第一章

一、选择题

1．以下说法不正确的是 B 。

A．数据可由若干个数据元素构成 B．数据项可由若干个数据元素构成

C．数据元素是数据的基本单位 D．数据项是不可分割的最小标识单位

2．以下属于数据结构中非线性结构的是 D 。

A．栈 B．串 C．队列 D．平衡二叉树

3．以下属于逻辑结构的是 B 。

A．顺序表 B．有序表 C．双链表 D．单链表

4．在计算机中存储数据时，通常不仅要存储各数据元素的值，还要存储 C 。

A．数据的处理方法 B．数据元素的类型

C．数据元素之间的关系 D．数据的存储方法

5．数据结构在计算机内存中的表示是指 A 。

A．数据的存储结构 B．数据结构 C．数据的逻辑结构 D．数据元素之间的关系

6．数据采用链式存储结构时，要求 A 。

A．每个结点占用一片连续的存储区域 B．所有结点占用一片连续的存储区域

C．结点的最后一个数据域是指针类型 D．每个结点有多少个后继就设多少个指针域

7．以下说法中错误的是 B 。

（1）原地算法是指不需要任何额外的辅助空间

（2）在相同的问题规模 n 下，时间复杂度为 $O(n\log_2 n)$的算法在执行时间上总是优于时间复杂度为 $O(n^2)$的算法

（3）时间复杂度通常是指最坏情况下，估计算法执行时间的一个上限

（4）一个算法的时间复杂度与实现算法的语言无关

A（1） B（12） C（14） D（3）

8．下面程序的时间复杂度为 B 。

```
for(i=1,s=0;i<=n;i++)
{
  t=1;
  for(j=1;j<=i;j++)
    t=t*j;
  s=s+t;
}
```

A．$O(n)$ B．$O(n^2)$ C．$O(n^3)$ D．$O(n^4)$

9【2017 统考真题】下列函数的时间复杂度是 B 。

```
int func(int n)
{
  int i=0,sum=0;
  while(sum<n)sum+=++i;
  return i;
}
```

A．$O(\log n)$ B．$O(n^{1/2})$ C．$O(n)$ D．$O(n\log n)$

10．以下函数中时间复杂度最小的是 A 。

A．$T_1(n)=1000\log_2 n$ B．$T_2(n)=n\log_2 n-1000\log_2 n$

C．$T_3(n)=n^2-1000\log_2 n$ D．$T_4(n)=2n\log_2 n-1000\log_2 n$

11【2019 统考真题】设n 是描述问题规模的非负整数，下列程序段的时间复杂度是 B 。

```
x=0;
while(n>=(x+1)*(x+1))
   x=x+1;
```

A．$O(\log n)$ B．$O(n^{1/2})$ C．$O(n)$ D．$O(n^2)$

12．下面程序的时间复杂度为 C 。

```
void fun(int n)
{
  int i=1;
  while(i<=n)
    i=i*2;
}
```

A．$O(n)$ B．$O(n^2)$ C．$O(\log_2 n)$ D．$O(n\log_2 n)$

13．下面程序的时间复杂度为 A 。

```
void fun(int n)
{
  int i=1, k=100;
  while(i<=n)
     k++;
     i+=2;
}
```

