第3章 栈和队列

3.1 由根及脉，本章导学

本章的知识结构如图 1-3-1 所示。

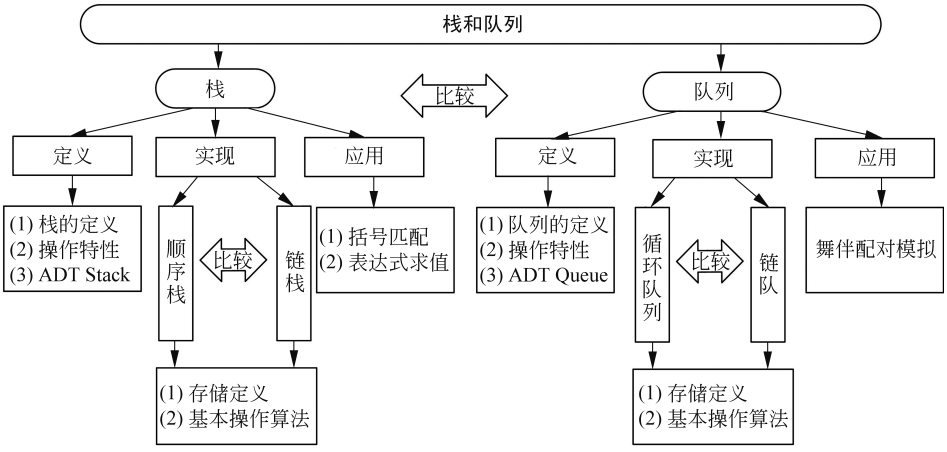


图 1-3-1 第 3 章知识结构图

本章包括两种操作受限的线性表：(堆)栈和队列。每一种结构分定义、实现和应用 3 个方面，定义中给出结构的定义、特性和抽象数据类型描述，实现中分顺序存储和链式存储两种实现。

3.1.1 栈

1. 栈的定义与特性

栈是限定只能在一端进行插入或删除操作的线性表。表中允许进行插入和删除操作的一端称为栈顶(top)。相应地，另一端称为栈底(bottom)。在栈顶插入元素的操作称为入栈/进栈/压栈，删除栈顶元素的操作称为出栈/弹栈。

栈具有“先进后出”或“后进先出”的特性。栈的抽象数据类型给出栈的逻辑结构及其上的 8 个基本操作的定义。

2. 顺序栈

顺序栈是采用顺序存储结构实现的栈。顺序栈的存储定义及存储示意图如图 1-3-2 所示。

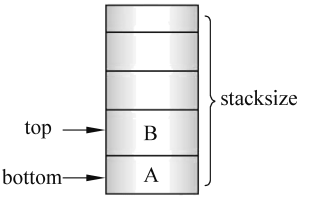


图 1-3-2　顺序栈存储示意图

```
template <class DT>
typedef struct SqStack
{  DT * bottom;              //栈底指针
   int top;                  //栈顶
   int stacksize;            //栈可用的最大容量
};
```

栈顶指向栈顶元素处,栈底在低地址处,即连续内存的起始位置。

在顺序栈的基本操作中,给出了 8 个基本操作中的 5 个操作的算法思想、算法描述和算法分析。

(1) 初始化栈(算法 3.1):申请存储空间,创建一个空栈。

(2) 销毁栈(算法 3.2):释放栈所占内存。

(3) 入栈(算法 3.3):栈非满时在栈顶增加一个元素。

(4) 出栈(算法 3.4):栈非空时删除栈顶元素。

(5) 获取栈顶元素(算法 3.5):取栈顶元素。

其余 3 个操作算法较简单,它们是测栈空、测栈满和清空栈。

3. 链栈

链栈是用链表实现的栈。链栈的结点定义及存储示意图如图 1-3-3 所示。

```
template <class DT>
struct SNode            //结点类型名
{  DT data;             //数据域,存储数据元素
   SNode * next;        //指针域,指向后继结点
};
```

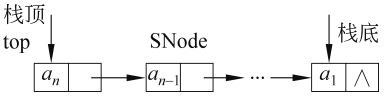


图 1-3-3　链栈存储示意图

采用无头结点的单链表表示链栈,头指针指向栈顶元素,也是链表标识。

申请链栈变量的语句为“SNode <DT> * S;”。

链栈操作中给出了 5 个操作的算法思想、算法描述和算法分析。

(1) 初始化(算法 3.6):栈顶指针置空,表示空栈。

(2) 销毁栈(算法 3.7):释放栈所占内存,与销毁单链表类似。

(3) 入栈(算法 3.8):在链表首元前增加一个数据元素,且头指针指向该新增结点。

(4) 出栈(算法 3.9):栈非空时删除首元结点,头指针指向其后继结点。

(5) 获取栈顶元素(算法 3.10):栈非空,取首元结点元素。

其余两个操作算法较简单，它们是测栈空和清空栈。

4. 栈的应用举例

栈的应用举例给出了以栈为工具的下列问题的求解。

(1) 括号匹配判断问题。采用栈存储左括号，与最近的右括号匹配。

(2) 中缀表达式求值，设置了操作数栈和操作符栈，完成中缀表达式的计算。

(3) 由中缀表达式求后缀表达式，利用操作符栈完成操作符按优先级高到低的运算顺序。

(4) 后缀表达式求值，利用操作数栈，实现操作数按运算符的优先级参与运算。

3.1.2　队列

1. 队列的定义和特性

队列是限定在表的一端进行插入、在另一端进行删除的线性表。允许插入的一端称为队尾，允许删除的一端称为队头或队首。在队尾插入新元素的操作称为进队或入队，从队首删除元素的操作称为出队或离队。队列具有“先进先出”或“后进后出”的特性。

