第3章 栈和队列

3.1 考纲要求及分析

考纲要求

(1) 栈和队列的基本概念。
(2) 栈和队列的顺序存储结构。
(3) 栈和队列的链式存储结构。
(4) 栈和队列的应用。

考纲分析

本章是必考内容,出题形式主要以选择题为主。本章要求:

(1) 理解栈和队列的定义及其操作特性,掌握栈和队列对插入和删除的操作定义。
(2) 对于栈和队列的存储结构,掌握顺序栈、链栈、共享栈、顺序队列、循环队列、链队列的存储方法,以及栈空、栈满、队空、队满的判定条件。
(3) 掌握栈和队列的插入、删除、判空等基本操作的算法描述和时间性能。
(4) 理解栈和队列的应用,例如,子程序调用、表达式求值、括号匹配等。

对于栈,常考的一类题是考查栈的后进先出特性,例如给定一个入栈序列,判断某个出栈序列的合法性(或不合法性),共享栈也是一个常考点。对于队列,循环队列是一个常考点,注意队空、队满的判定条件、队列长度的计算。

本章有一个难点是关于栈的证明题,主要采用反证法应用栈的操作特性来完成;有一个结合点是将栈、队列、链表和数组相结合,主要考查是否掌握栈和队列的操作特性,以及链表和数组的存储特点;有一个复杂的应用是递归,主要考查是否理解栈在递归调用过程中的作用,以及应用栈实现递归函数到非递归函数的转换。

由于栈和队列的算法比较简单,通常不会单独以算法设计题的形式出题,在树和图的算法设计中,栈和队列通常作为辅助数据结构,因此,需要熟练掌握栈和队列的基本操作语句。

3.2 栈

3.2.1 考核知识点

1. 栈的定义(★☆☆☆☆ ◆◇◇◇◇)

栈是限定仅在表尾进行插入和删除操作的线性表。允许插入和删除的一端称为栈顶，另一端称为栈底，不含任何数据元素的栈称为空栈。

【说明】 插入操作也称为入栈或压栈，删除操作也称为出栈或弹栈。

2. 栈的操作特性(★★★★★ ◆◆◇◇◇)

栈的操作具有后进先出的特性。

3. 栈的基本操作(★★★☆☆ ◆◆◇◇◇)

通常，栈的基本操作有：

- InitStack(S)：初始化一个空栈 S。
- Push(S,x)：在栈 S 中插入一个元素 x。
- Pop(S)：删除并返回栈 S 的栈顶元素。
- GetTop(S)：读取栈 S 的栈顶元素但不删除。
- Empty(S)：如果栈 S 为空，返回 1，否则，返回 0。

4. 顺序栈及其实现(★★★☆☆ ◆◇◇◇◇)

1) 顺序栈的存储结构定义

栈的顺序存储结构称为**顺序栈**，通常把数组中下标为 0 的一端作为栈底，同时附设指针 top 指示栈顶元素在数组中的位置。设数组长度为 StackSize，其存储结构定义如下：

```
#define StackSize 100
typedef struct
{
  ElemType data[StackSize];  //ElemType 表示不确定的数据类型
  int top;
} SeqStack;
```

2) 顺序栈基本操作的实现

顺序栈的基本操作本质上是顺序表基本操作的简化，插入和删除操作只在栈顶(即表尾)进行，栈空时栈顶指针 top＝－1；栈满时栈顶指针 top＝StackSize－1。入栈时，栈顶指针 top 加 1；出栈时，栈顶指针 top 减 1。

根据栈的操作定义很容易写出顺序栈的入栈和出栈算法。

顺序栈入栈算法 Push

```
void Push(SeqStack S, ElemType x)
{
  if(S.top==StackSize-1) printf("上溢");
  else {
    S.top++;
    S.data[S.top]=x;
  }
}
```

顺序栈出栈算法 Pop

```
ElemType Pop(SeqStack S)
{
  if(S.top==-1) printf("下溢");
  else {
    x=S.data[S.top];
    S.top--;
    return x;
  }
}
```

顺序栈基本操作的算法都非常简单，并且其时间复杂度均为 $O(1)$。

5. 两栈共享空间(★★★☆☆ ◆◆◇◇◇)