A．$O(n)$ B．$O(n^2)$ C．$O(\log_2 n)$ D．$O(n\log_2 n)$

14.【2013 统考真题】已知两个长度分别为m 和n 的升序链表，若将它们合并为长度为$m+n$ 的一个降序链表，则最坏情况下的时间复杂度是 D 。

A．$O(n)$ B．$O(nm)$ C．$O(\min(m,n))$ D．$O(\max(m,n))$

15．算法的时间复杂度为$O(n^2)$，表明该算法的 C。

A．问题规模是n^2 B．执行时间等于n^2

C．执行时间与n^2 成正比 D．问题规模与n^2 成正比

二、判断题

1．数据元素是数据的最小单位（×）

2．数据对象就是一组任意数据元素的集合（×）

3．数据的逻辑结构与数据元素在计算机中如何存储有关（×）

4．如果数据元素值发生改变，则数据的逻辑结构也随之改变（×）

5．逻辑结构相同的数据，可以采用多种不同的存储方法（√）

6．逻辑结构不相同的数据，必须采用多种不同的存储方法（×）

7．数据的逻辑结构是指数据的各数据项之间的逻辑关系（×）

8．算法的优劣与算法描述语言无关，但与所用的计算机有关（×）

9．程序一定是算法（×）

10．算法的可行性是指指令不能有二义性（×）

第二章

一、选择题

1．线性表是具有 *n* 个 C 的有限序列。

A．表元素　B．字符　C．数据元素　D．数据项

2．关于线性表的正确说法是 D 。

A．每个元素都有一个前驱和一个后继元素

B．线性表中至少有一个元素

C．表中元素的排列顺序必须是由小到大或由大到小

D．除首元素和尾元素外，其余元素有且仅有一个直接前驱和一个直接后继元素

3．线性表采用链表存储时，其存放各个元素的单元地址是 D 。

A．必须连续的　B．一定不连续的

C．部分地址必须连续的　D．连续与否均可以

4．线性表的静态链表存储结构与顺序存储结构相比，优点是 C 。

A．所有的操作算法实现简单　B．便于随机存取

C．便于插入和删除　D．便于利用零散的存储器空间

5．设线性表中有 *n* 个元素，以下 A 操作在单链表上实现要比在顺序表上实现效率高。

A．删除指定位置元素的后一个元素

B．在第 *n* 个元素的后面插入一个新元素

C．顺序输出前 *k* 个元素

D．交换第 *i* 个元素和第 *n-i*+1 个元素的值

6．在单链表中，增加一个头结点的目的是 C 。

A．使单链表至少有一个结点　B．标识链表中重要结点的位置

C．方便运算的实现　D．说明单链表是线性表的链式存储结构

7．在一个双链表中，在*p 结点之后插入结点*q 的操作是 B 。

A．q->prior=p;p->next=q;p->next->prior=q;q->next =p->next;

B．q->next=p->next;p ->next->prior=q;p->next=q;q->prior=p;

C．p->next=q;q->prior=p;q->next=p->next;p-> next->prior=q;

D．p->next->prior=q;q->prior=p;p->next=q;q->next=p->next;

8．在一个双链表中，在*p 结点之前插入结点*q 的操作是 D 。

A．p->prior=q;q->next=p;p->prior->next=q; q->prior=p->prior;

B．q->prior=p-> prior;p->prior->next=q;q->next=p;p->prior =q->next;

C．q->next=p;p->next=q;q-> prior->next=q;q->next=p;

D．p->prior->next=q;q->next=p;q->prior=p->prior;p->prior=q;

9．在一个双链表中，删除*p 结点的操作是 A 。

A．p->prior->next=p->next;p->next->prior =p->prior;

B．p->prior=p->prior->prior;p->prior->prior=p;

C．p->next->prior=p;p->next=p->next->next;

D．p->next=p->prior->prior;p->prior=p->prior->prior;

10．在一个双链表中，删除*p 结点之后的一个结点，其时间复杂度为 B 。

A．$O(n\log_2 n)$　B．$O(1)$　C．$O(n)$　D．$O(n^2)$

11．在长度为n 的 A 上，删除第一个元素，其算法的时间复杂度为$O(n)$。

A．只有表头指针的、不带表头结点的循环单链表

B．只有表尾指针的、不带表头结点的循环单链表

C．只有表尾指针的、带表头结点的循环单链表

D．只有表头指针的、带表头结点的循环单链表

12．下面关于线性表的叙述错误的是 D 。

A．线性表采用顺序存储必须占用一片连续的存储空间

B．线性表采用链式存储不必占用一片连续的存储空间

C．线性表采用链式存储便于插入和删除操作的实现

D．线性表采用顺序存储便于插入和删除操作的实现

13．在双链表的两个结点之间插入一个新结点时，需要修改 D 个指针域。

A．1　B．2　C．3　D．4

14．在单链表中，要删除某一指定的结点，必须找到该结点的 C 结点。

A．后继 B．头　C．前驱 D．尾

15．一个单链表长度算法的时间复杂度为 B 。

A．$O(\log_2 n)$　B．$O(n)$　C．$O(1)$　D．$O(n^2)$

二、判断题

1．线性表中每个元素都有一个前驱元素和一个后继元素（×）

2．静态链表既有顺序存储结构的优点，又有动态链表的优点，所以利用它存取第i 个元素的时间与元素个数n 无关（×）

3．静态链表与动态链表在元素的插入、删除方面类似，不需要做元素的移动（√）

4．在循环单链表中，从任一结点出发都可以通过前后移动操作遍历整个循环链表（×）

5．在双链表中，可以从任一结点开始沿着同一方向查找到任何其他结点（×）

三、算法设计题

1.【2018 统考真题】给定一个含 n（n≥1）个整数的数组，请设计一个在时间上尽可能高效的算法，找出数组中未出现的最小正整数。例如，数组{-5,3,2,3}中未出现的最小正整数是 1；数组{1,2,3}中未出现的最小正整数是 4。要求：