队列的抽象数据类型给出队列的逻辑结构及其上的 8 个基本操作的定义。

2. 顺序列队

顺序队列是采用顺序存储结构实现的队列，顺序队列的存储定义及存储示意图如图 1-3-4 所示。

```
template <class DT>
typedef struct SqQueue
{  DT * base;                  //存储空间基地址
   int front;                  //队头指针,指向队头元素
   int rear;                   //队尾指针,指向队尾元素的后面
   int queuesize;              //队列容量
};
```

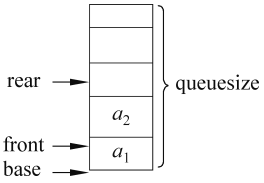


图 1-3-4　顺序队列存储示意图

为了解决顺序队列中的“假溢出”现象，假设连续存储空间的头、尾相接形成“循环队列”，相应地，队空、队满条件及入队和出队时队尾、队头指针变化如下：

▲ 队尾指针增 1：Q.rear=(Q.rear+1)%Q.queuesize。

▲ 队头指针增 1：Q.front=(Q.front+1)%Q.queuesize。

▲ 队空：Q.front==Q.rear。

▲ 队满：(Q.rear+1)%Q.queuesize==Q.front。

顺序队列的实现中给出了 8 个基本操作中 5 个操作的算法思想、算法描述和算法

分析。

(1) 初始化队列(算法 3.14)：申请存储空间，创建一个空队，初始化时，Q.front＝Q.rear＝0。

(2) 销毁队列(算法 3.15)：释放队列所占内存。

(3) 入队(算法 3.16)：队非满时在队尾增加元素。

(4) 出队(算法 3.17)：队非空时删除队头元素。

(5) 获取队头元素(算法 3.18)：取队头元素。

其余 3 个操作算法较简单，它们是测队空、测队满和清空队。

3. 链队

链队是采用链式存储结构实现的队列。链队采用有头结点的单链表存储，队头指针指向头结点，队尾指针指向链队尾元素。

存储定义及存储示意图如图 1-3-5 所示。

```
template <class DT>
struct QNode                    //结点
{ DT data;                      //数据域,存储数据元素
  QNode * next;                 //指针域,指向后继结点
};
template <class DT>
struct LinkQueue                //链队
{ QNode<DT> * front;            //队头指针
  QNode<DT> * rear;             //队尾指针
}
```

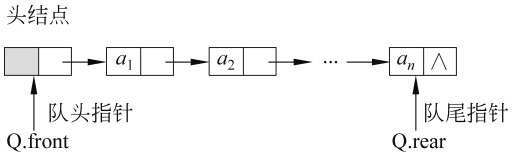


图 1-3-5　链队示意图

链队的实现中给出了 8 个基本操作中 5 个操作的算法思想、算法描述和算法分析。

(1) 初始化队列(算法 3.19)：创建头结点，指针域为空，front 与 rear 均指向它，表示一个空队。

(2) 销毁队列(算法 3.20)：释放队列所占内存，与销毁单链表类似。

(3) 入队(算法 3.21)：在队尾增加一个数据元素。

(4) 出队(算法 3.22)：队非空时删除首元结点。

(5) 获取队头元素(算法 3.23)：队非空时取首元结点元素。

其余 3 个操作算法较简单，它们是测队空、测队满和清空队。

4. 队列的应用举例

队列的应用给出了男女舞伴配对的模拟。分别用两个队列模拟男舞者和女舞者，按先来先出队进行男、女舞者配对。

3.2　拨云见日，谜点解析

3.2.1　线性表与栈

线性表和栈具有相同的逻辑结构，均为线性结构，即数据元素之间具有一对一的关系。栈是增、删操作上受限的线性表。

栈是增和删都只能在表的一端进行的线性表。如同一只桶，物品进、出只能通过桶口。对于桶，后放入物品压在先放的物品之上，要取下面的物品，必须先把上面的物品取出。堆栈中也一样，后进栈的数据元素压在先进栈的数据元素之上，要取出下面的元素必须先取走其上的数据，形成“先入后出/后进先出”的特性。

只有一个出口的桶，在不破坏桶的前提下，不能从桶的中间抽取东西或把东西塞到桶的中间位置。对栈的操作必须遵守类似约定。线性表中与位序相关的操作，在栈中通常均不可以实施。不要在栈中进行按位序访问、按值查找、在第 i 个位序插入或删除元素等操作。

栈与线性表同属线性结构，存储结构相同时，两者的初始化、销毁等操作方法类似。

3.2.2　线性表与队列

线性表和队列具有相同的逻辑结构，均为线性结构，即数据元素之间具有一对一的关系。队列是增、删操作上受限的线性表。

队列规定在表的一端插入，在另一端删除，如同日常生活中人们排的队，先来的排在队前可以先出队，后来的排在队尾。队列也是这样，元素依次在队尾入队，从队头出，形成“先入先出/后入后出”的特性。