1）两栈共享空间的存储结构定义

在一个程序中如果同时使用具有相同数据类型的两个栈时，一种可取的方法是充分利用顺序栈单向延伸的特性，使用一个数组来存储两个栈，让一个栈的栈底为该数组的始端，另一个栈的栈底为该数组的末端，每个栈从各自的端点向中间延伸，如图 3-1 所示。

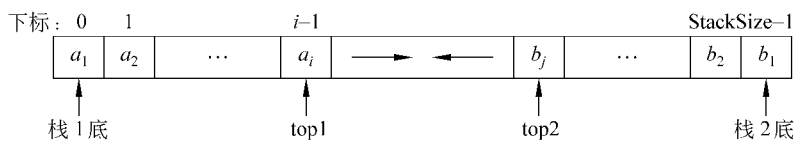

图 3-1 两栈共享空间示意图

其中，top1 和 top2 分别为栈 1 和栈 2 的栈顶指针。设 StackSize 为整个数组空间的大小，其存储结构定义如下：

```
#define StackSize 100
typedef struct
{
  ElemType data[StackSize];      //ElemType 表示不确定的数据类型
  int top1, top2;
} BothStack;
```

2）两栈共享空间上基本操作的实现

设i表示整型数值，它只取1和2两个值，当i=1时，表示对栈1操作，当i=2时表示对栈2操作。当top1=-1时栈1为空；当top2=StackSize时栈2为空。当top1=top2-1(或top2=top1+1)时为栈满。另外，当新元素压入栈2时，栈顶指针top2不是加1而是减1；当从栈2删除元素时，top2不是减1而是加1。

两栈共享空间入栈算法 Push

```
void Push(BothStack S, int i, ElemType x)
{
    if(S.top1 ==S.top2-1) printf("上溢");
    else {
       if(i==1) S.data[++S.top1]=x;
       if(i==2) S.data[--S.top2]=x;
    }
}
```

两栈共享空间出栈算法 Pop

```
ElemType Pop(BothStack S, int i)
{
    if(i==1) {    //将栈1的栈顶元素出栈
       if(S.top1==-1) printf("下溢");
       else return S.data[S.top1--];
    }
    if(i ==2) {    //将栈2的栈顶元素出栈
       if(S.top2 ==StackSize) printf("下溢");
       else return S.data[S.top2++];
    }
}
```

【说明】 只有当两个栈的空间需求有相反的关系时，两栈共享空间才会奏效，也就是说，最好一个栈增长时另一个栈缩短。

6. 链栈及其实现(★★★☆☆ ◆◆◇◇◇)

1）链栈的存储结构定义

栈的链接存储结构称为**链栈**。通常链栈用单链表表示，其结点结构与单链表相同。

【说明】 链栈没有必要像单链表那样为了运算方便附加一个头结点。

2）链栈上基本操作的实现

链栈的插入和删除操作只需处理栈顶即第一个位置的情况。链栈的插入算法如下：

链栈插入算法 Push

```
void Push(Node *top, ElemType x)
 {
  s=(Node *)malloc(sizeof(Node));
```

```
    s->data=x;  //申请一个数据域为x的结点s
    s->next=top; top=s;       //将结点s插在栈顶
}
```

链栈的删除算法如下：

链栈删除算法 Pop

```
ElemType Pop(Node * top)
{
  if(top==NULL) printf("下溢");
  else {
    x=top->data;                    //暂存栈顶元素
    p=top; top=top->next;           //将栈顶结点摘链
    free p;
    return x;
  }
}
```

链栈上基本操作的算法都非常简单，其时间复杂度均为 $O(1)$。

3.2.2 典型题解析

1. 选择题

试题点拨 主要考查栈的基本操作(初始化、进栈、出栈、判空等)在不同存储结构(顺序栈和链栈)下的执行过程，对于顺序栈注意栈底的位置和栈顶指针的变化，对于链栈注意如何用链表实现栈以及插入和删除操作的位置。

栈最重要的考点是元素以同样顺序进栈后判断出栈的不同情况。两栈共享空间也是一个常见的考点，注意存储方法、栈底的位置和栈顶指针的变化。

(1) 经过以下栈运算后，x 的值是(　　)。

```
InitStack(s); Push(s,a); Push(s,b); Pop(s,x); GetTop(s,x);
```

