第3章 特殊线性表

本章学习目标

- 理解栈的定义及存储结构。
- 理解队列的定义及存储结构。
- 理解串的定义及存储结构。


3-1

本章介绍栈、队列和串等在软件设计中常用的几种数据结构，它们的逻辑结构和线性表相同。其特点在于运算受到了限制：栈按“后进先出”的规则进行操作，队列按“先进先出”的规则进行操作，故称运算受限制的线性表；串的特殊性体现在组成串的节点是单个字符，所以存储时有一些特殊的技巧。

3.1 栈

3.1.1 栈的定义

栈实际上也是线性表，只不过是一种特殊的线性表。在这种特殊的线性表中，其插入与删除运算都只在线性表的一端进行。即在这种线性表的结构中，一端是封闭的，不允许插入与删除元素；另一端是开口的，允许插入与删除元素。在顺序存储结构下，对这种类型线性表的插入与删除运算是不需要移动表中其他数据元素的。这种线性表称为栈。**栈**(Stack)是限定在一端进行插入与删除的线性表，其示意图如图 3.1 所示。

在栈中，允许插入与删除的一端称为**栈顶**，而不允许插入与删除的另一端称为**栈底**。栈顶元素总是最后被插入的元素，从而也是最先能被删除的元素；栈底元素总是最先被插入的元素，从而也是最后才能被删除的元素。即栈是按照“**先进后出**”(First In Last Out，FILO)或“**后进先出**”(Last In First Out，LIFO)的原则组织数据的，因此，栈也被称为“先进后出”

表或“后进先出”表。通常用指针 top 来指向栈顶的位置，用指针 bottom 指向栈底的位置。往栈中插入一个元素称为入栈运算，从栈中删除一个元素（即删除栈顶元素）称为出栈运算。栈的操作只能在栈顶进行，栈顶指针 top 动态反映了栈中元素的变化情况。

栈这种数据结构在日常生活中也是很常见的。例如，子弹夹就是一种栈的结构，最后压入的子弹总是最先被弹出，而最先压入的子弹最后才能被弹出。又如，在用一端为封闭另一端为开口的容器装物品时，也是遵循“先进后出”或“后进先出”原则的。

抽象数据类型栈的定义如下。

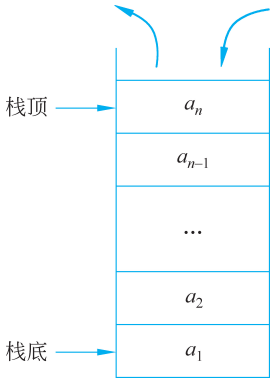


图 3.1 栈的示意图

```
ADT Stack {
    数据对象：D={ a_i | a_i ∈ ElemSet, i=1,2,…,n, n≥0 }
    数据关系：R={ <a_{i-1}, a_i> | a_{i-1}, a_i ∈ D, i=2,…,n }
                        约定 a_n 端为栈顶，a_1 端为栈底。
    基本操作：P={初始化栈，销毁栈，入栈，…}
} ADT Stack
```

栈的基本操作有以下几种。

（1）初始化操作：__init__(self, max_size=10)。

初始条件：栈 self 不存在。

操作结果：构造一个空栈。

（2）销毁操作：destroy(self)。

初始条件：栈 self 已存在。

操作结果：栈 self 被销毁。

（3）插入元素操作：push(self, item)。

初始条件：栈 self 已存在。

操作结果：插入元素 item 为新的栈顶元素。

（4）删除元素操作：pop(self)。

初始条件：栈 self 已存在且非空。

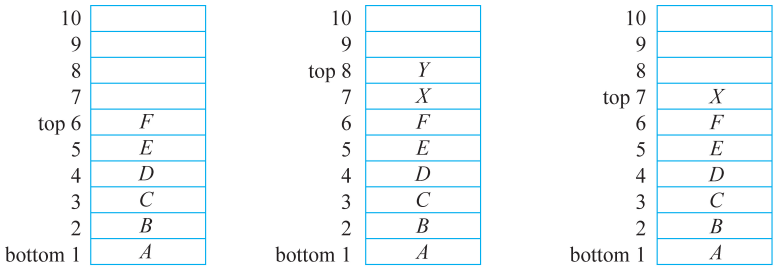
操作结果：删除 self 的栈顶元素，用 item 返回其值。

3.1.2 栈的存储及运算实现

与一般的线性表一样，在程序设计语言中，用一维数组 $S(1:m)$ 作为入栈出栈的顺序存储空间，其中，m 为栈的最大容量。通常，栈底指针指向栈空间的低地址一端（即数组的起始地址这一端）。如图 3.2 所示，图 3.2(a)是容量为 10 的栈顺序存储空间，栈中已有 6 个元素；图 3.2(b)与图 3.2(c)分别为入栈与出栈后的状态。