生活中“不能插队”是共识，对数据结构的队列操作必须遵守类似约定。线性表中与位序相关的操作，在队列中通常均不可以实施。不要在队列中进行按位序访问、按值查找、在第 i 个位序插入或删除元素等操作。

队列与线性表同属线性结构，存储结构相同时，两者的初始化、销毁等操作方法类似。

3.2.3　栈、顺序栈和链栈

顺序栈和链栈是栈的两种不同存储结构的实现。前者采用顺序存储结构，后者采用链式存储结构。以栈作解决问题的工具时，顺序栈和链栈作用上没有什么区别。但不同的存储结构决定了他们各自的特性。如果使用顺序栈，初始化时需指定容量，这就要求编程人员预估所需空间，以免使用中空间不够影响系统性能。链栈没有栈满，但所需内存多于顺序栈。

3.2.4　栈顶指针

栈中元素的增、删均发生于栈的顶部，栈中通常设置栈顶指针标识栈顶。

顺序栈中，栈顶指针 top 的设置有两种方法：(1)指在栈顶元素处，本教材中采用此

方法；(2)指在栈顶元素的上方。两种方法在入栈、出栈、取栈顶元素及初始化时均有区别，详见表 1-3-1。

表 1-3-1 顺序栈两种栈顶指针设置的操作对比

序号	比较内容	方 法（1）	方 法（2）
1	top	指向栈顶元素处	指到栈顶元素上方
2	空栈	top＝－1	top＝0
3	入栈	• top＋＋，指向可用单元 • 入栈元素存入 top 所指单元	• 入栈元素存入 top 所指单元 • top＋＋
4	出栈	• 取 top 所指单元内容 • top－－	• top－－ • 取 top 所指单元内容
5	取栈元素	取 top 所指单元，top 不变	取 top－1 所指单元，top 不变

链栈用头指针 top 标识，如果有头结点，top 指向头结点，头结点后的结点为栈顶元素；如果没有头结点（本教材中采用此方法），top 所指结点为栈顶元素。两种方法在入栈、出栈、取栈顶元素及初始化时均有区别，详见表 1-3-2。

表 1-3-2 链栈两种栈顶指针设置的操作对比

序号	比较内容	有 头 结 点	无 头 结 点
1	top	指向头结点	指向栈顶元素
2	空栈	top 指向头结点	top＝NULL
3	入栈	• 在头结点后插入新结点 • top 不变	• 在 top 所指结点前插入新结点 • top 指向新结点
4	出栈	• 删除头结点的后继结点 • top 不变	• 删除 top 所指结点 • top 指向新首元结点
5	取栈元素	取头结点后结点数据元素	取 top 所指结点数据元素

3.2.5 队列、顺序队列、链队

顺序队列和链队是队列的两种不同存储结构的实现。前者采用顺序存储结构，后者采用链式存储结构。以队列作为解决问题的工具时，顺序队列和链队作用上没有什么区别。但不同的存储结构决定了它们各自的特性。如果使用顺序队列，初始化时需指定容量，这就要求编程人员预估所需空间，以免使用中因空间不够影响系统性能。链队没有栈满，但所需内存多于顺序队列。

3.2.6 队头、队尾指针

队列只能在队头删除元素，在队尾增加元素，为此设置队头指针和队尾指针。

顺序队列的队头指针 front 有两种设置方法：指向队头元素（本教材采用此方法）和指向队头元素前的 1 个单元。队尾指针 rear 也有两种设置方法：指向队尾元素和指向队

尾元素后的单元(本教材采用此方法)。不同的设置在出队、入队、取队头元素及初始化时均有区别,下面以循环队列为例,取其中两种设置作为对比:(1)本教材的设置;(2)front 指向队首元素前,rear 指向队尾元素。具体如表 1-3-3 所示。

表 1-3-3　循环队列两种指针设置的操作对比

序号	比较内容	方法(1)(教材)	方 法 (2)
1	队头队尾	front 指向队头元素 rear 指向队尾元素后	front 指向队头元素前 rear 指向队尾元素
2	初始化	front=0,rear=0	front=queuesize-1,rear=-1
3	入队	• 元素送入 rear 所指单元 • rear=(rear+1) %queuesize	• rear=(rear+1) %queuesize • 元素送入 rear 所指单元
4	出队	• 取 front 所指单元数据元素 • front=(front+1)%queuesize	• front=(front+1)%queuesize • 取 front 所指单元数据元素
5	取队头元素	• 取 front 所指单元数据元素 • front 不变	• 取 front 所指单元后一单元的数据元素 • front 不变

链队可以采用有头结点的单链表,也可以采用无头结点的单链表。如果有头结点,头指针指向头结点,头结点后的首元结点为队头元素;如果无头结点,头指针指向队头元素。两者的尾指针均指向链队的队尾元素。有无头结点在入队、出队、取队头元素及初始化时的区别如表 1-3-4 所示。