A. a　　B. b　　C. 1　　D. 0

【解答】 A

【分析】 本题要求熟悉栈的基本操作，理解所给运算的含义。InitStack(s)表示对栈 s 进行初始化；Push(s,a)表示将元素 a 压入栈 s 中；Pop(s,x)表示将栈 s 的栈顶元素弹出并送入变量 x 中；GetTop(s,x)表示取栈顶元素并送入变量 x 中但不删除该元素。

(2) 经过以下栈运算后，StackEmpty(s)的值是(　　)。

```
InitStack(s); Push(s,a); Push(s,b); Pop(s,x); Pop(s,y);
```

A. a　　B. b　　C. 1　　D. 0

【解答】 C

【分析】 注意 StackEmpty(s)的返回值，如果栈 s 为空，则返回 1，否则返回 0。

(3)(　　)不是栈的基本运算。

A. 删除栈顶元素　B. 删除栈底元素　C. 判断栈是否为空　D. 将栈置为空栈

【解答】 B

【分析】 因为栈的操作满足后进先出,所以,通常不能删除栈底元素。

(4) 设有一个空栈,栈顶指针为1000H(十六进制,下同),每个元素需要1个单位的存储空间,则执行PUSH, PUSH, POP, PUSH, POP, PUSH, POP, PUSH操作后,栈顶指针的值为(　　)。

A. 1002H　　B. 1003H　　C. 1004H　　D. 1005H

【解答】 A

【分析】 栈顶指针的位置只与执行的操作有关,而与插入或删除的元素无关。执行PUSH操作后栈顶指针的值加1,执行POP操作后栈顶指针的值减1。

(5) 一个栈的入栈序列是{1,2,3,4,5},则栈的不可能的输出序列是(　　)。

A. {5,4,3,2,1}　　B. {4,5,3,2,1}

C. {4,3,5,1,2}　　D. {1,2,3,4,5}

【解答】 C

【分析】 此题有一个技巧:在输出序列中任意元素后面不能出现比该元素小并且是升序(指的是元素的序号)的两个元素。

(6) 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. $n-i+1$

【解答】 D

【分析】 此时,输出序列一定是输入序列的逆序。

(7) 若一个栈的输入序列是1,2,3,…,n,其输出序列是p_1, p_2, …, p_n,若$p_1=3$,则p_2的值(　　)。

A. 一定是2　　B. 一定是1　　C. 不可能是1　　D. 以上都不对

【解答】 C

【分析】 由于$p_1=3$,说明1,2,3均入栈后3出栈,此时可能将当前栈顶元素2出栈,也可以继续执行入栈操作,因此p_2的值可能是2,但一定不能是1,因为1不是栈顶元素。

(8) 若一个栈的输入序列是p_1,p_2,…,p_n,其输出序列是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

【解答】 C

【分析】 由于$p_3=1$,则入栈序列是p_1,p_2,…,p_n,第一个出栈的元素是p_3,即p_1, p_2, p_3入栈后执行出栈操作,第二个出栈的元素是2,而此时p_1不是栈顶元素,因此,p_1的值不可能是2。

(9) 当字符序列t3_依次通过栈,输出长度为3且可用作C语言标识符的序列有(　　)。

A. 4个　　B. 5个　　C. 3个　　D. 6个

【解答】 C

【分析】 输出长度为3说明将字符序列全部出栈，可以作为C语言标识符的序列只能以字母t或下划线_开头，而栈的输出序列中以字母t或下划线_开头的有三个，分别是t3_、t_3和_3t。