和线性表类似，栈也有两种存储表示方法，即顺序栈和链栈。以顺序栈为例，栈的顺序存储结构，其类型定义如下。

栈的基本运算有 4 种：初始化、入栈、出栈与读栈顶元素。下面分别介绍在顺序存储结构下栈的这几种运算。



(a) 有6个元素的栈　(b) 插入 X 和 Y 的栈　(c) 退出一个元素的栈

图 3.2　入栈和出栈示意图

1. 初始化运算

```
def __init__(self, max_size=10):
        """初始化顺序栈
        Args:
            max_size: 栈的最大容量,默认为 10
        """
        self.max_size =max_size                  #栈的最大容量
        self.stack =[None] * max_size            #动态分配的数组空间
        self.top =-1                             #栈顶指针,初始为-1 表示空栈
        self.bottom =0                           #栈底指针,固定为 0
        self.stacksize =max_size                 #栈的当前可用容量
```

2. 入栈运算

入栈运算是指在栈顶位置插入一个新元素。这个运算有两个基本操作：首先将栈顶指针进 1(即 top+1),然后将新元素插入到栈顶指针指向的位置。当栈顶指针已经指向存储空间的最后一个位置时,说明栈空间已满,不可能再进行入栈操作。这种情况称为栈“上溢”错误。

```
def push(self, item):
        """入栈运算
        Args:
            item: 要入栈的元素
        Returns:
            bool: 入栈是否成功
        """
        #如果栈满,自动扩容
        if self.is_full():
            new_size =self.stacksize * 2          #扩容为原来的 2 倍
            self._resize(new_size)
        #栈顶指针上移,放入元素
        self.top +=1
        self.stack[self.top] =item
        return True
```

3. 出栈运算

出栈运算是指取出栈顶元素并将其赋给一个指定的变量。这个运算有两个基本操作：

首先将栈顶元素(栈顶指针指向的元素)赋给一个指定的变量,然后将栈顶指针退 1(即 top—1)。当栈顶指针为 0 时,说明栈空,不可能进行出栈操作。这种情况称为栈“下溢”错误。

```
def pop(self):
        """出栈运算
        Returns:
            出栈的元素
        Raises:
            Exception: 栈为空时抛出异常
        """
        if self.is_empty():
            raise Exception("栈为空,无法执行出栈操作")
        #获取栈顶元素
        item =self.stack[self.top]
        self.stack[self.top] =None          #清空栈顶位置
        self.top -=1                        #栈顶指针下移
        #如果栈空且容量过大,可以缩容(可选)
        if self.is_empty() and self.stacksize >self.max_size * 2:
            self._resize(self.max_size)
        return item
```

4. 读栈顶元素

读栈顶元素是指将栈顶元素赋给一个指定的变量。必须注意,这个运算不会删除栈顶元素,只是将它的值赋给一个变量,因此,在这个运算中,栈顶指针不会改变。当栈顶指针为 0 时,说明栈空,读不到栈顶元素。

```
def get_top(self):
        """读栈顶元素(不弹出)
        Returns:
            栈顶元素
        Raises:
            Exception: 栈为空时抛出异常
        """
        if self.is_empty():
            raise Exception("栈为空,无法读取栈顶元素")
        return self.stack[self.top]
```

3.2 队列

3.2.1 队列的定义

在计算机系统中,如果一次只能执行一个用户程序,则在需要执行多个用户程序时,这些用户程序必须先按照到来的顺序进行排队等待。这通常是由计算机操作系统来进行管理的。在操作系统中,用一个线性表来组织管理用户程序的排队执行,其原则有以下几点。