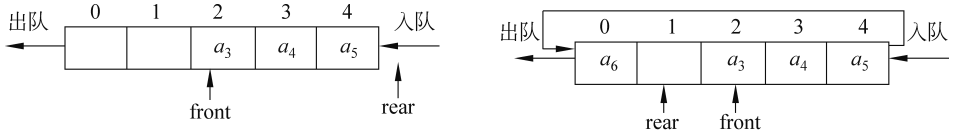
表 1-3-4　链队有无头结点的操作对比

序号	比较内容	有头结点(教材)	无头结点
1	队头、队尾	• front 指向头结点 • rear 指向尾结点	• front 指向队头结点 • rear 指向尾结点
2	空队	front 和 rear 均指向头结点	front 和 rear 均为 NULL
3	入队	• 在 rear 后插入新结点 • rear 指向新的尾结点	
4	出队	• 删除 front 的后继结点 • front 不变 • 删除后如果表空,rear 指向头结点	• 删除 front 所指结点 • front 指向新的首元结点 • 删除后如果表空,rear 为 NULL
5	取队头元素	取 front 后继结点数据元素	取 front 所指结点数据元素

3.2.7　“假溢出”及其相关问题

顺序队列中,数据元素从队尾入队时队尾 rear 指针后移,从队头出队时队头指针 front 后移。这样可能出现图 1-3-6(a)所示情况,rear 已经越界,但队列中依然有可用空间,此现象称为“假溢出”。

为了解决“假溢出”问题,将顺序队列看成首尾相连(见图 1-3-6(b)),入队时的 rear 移动为 rear=(rear+1)%queuesize,这样当 rear 在增加过程中越界时指针回到 0,就可避



（a）假溢出　　（b）循环队列

图 1-3-6　顺序队列“假溢出”与循环队列

免“假溢出”现象。但这样带来队空与队满判断条件相同的问题，入队时当 rear==front 时，队满；出队时当 rear==front 时为队空。教材中给出的处理方法是牺牲一个存储单元来区分队空和队满条件，分别如下。

队空：front == rear；

队满：(rear+1) % queuesize == front。

以下两种方法也可解决此问题。

方法一：计数法。通过一个计数器 k 累计队列中元素个数，入队时，k++，出队时 k--，当 k==0 时为队空，当 k==queuesize 时为队满。

事实上，队列中元素个数可以通过 front 和 rear 直接计算出，即元素个数=(rear-front+queuesize)% queuesize。因此，不设计数器也可以，但设了计数器可以更直观，且可省去每次判断时的个数计算。

方法二：标志法。设置一个标志 flag，入队时 flag=1，出队时 flag=0。当 front==rear 且 flag=1 时为队满；当 front==rear 且 flag=0 时为队空。

入队与出队算法描述如下。

```
// 入队算法
0 | template<class DT>
1 | bool EnQueue(SqQueue<DT>&Q ,DT e)      //在队尾插入一个新元素
2 | { if(Q.rear==Q.front && flag==1)       //队满,不能入队
3 |       return false;                    //返回 false
4 |   flag=1;                               //置入队标志
5 |   Q.base[Q.rear]=e;                     //元素 e 放在队尾指针处
6 |   Q.rear=(Q.rear+1)%Q.queuesize        //队尾指针增 1
7 |   return true;                          //返回 true
8 | }
// 出队算法
0 | template<class DT>
1 | bool DeQueue(SqQueue<DT>&Q,DT &e)       //删除队头元素
2 | { if(Q.front==Q.rear && flag==0)        //队空,不能出队
3 |       return false;                    //返回 false
4 |   flag=0;                               //置出队操作标识
5 |   e=Q.base[Q.front];                    //取队头元素,赋值给 e
6 |   Q.front=(Q.front+1)%Q.queuesize;      //队头指针加 1
7 |   return true;                          //返回 true
8 | }
```

3.2.8　递归与非递归

函数调用需要栈的支持，递归是特殊的函数调用，因此，递归是栈的典型应用之一。递归程序通常比较简洁，有时寥寥数语可以解决一个比较复杂的问题。但递归程序执行效率通常比较差，因此对递归问题常采用非递归算法。将递归算法转换为非递归算法有两种方法：一是用循环直接求解，不需要回溯；二是用栈实现递归中的回溯。

1. 直接求解

该方法可用于消除尾递归和单向递归，用循环代替递归。

尾递归是指在递归算法中，递归调用语句只有一处且在算法的最后。如求阶乘的算法，递归算法与非递归算法描述如下。

【递归算法】

```
1  long fact_1(int n)
2  {
3      if (n==0)
4          return 1;
5      else
6          return n*fact(n-1);
7  }
```

【非递归算法】

```
long fact_2(int n)
{
    for(i=1,p=1;i<=n;i++)
      p=p*i;
    return p;
}
```

递归算法中的语句 6 为递归调用，以此实现阶乘的连乘；非递归算法中由 for 循环实现阶乘的连乘。

单向递归是指递归算法中虽然有多处递归调用，但各递归调用的参数之间没有关系，并且这些递归调用语句都处在递归算法的最后。尾递归是单向递归的特例。如求斐波那契数列，递归算法与非递归算法描述如下。

【递归算法】

```
1  long fib_1(int n)
2  {  if (n==1 || n==2)
3       return 1;
4     else
5       return fib_1(n-1)+fib_1(n-2);
6  }
7
8
9
10
11
```

【非递归算法】

```
long fib_2(int n)
{  if (n==1 || n==2)
     return 1;
   else
   {  f1=1,f2=1;
      for(i=3;i<=n;i++)
      {  f=f1+f2;
         f1=f2,f2=f;
      }
   return f;
}
```