（1）给出算法的基本设计思想；

（2）根据设计思想，采用 C 或 C++语言描述算法，关键之处给出注释；

（3）说明你所设计算法的时间复杂度和空间复杂度。

1）要求在时间上尽可能高效，因此采用空间换时间的办法。分配一个用于标记的数组 B[n]，用来记录 A 中是否出现了 1～n 中的正整数，B[0]对应正整数 1，B[n-1]对应正整数 n，初始化 B 中全部为 0。由于 A 中含有 n 个整数，因此可能返回的值是 1～n+1，当 A 中 n 个数恰好为 1～n 时返回 n+1。当数组 A 中出现了小于等于 0 或大于 n 的值时，会导致 1～n 中出现空余位置，返回结果必然在 1～n 中，因此对于 A 中出现了小于等于 0 或大于 n 的值可以不采取任何操作。

经过以上分析可以得出算法流程：从 A[0]开始遍历 A，若 0<A[i]<=n，则令 B[A[i]-1]=1；否则不做操作。对 A 遍历结束后，开始遍历数组 B，若能查找到第一个满足 B[i]==0 的下标 i，返回 i+1 即为结果，此时说明 A 中未出现的最小正整数在 1～n 之间。若 B[i]全部不为 0，返回 i+1（跳出循环时 i=n，i+1 等于 n+1），此时说明 A 中未出现的最小正整数是 n+1。

2）算法实现：

```
int findMissMin(int A[],int n)
{
    int i,*B;                          //标记数组
    B=(int *)malloc(sizeof(int)*n);//分配空间
    memset(B,0,sizeof(int)*n);         //赋初值为 0
    for(i=0;i<n;i++)
        if(A[i]>0&&A[i]<=n)            //若 A[i]的值介于 1~n，则标记数组 B
            B[A[i]-1]=1;
    for(i=0;i<n;i++)                   //扫描数组 B，找到目标值
        if (B[i]==0) break;
    return i+1;                        //返回结果
}
```

3）时间复杂度：遍历 A 一次，遍历 B 一次，两次循环内操作步骤为 $O(1)$量级，因此时间复杂度为 $O(n)$。空间复杂度：额外分配了 B[n]，空间复杂度为 $O(n)$。

2【2019 统考真题】设线性表 $L=(a_1,a_2,a_3,\ldots,a_n\text{-}2,a_n\text{-}1,a_n)$采用带头结点的单链表保存，链表中的结点定义如下。

```
typedef struct node
{
  int data;
  struct node *next;
}NODE;
```

请设计一个空间复杂度为 $O(1)$且时间上尽可能高效的算法，重新排列 L 中的各结点，得到线性表 $L'=(a_1,a_n,a_2,a_n\text{-}1,a_3,a_n\text{-}2,\ldots)$。要求：

（1） 给出算法的基本设计思想；

（2） 根据设计思想，采用 C 或 C++语言描述算法，关键之处给出注释；

（3） 说明你所设计算法的时间复杂度。

1）算法的基本设计思想

先观察 $L=(a_1,a_2,a_3,\cdots,a_{n-2},a_{n-1},a_n)$ 和 $L'=(a_1,a_n,a_2,a_{n-1},a_3,a_{n-2},\cdots)$，发现 L' 是由 L 摘取第一个元素，再摘取倒数第一个元素……依次合并而成的。为了方便链表后半段取元素，需要先将 L 后半段原地逆置［题目要求空间复杂度为 $O(1)$，不能借助栈］，否则每取最后一个结点都需要遍历一次链表。①先找出链表 L 的中间结点，为此设置两个指针 p 和 q，指针 p 每次走一步，指针 q 每次走两步，当指针 q 到达链尾时，指针 p 正好在链表的中间结点；②然后将 L 的后半段结点原地逆置。③从单链表前后两段中依次各取一个结点，按要求重排。

2）算法实现

```
void change_list(NODE*h)
{    NODE *p,*q,*r,*s;
   p=q=h;
   while(q->next!=NULL)                    //寻找中间结点
   {  p=p->next;                           //p 走一步
      q=q->next;
      if(q->next!=NULL) q=q->next;         //q 走两步
   }
   q=p->next;          //p 所指结点为中间结点，q 为后半段链表的首结点
   p->next=NULL;
   while(q!=NULL)      //将链表后半段逆置
   {  r=q->next;
      q->next=p->next;
      p->next=q;
      q=r;
   }
   s=h->next;          //s 指向前半段的第一个数据结点，即插入点
   q=p->next;          //q 指向后半段的第一个数据结点
   p->next=NULL;
   while(q!=NULL)      //将链表后半段的结点插入到指定位置
   {  r=q->next;       //r 指向后半段的下一个结点

      q->next=s->next; //将 q 所指结点插入到 s 所指结点之后
      s->next=q;
      s=q->next;        //s 指向前半段的下一个插入点
      q=r;
   }
}
```

3）第 1 步找中间结点的时间复杂度为 $O(n)$，第 2 步逆置的时间复杂度为 $O(n)$，第 3 步合并链表的时间复杂度为 $O(n)$，所以该算法的时间复杂度为 $O(n)$。