(1) 初始时线性表为空。

(2) 当有用户程序到来时,将该用户程序加入到线性表的末尾进行等待。

(3) 当计算机系统执行完当前的用户程序后,就从线性表的头部取出一个用户程序执行。

由此可以看出,在这种线性表中,需要加入的元素总是插入到线性表的末尾,并且又总是从线性表的头部取出(删除)元素。这种线性表称为队列。

队列(Queue)是指允许在一端进行插入而在另一端进行删除的线性表。允许插入的一端称为**队尾**,通常用一个称为尾指针(rear)的指针指向队尾元素,即尾指针总是指向最后被插入的元素;允许删除的一端称为**队头**,通常用一个称为头指针(front)的指针指向队头元素。显然,在队列这种数据结构中,最先插入的元素将最先能够被删除,反之,最后插入的元素将最后才能被删除。因此,队列又称为"**先进先出**"的线性表,它体现了"先来先服务"的原则。在队列中,队尾指针 rear 与排头指针 front 共同反映了队列中元素动态变化的情况。图 3.3 是具有 6 个元素的队列示意图。

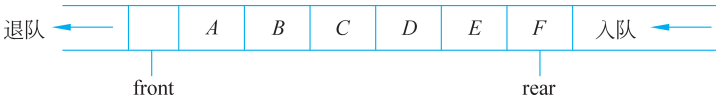
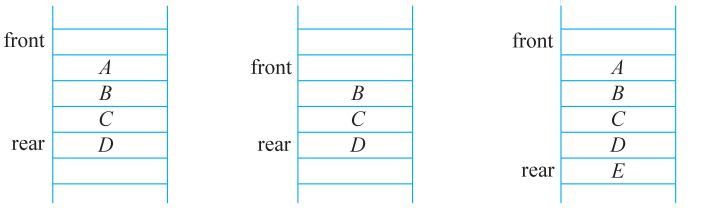


图 3.3 6 个元素的队列示意图

向队列的队尾插入一个元素称为入队运算,从队列的队头删除一个元素称为出队运算。由图 3.4 可以看出,在队列的末尾插入一个元素(入队运算)只涉及队尾指针 rear 的变化,而要删除队列中的队头元素(出队运算)只涉及队头指针 front 的变化。

与栈类似,在程序设计语言中,用一维数组作为队列的顺序存储空间。



(a) 一个队列　(b) 删除一个元素后的队列　(c) 插入一个元素后的队列

图 3.4 队列运算示意图

抽象数据类型队列的定义如下。

```
ADT Queue {
    数据对象:D={ a_i | a_i∈ElemSet, i=1,2,…,n,n≥0 }
    数据关系:R={ <a_{i-1},a_i> | a_{i-1},a_i∈D,i=2,…,n }
                约定其中 a_1 端为队头,a_n端为队尾
    基本操作:P={初始化队列,入队操作,出队操作,… }
} ADT Queue
```

队列的基本操作有以下几种。

(1) 初始化操作:__init__(self, maxsize=10)

初始条件:队列 self 不存在。

操作结果:构造一个空队列 self。

(2) 读队头元素：get_front(self)

初始条件：self 为非空队列。

操作结果：返回 self 的队头元素。

(3) 入队操作：enqueue(self, item)

初始条件 ：队列 self 已存在。

操作结果：插入元素 item 为 self 的新的队尾元素。

(4) 出队操作：dequeue(self)

初始条件：self 为非空队列。

操作结果：删除 self 的队头元素，并用 item 返回其值。

3.2.2 队列的存储及运算实现

和线性表类似，队列也可以有两种存储表示：循环队列和链队列。在实际应用中，队列的顺序存储结构一般采用循环队列的形式。

1. 循环队列

所谓循环队列，就是将队列存储空间的最后一个位置绕到第一个位置，队列循环使用，如图 3.5 所示。在循环队列结构中，当存储空间的最后一个位置已被使用而再要进行入队运算时，只要存储空间的第一个位置空闲，便可将元素加入第一个位置，即将存储空间的第一个位置作为队尾。在循环队列中，用队尾指针 rear 指向队列中的队尾元素，用队头指针 front 指向队头元素的前一个位置，因此，从队头指针 front 指向的后一个位置直到队尾指针 rear 指向的位置之间所有的元素均为队列中的元素。循环队列的初始状态为空，即 rear＝front＝m，如图 3.6 所示。