递归算法中语句 5 是尾递归，求第 n 项的值；非递归算法中，由 for 循环求第 n 项的值。

2. 借用栈

借用栈指用栈保存运算的中间结果，模拟递归中的回溯。如十进制数转为八进制数，

递归算法与非递归算法描述如下。

【递归算法】

```
long tran10to8_1(int n)
{  if (n!=0)
   {  tran10to8_1(n/8);
      cout<<n%8;
   }
}
```

【非递归算法】

```
void tran10to8_2 (int n)
{  Stack<int>s;
   while (n)
   {  Push(S ,n%8);
      n =n/8;
   }
   while (!StackEmpty(S))
   {  Pop(S ,e);
      cout<<e;
   }
}
```

递归算法中的语句 3,通过递归求八进制数的各位;非递归算法通过循环和堆栈,求得低位到高位的各个八进制数位,然后通过出栈依高位到低位输出各数位。

借用栈将递归转为非递归,在后续学习中会有许多实例。

3.3 积微成著,要点集锦

(1) 线性表、栈和队列具有相同的逻辑结构,均为线性结构,且均可以采用顺序存储和链式存储。

(2) 栈是插入和删除操作均只能在表的一端的线性表,具有“先进后出”或“后进先出”的特点。

(3) 队列是只能在表的一端插入、在另一端删除的线性表,具有“先进先出”或“后进后出”的特点。

(4) 顺序栈利用一组地址连续的存储单元由栈底到栈顶依次存放栈的数据元素。

(5) 顺序栈中,如果栈顶指针指向栈顶元素,则入栈时先移动栈顶指针,后存入元素;出栈则相反,先取栈顶元素,后移动栈顶指针。

(6) 在栈中增加元素和删除元素时栈顶指针的移动方向相反。

(7) 链栈用链表存储栈元素,没有头结点时,链表的头指针指向栈顶元素。

(8) 循环队列是利用一组地址连续的存储单元由队首到队尾依次存放队列的数据元素,且假设首尾相连。

(9) 入栈序列相同,出栈序列不一定相同。一个入栈序列可以有多个不同的出栈序列。

(10) 一个入队序列,只能有一个出队序列,且入队序列相同,出队序列一定相同。

(11) 不能在栈的任意位置增、删数据元素,也不能在队列的任意位置增、删元素。

(12) 无论是顺序栈、链栈,还是顺序队列和链队,在其中增加和删除数据元素,均不需要移动数据元素。

(13) 无论顺序栈还是链栈,在栈中增、删数据元素的时间复杂度为 $O(1)$。

(14) 无论顺序队列还是链队列,在队列中增、删数据元素的时间复杂度为 $O(1)$。

(15) 数据结构中的队列和生活中的队列区别是：生活中的队列，队头是不动的，队列中的元素会随着出队而移动；数据结构中的队列，队头随着出队而移动，队列中的元素不会随着出队而移动。

(16) 顺序队列中元素个数可以由队头指针和队尾指针计算出来。

(17) 队列中增、删数据元素时队头和队尾指针移动的方向一样。

(18) 顺序栈和循环队列与顺序表一样，使用中有容量是有限的，链栈和链队与链表一样，不存在容量受限问题。

(19) 中缀表达式的计算需设置两个栈：一个操作数栈，一个操作符栈。

(20) 表达式中双目操作符具有就近操作数参与运算的特点。

(21) 中缀表达式中运算符能够被执行的前提是其后的运算符优先级低于它。

(22) 由中缀表达式求后缀表达式，操作数按原顺序输出，操作符(括号除外)按优先级顺序插在操作数序列中。

(23) 栈可用于求解中需回溯的问题，如函数的嵌套调用。

(24) 队列用于解决类似资源先来先分配等问题。

3.4　启智明理，习题解答

3.4.1　主教材习题解答

一、填空题

1. 栈是__①__的线性表，其运算遵循__②__的原则。

【答案】 ①操作受限；②后进先出

【知识点】 栈的定义和特点

2. 若一个栈的输入序列是 1、2、3，则不可能的栈输出序列是_____。

【答案】 312

【解析】 3 要出栈必先入栈；按题意可知 3 入栈时 1、2 已入栈，因此，3 出栈后，只能是 2 出栈，其次是 1，312 不可是出栈序列。

【知识点】 栈的特点

3. 用 S 表示入栈操作，X 表示出栈操作，若元素入栈的顺序为 1234，为得到 1342 的出栈顺序，相应的 S 和 X 的操作串为_____。

【答案】 SXSSXSXX

【解析】 1 进，1 出，2 进，3 进，3 出，4 进，4 出，2 出

【知识点】 栈的操作

4. 循环队列的引入，其目的是_____。

【答案】 克服假溢出现象。

【知识点】 循环队列的意义

5. 队列是限制插入只能在表的一端，而删除在表的另一端进行的线性表，其特点是_____。

【答案】 先进先出

【知识点】 队列的定义

6. 已知链队列的头尾指针分别是 f 和 r，则将值 x 入队的操作序列是_____。

【答案】 s=new QNode<DT>; s->data=x; s->next=r->next;r->next=s;r=s;