(10) 在一个具有 n 个单元的顺序栈中，假定以地址低端(即下标为0的单元)作为栈底，以top作为栈顶指针，当出栈时，top的变化为(　　)。

A. 不变　　B. top=0;　　C. top=top-1;　　D. top=top+1;

【解答】 C

【分析】 栈底固定在数组的低端，出栈时栈顶指针top需要减1；如果栈底固定在数组的高端，则出栈时栈顶指针需要加1。

(11) 向一个栈顶指针为h的带头结点的链栈中插入指针s所指的结点时，应执行(　　)。

A. h->next=s;　　B. s->next=h;

C. s->next=h; h->next=s　　D. s->next=h->next; h->next=s;

【解答】 D

【分析】 结点s应插在头结点的后面。

(12) 从栈顶指针为top的链栈中删除一个结点，用x保存被删除结点的值，则执行(　　)。

A. x=top; top=top->next;

B. x=top->data;

C. top=top->next; x=top->data;

D. x=top->data; top=top->next;

【解答】 D

【分析】 删除链栈中的第一个结点，即指针top指向的结点。备选答案A保存的是栈顶指针；备选答案B只保存了被删结点的值，没有执行删除操作；备选答案C保存的是被删结点的后继结点的值。

(13) 设数组S[n]作为两个栈S1和S2的存储空间，对任何一个栈只有当S[n]全满时才不能进行进栈操作。为这两个栈分配空间的最佳方案是(　　)。

A. S1的栈底位置为0，S2的栈底位置为n-1

B. S1的栈底位置为0，S2的栈底位置为n/2

C. S1的栈底位置为0，S2的栈底位置为n

D. S1的栈底位置为0，S2的栈底位置为1

【解答】 A

【分析】 两栈共享空间首先两个栈是相向增长的，栈底应该分别指向两个栈中的第一个元素的位置，并注意题目表明数组下标是从0开始的。

(14) 为了增加内存空间的利用率和减少溢出的可能性，两个栈共享一片连续的内存空间时，应将两栈的栈底分别设在这片内存空间的两端，这样，当(　　)时才产生上溢。

A. 两个栈的栈顶同时到达栈空间的中心点

B. 其中一个栈的栈顶到达栈空间的中心点

C. 两个栈的栈顶在栈空间的某一位置相遇

D. 两个栈均不空，且一个栈的栈顶到达另一个栈的栈底

【解答】 C

【分析】 两栈共享空间只有当两个栈的栈顶在栈空间的某一位置相遇时，即数组中没有空闲单元时，才会发生溢出。

(15) 两个栈共享一个数组空间的好处是(　　)。

A. 减少存取时间，降低发生上溢的可能性

B. 节省存储空间，降低发生上溢的可能性

C. 减少存取时间，降低发生下溢的可能性

D. 节省存储空间，降低发生下溢的可能性

【解答】 B

【分析】 栈的基本操作(如插入、删除)的时间性能都是 $O(1)$，所以两栈共享空间并不是为了减少存取时间；当栈为空时再执行删除操作会发生下溢，下溢通常作为一个判定条件而无需降低其可能性。

2. 应用题

试题点拨 主要考查元素以同样顺序进栈后出栈的不同情况。

(1) 设有一个栈，元素进栈的次序为 A,B,C,D,E，能否得到如下出栈序列，若能，请写出操作序列，若不能，请说明原因。

① C,E,A,B,D

② C,B,A,D,E

【解答】 ①不能，因为在 C、E 出栈的情况下，A 一定在栈中，而且在 B 的下面，不可能先于 B 出栈。②可以，设 I 为进栈操作，O 为入栈操作，则其操作序列为 $IIIOOOIOIO$。

(2) 在操作序列 push(1)、push(2)、pop、push(5)、push(7)、pop、push(6)之后，栈顶元素和栈底元素分别是什么？(push(k)表示整数 k 入栈，pop 表示栈顶元素出栈。)

【解答】 栈顶元素为 6，栈底元素为 1。其执行过程如图 3-2 所示。

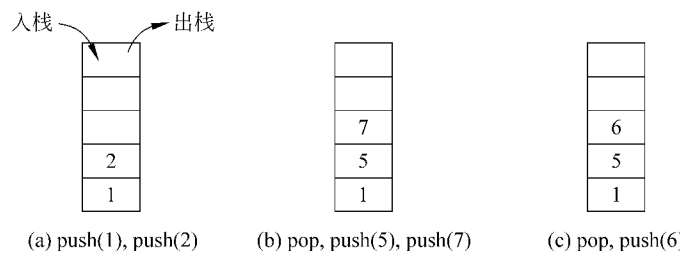

图 3-2 栈的执行过程示意图

(3) 设元素 1，2，3，P，A 依次经过一个栈，进栈次序为 $123PA$，在栈的输出序列中，有哪些序列可作为 C++ 程序设计语言的变量名。

【解答】 可作为 C++ 程序设计语言变量名的出栈序列应该以 P 或 A 开头，共有 5 个，分别是 $PA321$，$P3A21$，$P32A1$，$P321A$，$AP321$。