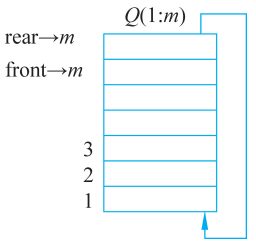
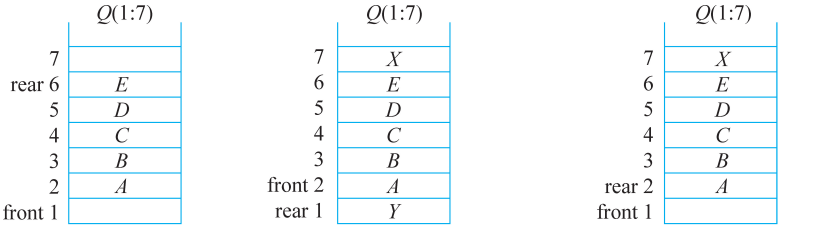


图 3.5 循环队列存储空间示意图



(a) 循环队列　(b) 加入 X、Y 的循环队列　(c) 退出一个元素的循环队列

图 3.6 循环队列运算

队列的顺序存储结构和顺序栈类似。由于队列的操作是在两端进行的，为了方便操作，采用以下的顺序存储结构，其类型定义为：

```
def __init__(self, maxsize=10):
        """初始化循环队列
        Args:
            maxsize: 队列的最大长度,默认为 10
        """
        self.maxsize =maxsize                    #队列的最大长度
```

```
        self.base = [None] * maxsize          #队列存储空间
        self.front = 0                        #队头指针
        self.rear = 0                         #队尾指针
        self._size = 0                        #当前队列中的元素个数
```

循环队列主要有两种基本运算：入队运算和出队运算。

1）入队运算

入队运算是指在循环队列的队尾加入一个新元素。这个运算有两个基本操作：首先将队尾指针加 1(即 rear+1)，然后将新元素插入到队尾指针指向的位置。当循环队列非空且队尾指针等于队头指针时，说明循环队列已满，不能进行入队运算，该情况称为“上溢”。

```
def enqueue(self, item):
        """入队运算
        Args:
            item: 要入队的元素
        Returns:
            bool: 入队是否成功
        """
        #如果队列已满，自动扩容
        if self.is_full():
            new_size = self.maxsize * 2          #扩容为原来的 2 倍
            self._resize(new_size)
        #将元素放入队尾
        self.base[self.rear] = item
        self.rear = (self.rear + 1) % self.maxsize
        self._size += 1
        return True
```

2）出队运算

出队运算是指在循环队列的队头位置退出一个元素并赋给指定的变量。这个运算有两个基本操作：首先将队头指针进 1(即 front=front+1)；然后将队头指针指向的元素赋给指定的变量。当循环队列为空时，不能进行出队运算，这种情况称为“下溢”。

```
def dequeue(self):
        """出队运算
        Returns:
            出队的元素
        Raises:
            Exception: 队列为空时抛出异常
        """
        if self.is_empty():
            raise Exception("队列为空，无法执行出队操作")
        #获取队头元素
        item = self.base[self.front]
        self.base[self.front] = None          #清空队头位置
        self.front = (self.front + 1) % self.maxsize
        self._size -= 1
        #如果队列很空且容量过大，可以缩容(可选)
```

```
        if self._size <self.maxsize // 4 and self.maxsize >10:
            new_size =max(10, self._size * 2)
            self._resize(new_size)
        return item
```

2. 链队列

用链表表示的队列简称为**链队列**。一个链队列显然需要一个头指针和一个尾指针才能唯一确定。为了操作方便起见，给链队列添加了一个表头节点，并令头指针指向表头节点。

链队列结构的类型定义如下：

```
class QNode:
    """链队列节点类"""
    def __init__(self, data=None):
        """初始化队列节点
        Args:
            data: 节点存储的数据
        """
        self.data =data
        self.next =None
    def __str__(self):
        """返回节点的字符串表示"""
        return str(self.data)
```

1）初始化操作

```
def __init__(self, maxsize=None):
        """初始化链队列
        Args:
            maxsize: 队列的最大长度,None 表示无限制
        """
        self.front =None          #队头指针
        self.rear =None           #队尾指针
        self._size =0             #当前队列中的元素个数
        self.maxsize =maxsize     #队列的最大长度(None 表示无限制)
```

2）入队运算

```
def enqueue(self, item):
        """入队运算
        Args:
            item: 要入队的元素
        Returns:
            bool: 入队是否成功
        """
        #检查队列是否已满
        if self.is_full():
            print("队列已满,无法入队")
            return False
        #创建新节点
        new_node =QNode(item)
```

```
        if self.is_empty():
            #队列为空,新节点既是队头也是队尾
            self.front =new_node
            self.rear =new_node
        else:
            #队列不为空,将新节点添加到队尾
            self.rear.next =new_node
            self.rear =new_node
        self._size +=1
        return True
```