【解析】 插入操作时，新结点链在队尾，并成为新的队尾。

【知识点】 链队的入队

7. 表达式求值是_____应用的一个典型例子。

【答案】 栈

【解析】 栈的应用有括号匹配校验、递归、进制转换、迷宫求解、表达式求值等；队列的应用包括舞伴问题、缓冲区、页面替换算法等。

【知识点】 栈的应用

8. 循环队列用数组 A[0..$m-1$]存放其元素值，已知其头尾指针分别是 front 和 rear，则当前队列的元素个数是_____。

【答案】 (rear-front+m)%m

【解析】 循环队列中，rear 可能比 front 大，也可能比 front 小或相等，元素个数是非负数，因此，求长度时需+表容量后对表容量求模。

【知识点】 循环队列

9. 以下运算实现在链栈上的进栈，请用适当语句补充完整。

```
void PushLStackTp * ls,DT x
{
  Lstack * p;
  p=new Lstack;
  _____①_____
    p->next=ls;
  _____②_____
}
```

【答案】 ①p->data=x; ②ls=p

【知识点】 链栈入栈操作

10. 以下运算实现在链队上的入队，请用适当语句补充完整。

```
void EnQueue(QueptrTp * lq,DT x)
{
  Lqueue * p;
  p=new Lqueue;
  _____①_____=x;
  p->next=NULL;
      (lq->rear)->next=_____②_____;
  _____③_____;
}
```

【答案】 ①p->data; ②p; ③lq->rear=p

【知识点】 链队的入队

二、简答题

1. 简述顺序栈的类型定义。

【答】 顺序栈的类型定义如下：

```
template <class DT>
struct SqStack
{  DT * base;              // 栈底指针
   int top;                // 栈顶
   int stacksize;          // 栈可用的最大容量
};
```

2. 简述链栈的类型定义。

【答】 链栈采用无头结点的单链表表示，头指针指向栈顶元素。链表结点定义如下：

```
template <class DT>
struct SNode           // 结点类型名
{  DT data;            // 数据域，存储数据元素
   SNode * next;       // 指针域，指向后继结点
};
```

3. 简述循环队列的类型定义。

【答】 循环队列定义如下，使用中假设首尾相连。

```
template <class DT>
struct SqQueue
{  DT * base ;           // 存储空间基地址
   int front;            // 队头指针，指向队首元素
   int rear;             // 队尾指针，指向队尾元素的后面
   int queuesize;        // 队列容量
};
```

4. 简述链队的类型定义。

【答】 链队采用有头结点的单链表，链表结点类型定义如下：

```
template <class DT>
struct QNode               // 结点类型名
{  DT data;                // 数据域，存储数据元素
   QNode * next;           // 指针域，指向后继结点
};
```

链队中除头指针外另设队尾指针指向队尾元素，链队类型定义如下：

```
template <class DT>
struct LinkQueue
{  QNode<DT> * front;          // 头指针
```

```
  QNode<DT> * rear;          // 队尾指针
}
```

5. 对于循环队列，试写出求队列长度的算法。

【答】

```
int length(SqQueue Q)
  {
      len=(Q.rear-Q.front+Q.queuesize)%Q.queuesize;
      return len;
  }
```

【分析】 见填空题 8。

【知识点】 循环队列

6. 设有编号为 1、2、3、4 的 4 辆列车顺序进入一个车站的站台。试写出这 4 辆列车开出车站的所有可能的顺序。

【答】 1234，1243，1324，1342，1432，2143，2134，2314，2341，2431，3241，3214，3421，4321。

【分析】 对于 n 个不同元素进栈，出栈序列的个数为 $\frac{1}{n+1}C_{2n}^{n}=\frac{1}{n+1}\frac{(2n)!}{n!\times n!}$，当 $n=4$ 时，有 14 种可能。进栈的值如果是升序，那根据栈的“先入后出”特点，序列中某个值后比它小的只能是逆序序列，例如，1、2、3、4 依次入栈，下列序列均是不可能的出栈序列：1423、2413、3124、3142、3412、4213、4231、4312、4123、4132 等。

【知识点】 栈的特点

7. 阅读下列程序，写出程序的运行结果。

```
#define SqStack_maxsize 40
typedef struct SqStack
{
    char data[SqStack_maxsize];
    int top;
}
Main()
{
    SqStack sq;
    int i;
    char ch;
    InitStack(&sq);
  for(ch='A';ch<='A'+12;ch++)
  {
      Push(&sq,ch);
      cout<<ch;
  }
```

```
    cout<<endl;
    while(!EmptyStack(sq))
    {
        Pop(&sq,&ch);
        cout<<cn;
    }
    cout<<endl;
}
```

【答】 ABCDEFGHIJKLM
MLKJIHGFEDCBA

【分析】 第一个 for 循环语句，将 A～M 共 13 个字母进栈并输出；第二个 while 循环语句，将 M～A 依次出栈并输出。

【知识点】 顺序栈的操作

8. 阅读下列算法，写出其完整的功能。

```
void reverse_list(LNode *head)
{
    SqStack ls,p;
    DataType x;
    InitStack(&ls);
    p=head->next;
    while(p!=NULL)
    {
        Push(&ls,p->data);
        p=p->next;
    }
    p=head->next;
    while(!EmptyStack(&ls))
    {
        Pop(&ls,&x);
        p->data=x;
        p=p->next
    }
}
```