(4) 如果进栈序列为 A、B、C、D，则可能的出栈序列是什么？

【解答】 共 14 种，分别是：*ABCD*，*ABDC*，*ACBD*，*ACDB*，*ADCB*，*BACD*，*BADC*，*BCAD*，*BCDA*，*BDCA*，*CBAD*，*CBDA*，*CDBA*，*DCBA*。

3. 算法设计题

试题点拨 由于顺序栈和链栈的基本操作的实现非常简单，因此，一般不会单独以算法设计题的形式出现，但树或图的相关内容（例如二叉树的遍历、图的深度优先遍历）用栈作为辅助数据结构，所以，要求能够熟练写出栈的操作语句。

（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，入栈和出栈的操作序列可表示为仅由 *I* 和 *O* 组成的序列，称可以操作的序列为合法序列，否则称为非法序列。

① 下面所示的序列中哪些是合法的？

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

② 通过对①的分析，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）。

【分析】 在入栈出栈序列（即由 *I* 和 *O* 组成的字符串）的任一位置，入栈次数（即 *I* 的个数）都必须大于等于出栈次数（即 *O* 的个数），否则在形式上视作非法序列。由于题目中要求栈的初态和终态都为空，则整个序列的入栈次数必须等于出栈次数，否则在形式上视为非法序列。

【解答】 ① A 和 D 是合法序列，B 和 C 是非法序列。

② 具体算法如下：

出栈序列合法检查 Judge

```
int Judge(char A[])          //判断字符数组 A 中的输入输出序列是否是合法序列
{
   i=0;                      //扫描数组进行处理，i 为数组下标
   countI=countO=0;          //countI 和 countO 分别为 I 和字母 O 的个数
   while(A[i]!='\0')
   {
      if(A[i]=='I') countI++;          //入栈次数增 1
      else {
           countO++;                   //出栈次数增 1
           if(countO>countI) return 0;  //出栈次数大于入栈次数
      }
      i++;
   }
   if(countI!=countO) return 0;
   else return 1;
}
```

（2）下面的算法将一个整数 e 压入堆栈 S，请在空格处填上适当的语句实现该操作。

```
typedef struct {
  int *base;
```

```
  int *top;
  int stacksize;
}SqStack;
int Push(SqStack S, int e)
{  if(    ①    )
   {  S.base=(int *)realloc(S.base,(S.stacksize+1)*sizeof(int));
      if(    ②    )
      {  printf("Not Enough Memory!\n");
         return 0;
      }
      S.top=    ③
      S.stacksize=    ④
   }
    ⑤
   return 1;
}
```

【分析】 可以看出栈 S 采用顺序存储，base 为数组的起始地址，stacksize 为栈中元素个数，top 为栈顶元素所在地址。①应是栈满的判定条件；②应该判断申请空间是否成功；③是调整栈顶指针 top；④是调整栈元素的个数；⑤是在 top 位置插入元素 e。

【解答】 ①S. top－S. base＞＝S. stacksize，②!S. base，③S. base＋S. stacksize，④S. satcksize＋1，⑤*S. top＋＋＝e

3.3 队列

3.3.1 考核知识点

1. 队列的定义(★☆☆☆☆ ◆◇◇◇◇)

队列是只允许在一端进行插入操作，而另一端进行删除操作的线性表。允许插入的一端称为队尾，允许删除的一端称为队头。

【说明】 插入操作也称入队或进队，删除操作也称出队。

2. 队列的操作特性(★★★★☆ ◆◆◇◇◇)

队列的操作具有先进先出的特性。

3. 队列的基本操作(★★★★☆ ◆◆◇◇◇)

- InitQueue(Q)：初始化一个空队列 Q。
- EnQueue(Q,x)：在队列 Q 的队尾插入一个元素 x。
- DeQueue(Q)：删除并返回队列 Q 的队头元素。
- GetQueue(Q)：读取队列 Q 的队头元素但不删除。