3）出队运算

```
def dequeue(self):
        """出队运算
        Returns:
            出队的元素
        Raises:
            Exception: 队列为空时抛出异常
        """
        if self.is_empty():
            raise Exception("队列为空,无法执行出队操作")
        #获取队头节点
        node_to_remove =self.front
        item =node_to_remove.data
        #更新队头指针
        self.front =self.front.next
        #如果队列变空,更新队尾指针
        if self.front is None:
            self.rear =None
        #释放节点(断开连接)
        node_to_remove.next =None
        self._size -=1
        return item
```

3.3 串

3.3.1 串的定义

串(String)是字符串的简称。它是一种在数据元素的组成上具有一定约束条件的线性表,即要求组成线性表的所有数据元素都是字符,所以,人们经常又这样定义串:串是有限长的字符序列。串一般记作:

$$s='a_1a_2\cdots a_n' \quad (n\geqslant 0)$$

其中,s 是串的名称,用单引号括起来的字符序列是串的值;a_i 可以是字母、数字或其他字符;串中字符的数目 n 被称作串的长度。当 $n=0$ 时,串中没有任何字符,其串的长度为 0,

通常被称为**空串**。

串中任意连续的字符组成的子序列被称为该串的**子串**。包含子串的串又被称为该子串的**主串**。通常称字符在序列中的序号为该字符在串中的**位置**。称两个串是**相等**的，当且仅当两个串的长度相等，并且各个对应的字符也都相同。

抽象数据类型串的定义如下。

```
ADT String {
    数据对象：D={ a_i | a_i ∈ CharacterSet, i=1,2,…,n, n≥0 }
    数据关系：R={ <a_{i-1}, a_i> | a_{i-1}, a_i ∈ D, i=2,…,n }
    基本操作：P={初始化串，串的连接，串的比较，… }
} ADT String
```

串的基本操作有以下几种。

(1) 清除操作：clear(self)。

初始条件：串 self 存在。

操作结果：将 self 清为空串。

(2) 定位操作：index(self, pattern, start=0)。

初始条件：串 self 和 pattern 存在，pattern 是非空串，0≤start≤len(self)。

操作结果：若 self 在第 start 位置后存在和 pattern 相同的子串，则返回第一次出现 pattern 的位置；否则返回－1。

例如：

```
S="aabcababcaabca" ,T="abc"
Index(S, T, 4)  =  7;
Index(S, T, 12)  =  -1;
```

(3) 串替换操作：replace(self, old_str, new_str)。

初始条件：串 self，old_str 和 new_str 均已存在，且 old_str 是非空串。

操作结果：用 new_str 替换主串 self 中出现的所有与(模式串)new_str 相等的不重叠的子串。

例如：

```
S="aabcababcaabca" ,T="abc",V="S"
replace(S, T, V) ;  经置换后得到 S="aSabSaSa"
```

(4) 串插入操作：insert(pos, insert_str)。

初始条件：串 self 和 insert_str 存在，0≤pos≤len(S)。

操作结果：在 self 的第 pos 个字符前插入 insert_str。

例如：

```
S="Chater", T="rac",
```

则执行 insert(3, T)之后得到 S="Character"。

(5) 串删除操作：delete(pos,len)。

初始条件：串 self 存在，0≤pos≤len(self)。

操作结果：从串 self 中第 pos 个字符起删除长度为 len 的子串。

(6) 串连接操作：concat(s2)。

初始条件：串 self 和 s2 存在。

操作结果：返回由 self 和 s2 连接而成的新串。

例如：

```
S1=MyString("Hello")
S2 =MyString(" World")
S1.concat(S2)
```

求得：S1="Hello World"。

(7) 求子串：substring (pos , length)。

初始条件：主串存在 ，0≤pos≤len(s1)且 0≤length≤len(s1)－pos＋1。

操作结果：用 sub 返回主串 s1 的第 pos 个字符起长度为 len 的子串。

例如：

```
s7 =MyString("Programming")
sub1 =s7.substring(3, 5)
sub2 =s7.substring(7)
```

打印输出 s7 和 sub1、sub2 分别为：

```
原串 s7: Programming
子串 sub1: gramm
子串 sub2: min
```

对于串的基本操作集可以有不同的定义方法，在使用高级程序设计语言中的串类型时，应以该语言的参考手册为准。串的逻辑结构和线性表极为相似，区别仅在于串的数据对象约束为字符集。而串的基本操作和线性表有很大差别：在线性表的基本操作中，大多以“单个元素”作为操作对象；在串的基本操作中，通常以“串的整体”作为操作对象。