【答】 借助一个栈将一个带头结点的单链表倒置。

【分析】 由 p=head->next 可知是一个带头结点的单链表，第一个 while 循环将链表结点依次进栈，第二个 while 循环依次出栈生成倒置的单链表。

【知识点】 链栈的操作

9. 写出下列中缀表达式的后缀表达式。

(1) －A＋B－C＋D；

(2) (A＋B)×D＋E/(F＋A×D)＋C；

(3) A&&B||!(E>F)。

【答】 (1) A－B＋C－D＋

(2) AB＋D×EFAD＋＋/＋C＋

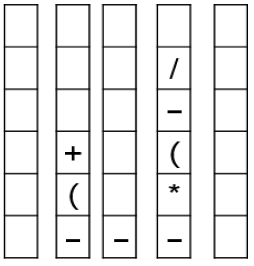
(3) AB&&EF! ||

【知识点】 由中缀表达式求后缀表达式

10. 用栈实现将中缀表达式 8－(3＋5)×(5－6/2)转换成后缀表达式：(1)写出其后缀表达式；(2)画出中缀表达式转变成后缀表达式过程中栈的变化过程图。

【答】 (1) 8 3 5 ＋ 5 6 2 / － * －

(2)



三、算法设计题

1. 假设以 I 和 O 分别表示入栈和出栈操作。栈的初态和终态均为空。

(1) 下面所示的序列中哪些是可操作序列？

A. IOIIOIOO　　B. IOOIOIIO　　C. IIIOIOIO　　D. IIIOOIOO

(2) 写出一个算法，判定所给的操作序列是否合法。若合法，返回 true，否则返回 false(假定被判定的操作序列已存入一维数组中)。

【解】 (1) 可操作序列为 A、D。

(2) 此算法与括号匹配类似。扫描操作序列，I 进栈，碰到 O 时，出栈。当栈的终态为空时，为可操作序列。遇到下列情形为不可操作序列：

① 扫描到 O 时，栈为空；②栈的终态非空。

算法描述如下：

```
bool match(char *exp)
{ InitStack<char>(S);                          //创建栈
  flag=1;                                      //匹配标志
  ch=*p++;                                     //取操作符号
  while(ch!='\0'&& flag==1)                    //未扫描完操作序列且匹配
  {  switch(ch)
     {  case 'I':                              //I 进栈
          Push(S,ch);
          break;
     case 'O':                                 //栈非空,出栈
         if(!StackEmpty(S))
             Pop(S,x);
         else                                  //栈空,不可操作序列
             flag=0;
       break;
  }
```

```
  if(StackEmpty(S) && flag)                          //可操作序列
    return true;
  else                                               //不可操作序列
      return false;
}
```

2. 假设以数组 cycque[$m+1$](数组范围设为 0～m)存放循环队列的元素,同时设变量 rear 和 quelen 分别指示循环队列中队尾元素位置和内含元素的个数。试给出此循环队列的队满条件,并且写出相应的入队列和出队列的算法。

【解】 设 cycque[m]、rear、quelen 皆为全局变量,当 quelen=0 时队空,quelen=m+1 时队满,相应入队和出队算法描述如下:

```
//入队列
template <class DT>
bool Enqueue(DT cycque[m],DT e)
{  if (quelen==m+1)                                  //队满
     return false;                                   //不能入队
   else
   {  real=(real+1)%(m+1);                           //队尾指针后移
      cycque[real]=e;                                //元素入队
      quelen++;
   }
   return true;
}
//出队
template <class DT>
bool DeQueue(DT cycque[m], DT &e)
{  if (quelen ==0)                                   //空队
     return false;                                   //不能出队
   else
   {  front =(real-quelen +1+(m+1))%(m+1);           //队头指针后移
      e=cycque [front];                              //元素出队
      quelen--;
      return true;
   }
}
```

3. 借助栈(可用栈的基本运算)来实现单链表的逆置运算。

【解】 设单链表有头结点。扫描单链表,将元素依次入栈;扫描单链表,将栈中元素依次出栈给单链表结点赋值。

算法描述如下:

```
Void invert(LNode<DT> * &L)
```

```
    {  InitStack(s);                                    //创建栈
       p=L->next                                        //从首元结点开始
       while (p!=NULLl)                                 //结点非空
       {  Push(s, p->data);                             //入栈

          p=p->next;
    }
    p=L->next;                                          //指向首元结点
    while(!EmptyStack(s))                               //栈不空
    {  Pop(s, p->data);                                 //出栈至链表
       p=p->next;                                       //下一个结点
    }
}
```

