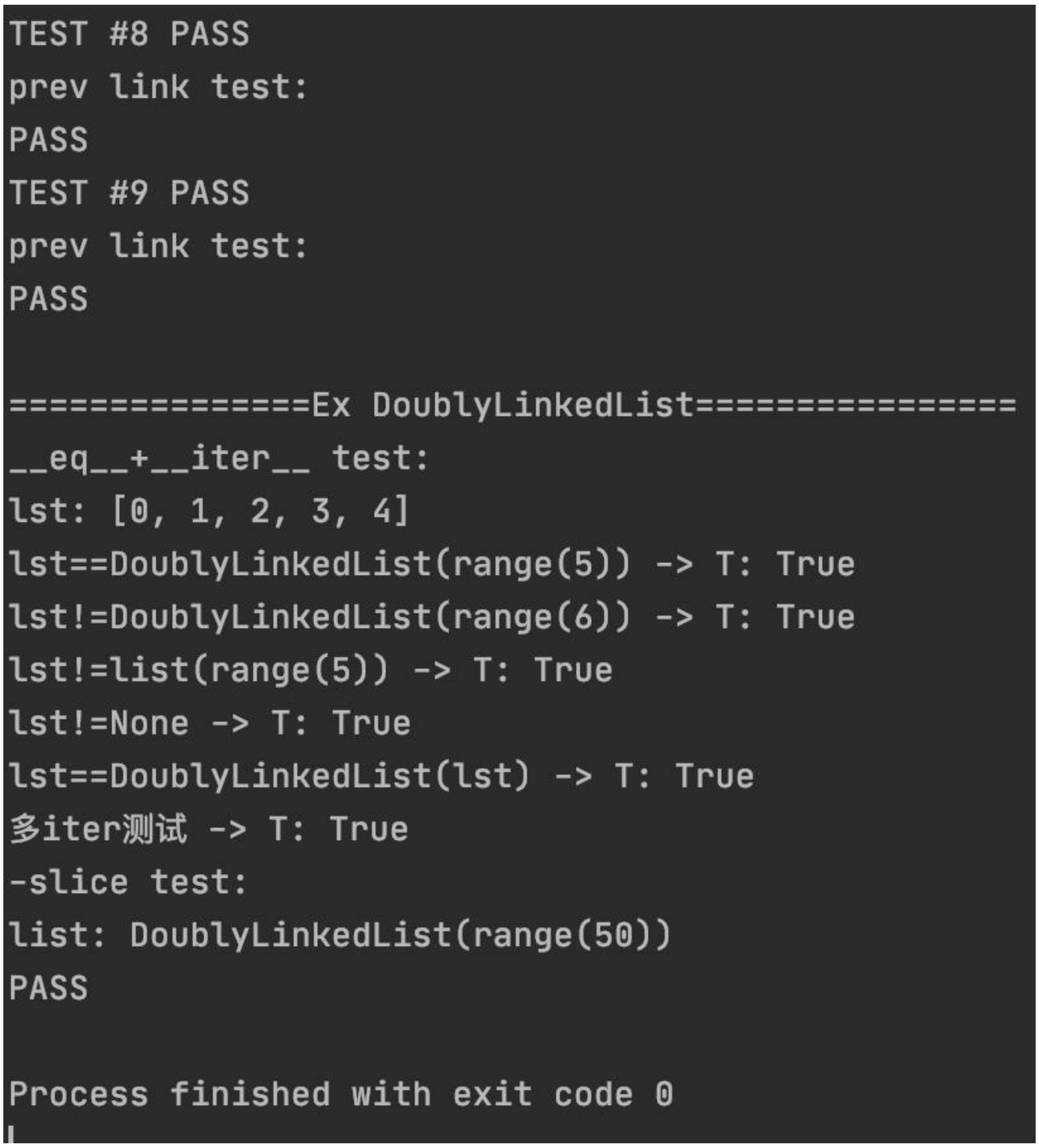
```
======== 1-calculate ========
32.0
20.0
1.0
======== 2-radix_sort ========
[1, 2, 3, 4, 5]
[4, 8, 18, 22, 30, 34, 55, 65, 91]
======== 3-HTMLMatch ========
True
False
======== 4-Link Stack & Link Queue ========
9 0
9 9
0 False
======== 5-DoublyLinkedList ========
16
False
False
10
2
18
2
[3, 0, 4, 8, 10, 12, 14, 16]
9
8
['10', 10, 14]
======== 1 中缀表达式求值 ========
>>> PASS
======== 2 基数排序 ========
```



```
======== 2 基数排序 ========
>>> PASS
======== 3 HTML MATCH ========
>>> PASS

===================1 LinkStack====================
TEST #0 PASS
TEST #1 PASS
TEST #2 PASS
TEST #3 PASS
TEST #4 PASS
TEST #5 PASS
TEST #6 PASS
TEST #7 PASS
TEST #8 PASS
TEST #9 PASS

===================2 LinkQueue====================
TEST #0 PASS
TEST #1 PASS
TEST #2 PASS
TEST #3 PASS
TEST #4 PASS
TEST #5 PASS
TEST #6 PASS
TEST #7 PASS
TEST #8 PASS
TEST #9 PASS
```



```
================3 DoublyLinkedList================
TEST #0 PASS
prev link test:
PASS
TEST #1 PASS
prev link test:
PASS
TEST #2 PASS
prev link test:
PASS
TEST #3 PASS
prev link test:
PASS
TEST #4 PASS
prev link test:
PASS
TEST #5 PASS
prev link test:
PASS
TEST #6 PASS
prev link test:
PASS
TEST #7 PASS
prev link test:
PASS
TEST #8 PASS
prev link test:
PASS
```


```
TEST #8 PASS
prev link test:
PASS
TEST #9 PASS
prev link test:
PASS

===============Ex DoublyLinkedList================
__eq__+__iter__ test:
lst: [0, 1, 2, 3, 4]
lst==DoublyLinkedList(range(5)) -> T: True
lst!=DoublyLinkedList(range(6)) -> T: True
lst!=list(range(5)) -> T: True
lst!=None -> T: True
lst==DoublyLinkedList(lst) -> T: True
多iter测试 -> T: True
-slice test:
list: DoublyLinkedList(range(50))
PASS

Process finished with exit code 0
```