3.3.2 串的存储

在程序设计语言中，如果串只是作为输入或输出的常量出现，则只需存储此串的串值，即字符序列即可。但在多数非数值处理的程序中，串也以变量的形式出现。

串的定长顺序存储类似于线性表的顺序存储结构，在 Python 中，字符串的最大长度取决于使用的 Python 解释器支持的内存大小，以及操作系统和硬件的限制。理论上说，Python 中的字符串可以非常大，因为字符串在 Python 内部是以 Unicode 字符序列的形式存储的，而 Unicode 字符可以占用 1～4B。

串的实际长度可在这个预定义长度的范围内随意设定，超过预定义长度的串值则被舍去，称为“截断”。按这种串的表示方法实现串的运算时，其基本操作为“字符序列的复制”。

小结

本章介绍了3种特殊的线性表：栈、队列和串，它们的逻辑结构和线性表相同，其特点在于运算受到了限制。

(1) 栈是一种运算受限的线性表，它只允许在栈顶进行插入和删除等运算。栈又称为后进先出(Last In First Out)的线性表，简称LIFO结构。

(2) 栈适合采用顺序存储结构，通过栈顶指针能够访问到栈顶元素和进行插入、删除等运算。

(3) 队列是一种运算受限的线性表，它的运算限制与栈不同，它是一种只允许在一端进行插入操作，而在另一端进行删除操作的线性表。队列是一种先进先出(First In First Out)的线性表，简称FIFO。允许插入的一端称为队尾，允许删除的一端称为队头。

(4) 队列也适合采用顺序和链式两种存储结构。在顺序队列中，存储空间是首尾相接的一个环，故称为循环队列。队列的链式存储结构，其实也就是线性表的单链表，只不过它只能尾进头出而已，我们把它简称为链队列。

(5) 串也同样是线性表中的一种，只是它的特殊性在于组成串的节点是单个字符，所以存储时有一些特殊的技巧。

习题

1. 填空题

(1) 栈是限定在一端进行插入与删除的线性表，允许插入与删除的一端称为_____，而不允许插入和删除的另一端称为_____。栈是按照“先进后出”或者“后进先出”的原则组织数据的。

(2) 队列是指允许在一端进行插入而在另一端进行删除的线性表。允许插入的一端称为_____，允许删除的一端称为_____。队列又称为“先进先出”的线性表。

(3) 串或者字符串(String)是由0或者多个_____组成的有限序列。

(4) 一个栈的进栈序列为1，2，…，n，对应的出栈序列为S_1，S_2，…，S_n。若$S_2=3$，则S_3可能取值的个数为_____。

2. 单选题

(1) 若元素1、2、3、4、5依次进栈，则出栈次序不可能出现(　　)的情况。

A. 5、4、3、2、1　　B. 2、1、5、4、3

C. 4、3、1、2、5　　D. 2、3、5、4、1

(2) 一个队列的入队序列是1、2、3、4，则队列的出队序列是(　　)。

A. 1、2、3、4　　B. 4、3、2、1

C. 1、4、3、2　　D. 3、4、1、2

(3) 为解决计算机主机与打印机速度不匹配的问题,通常设一个打印机数据缓冲区。主机将要输出的数据依次写入该缓冲区,而打印机则依次从该缓冲区中取出数据。该缓冲区的逻辑结构应该是(　　)。

A. 队列　　B. 栈　　C. 线性表　　D. 有序表

(4) 依次在初始为空的队列中插入元素 a、b、c、d 以后,紧接着作了两次删除操作,此时的队头元素是(　　)。

A. a　　B. b　　C. c　　D. d

(5) 链式队列节点为(data,link),top 指向队头,若想删除队头节点,并将删除节点的值保存到 x 中,则应执行操作(　　)。

A. x=top. data;top=top. link　　B. top=top. link;x=top. link

C. x=top;top=top. link　　D. x=top. link

3. 简答题

(1) 简述栈和线性表的差别。

(2) 简述队列和栈这两种数据结构的相同点和异同点。

(3) 串通常有几类方法?

4. 分析题

对于一个栈,给出输入项 A、B、C、D,如果输入项序列为 A、B、C、D,试给出全部可能的输出序列。