3.4.2 自测题及解答

一、判断题

1. 栈和线性表是两种不同的数据结构,它们的数据元素的逻辑关系不同。

【答案】 错误

【解析】 栈是一种只在表的一端进行增、删操作的特殊线性表。

【知识点】 栈定义

2. 栈底元素是不能删除的元素。

【答案】 错误

【解析】 栈底元素是最后可以出栈的元素,出栈表示删除。

【知识点】 栈

3. 顺序栈中元素值的大小是有序的。

【答案】 错误

【解析】 顺序栈是指用顺序存储的栈,栈中的元素不一定是有序的。

【知识点】 顺序栈

4. 栈顶元素和栈底元素有可能是同一个元素。

【答案】 正确

【解析】 当栈中只有一个元素时就是这种情况。

【知识点】 栈

5. 若用 data[1.. m]表示顺序栈的存储空间,则对栈的进栈、出栈操作最多只能进行 m 次。

【答案】 错误

【解析】 可以进行任意多次交替的进栈、出栈操作,但栈中最多只有 m 个元素。

【知识点】 栈的操作

6. 空的顺序栈没有栈顶指针。

【答案】 错误

【解析】 空栈指栈中没有元素,但顺序栈一定要有栈顶指针。

【知识点】　顺序栈

7. 在顺序栈中，将栈底放在数组的任意位置不会影响运算的时间性能。

【答案】　错误

【解析】　在顺序栈中，如果将栈底放在数组的两端，其进栈、出栈运算的时间性能都是最好的。如果将栈底放在数组的中间，要么将数组改为循环的(需要保存该栈底位置)，要么移动元素，其时间性能都不如将栈底放在数组两端好。

【知识点】　顺序栈

8. 循环队列不存在空间上溢出的问题。

【答案】　错误

【解析】　循环队列只是不存在假溢出，但因为采用的是顺序存储，所以，仍然存在空间上溢出的问题。

【知识点】　循环队列

9. 在采用单链表作为链栈时必须带有头结点。

【答案】　错误

【解析】　有无头结点的链表均可以用于存储链栈。因为操作只发生在栈顶，常用不带头结点的单链表表示链栈。

【知识点】　链栈

10. 顺序队列采用数组存放队中元素，而数组具有随机存取特性，因此，在顺序队列中可以随机存取元素。

【答案】　错误

【解析】　顺序队列采用数组存放队中元素，尽管数组具有随机存取特性，但队列对元素的访问和增、删操作只能发生在两端，因此，顺序队列不可以随机存取元素。

【知识点】　队列定义

11. 若用不带头结点的非循环单链表来表示链队，则可以用“队首指针和队尾指针的值相等”作为队空的标志。

【答案】　错误

【解析】　应该用“队首指针和队尾指针的值均为 NULL”作为队空的标志，因为当链队中只有一个结点时队首指针和队尾指针的值相等。

【知识点】　链队

12. 若采用“队首指针和队尾指针的值相等”作为循环队列为空的标志，则在设置一个空队时只需将队首指针和队尾指针赋同一个值，不管什么值都可以。

【答案】　正确

【解析】　因为无论出队和入队都要进行求余运算，将队首指针和队尾指针转化为有效的顺序队下标值，所以是正确的。

【知识点】　队列

二、单项选择题

1. 假定利用数组 a[n]顺序存储一个栈，用 top 表示栈顶指针，用 top＝＝－1 表示栈空，并已知栈未满，当元素 x 进栈时所执行的操作为(　　)。

A. a[－－top]＝x　　B. a[top－－]＝x

C. a[＋＋top]＝x　　D. a[top＋＋]＝x

【答案】 C

【解析】 初始时 top 为－1，则第一个元素入栈后，top 为 0，即指向栈顶元素，故入栈时应先将指针 top 加 1，再将元素入栈，只有选项 C 符合题意。

【知识点】 顺序栈的基本操作

2. 若一个栈的输入序列是 1，2，3，…，n，输出序列的第一个元素是 n，则第 i 个输出元素是(　　)。

A. 不确定　　B. $n-i$

C. $n-i-1$　　D. $n-i+1$

【答案】 D

【解析】 第 n 个元素第一个出栈，说明前 $n-1$ 个元素都已经按顺序入栈，由“先进后出”的特点可知，此时的输出序列一定是输入序列的逆序，故正确答案是 D。

【知识点】 栈的特点

3. 一个栈的输入序列为 1，2，3，…，n，输出序列的第一个元素是 i，则第 j 个输出元素是(　　)。

A. $i-j-1$　　B. $i-j$　　C. $j-i+1$　　D. 不确定

【答案】 D

【解析】 当第 i 个元素第一个出栈时，则 i 之前的元素可以依次排在 i 之后出栈，但剩余的元素可以在此时进栈并且也会排在 i 之前的元素出栈，因此，第 j 个出栈的元素是不确定的。

【知识点】 栈的特点

4. 若一个栈的输入序列是 P_1，P_2，…，P_n，输出序列是 1，2，3，…，n，若 $P_3=1$，则 P_1 的值(　　)。

A. 可能是 2　　B. 一定是 2　　C. 不可能是 2　　D. 不可能是 3

【答案】 C

【解析】 入栈序列是 P_1，P_2，…，P_n，由于 $P_3=1$，即 P_1，P_2，P_3 连续入栈后，第一个出栈元素是 P_3，说明 P_1，P_2 已经按序进栈，根据先进后出的特点可知，P_2 必定在 P_1 之前出栈，而第二个出栈元素是 2，而此时 P_1 不是栈顶元素，因此 P_1 的值不可能是 2。

【知识点】 栈的特点

5. 在一个具有 n 个单元的顺序栈中，假设以地址高端作为栈底，以 top 作为栈顶指针，则当做进栈处理时，top 的变化为(　　)。

A. top 不变　　B. top＝0　　C. top－－　　D. top＋＋

【答案】 C

【解析】 以高端为栈底，进栈时，栈顶指针往低地址处移动，是 top－－。正确答案是 C。

【知识点】 顺序栈

6. 向一个栈顶指针为 top 的链栈中插入一个 s 所指结点时，其操作步骤为(　　)。