第三章

一、选择题

1．经过以下运算后，x 的值是 A 。

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

A．a　　B．b　　C．1　　D．0

2．经过以下栈运算后，StackEmpty(s)的值是 C 。

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

A．a　　B．b　　C．1　　D．0

3．设一个栈的输入序列为A、B、C、D，则借助一个栈所得的输出序列不可能是 D 。

A．ABCD　　B．DCBA　　C．ACDB　　D．DABC

4．一个栈的进栈序列是abcde，则栈的不可能输出序列是 C 。

A．edcba B．decba C．dceab D．abcde

5. 已知一个栈的进栈序列是 1,2,3,…,n，其输出序列是 $p_1,p_2,\ldots,p_n$，若 $p_1=n$，则 p_i 的值是 C 。

A．i　　B．$n-i$　　C．$n-i+1$　D．不确定

6. 设n 个元素的进栈序列是$p_1,p_2,\ldots,p_n$，其输出序列是1,2,3,…,n，若$p_n=1$，则p_i（$1\le i\le n-1$） 的值是 A。

A．$n-i+1$　　B．$n-i$　　C．i　　D．不确定

7【2010 统考真题】若元素a、b、c、d、e、f 依次进栈，允许进栈、退栈的操作交替进行，但不允许连续3 次退栈工作，则不可能得到的出栈序列是 D 。

A．dcebfa　　B．cbdaef　　C．bcaefd　　D．afedcb

8【2017 统考真题】下列关于栈的叙述中，错误的是 C 。

Ⅰ．采用非递归方式重写递归程序时必须使用栈

Ⅱ．函数调用时，系统要用栈保存必要的信息

Ⅲ．只要确定了入栈次序，即可确定出栈次序

Ⅳ．栈是一种受限的线性表，允许在其两端进行操作

A．仅Ⅰ　　B．仅Ⅰ、Ⅱ、Ⅲ

C．仅Ⅰ、Ⅲ、Ⅳ　D．仅Ⅱ、Ⅲ、Ⅳ

9．设 n 个元素的进栈序列是 $p_1,p_2,\ldots,p_n$，其输出序列是 1,2,3,…,n，若 $p_3=3$，则 p_1 的值是__A__。

A．可能是2　　B．一定是2　　C．不可能是1　　D．以上都不对

10．设有 5 个元素的进栈序列是 a,b,c,d,e，其输出序列是c,e,d,b,a，则该栈的容量至少是__D__。

A．1　　B．2　　C．3　　D．4

11.【2018 统考真题】若栈*S*1 中保存整数，栈*S*2 中保存运算符，函数*F*()依次执行下述各步操作。

（1）从 *S*1 中依次弹出两个操作数 a 和 b。

（2）从 *S* 2 中弹出一个运算符 op。

（3）执行相应的运算 b op a。

（4）将运算结果压入 *S*1 中。

假定*S*1 中的操作数依次是5,8,3,2（2 在栈顶）*S*2 中的运算符依次是*、-、+（+在栈顶）调用3 次*F*()后，*S*1 栈顶保存的值是__B__。

A．-15　B．15　C．-20　D．20

12．判定一个顺序栈*st*（元素个数最多为*MaxSize*）为空的条件为__A__。

A．st.top==-1　　B．st.top!=-1

C．st.top!=MaxSize　　D．st.top==MaxSize

13．判定一个顺序栈*st*（元素个数最多为*MaxSize*）为栈满的条件为__D__。

A．st.top！==-1　　B．st.top=-1

C．st.top!=MaxSize-1　　D．st.top==MaxSize-1

14．表达式a*(b+c)-d 的后缀表达式是__B__。

A．a b c d * + -　　B．a b c + * d -　　C．a b c * + d -　　D．-+ * a b c d

15．若一个栈用数组 data[n]存储，初始栈顶指针 top 为 n，则以下元素 x 进入栈的正确操作是__D__。

A．top++; data[top]=x;　B．data[top]=x;top++;

C．top--; data[top]=x;　　D．data[top]=x;top--;

16．若一个栈用数组 data[n]存储，初始栈顶指针 top 为 0，则以下元素 x 进入栈的正确操作是__A__。

A．top++; data[top]=x;　B．data[top]=x;top++;

C．top--; data[top]=x;　　D．data[top]=x;top--;

17．以下各链表均不带有头结点，其中最不适合用作链栈的链表是 D 。

A．只有表头指针、没有表尾指针的循环双链表

B．只有表尾指针、没有表头指针的循环双链表

C．只有表尾指针、没有表头指针的循环单链表

D．只有表头指针、没有表尾指针的循环单链表

18．循环队列*qu* 的队满条件（front 队首指针指向队首元素的前一位置，rear 队尾指针指向队尾元素）是 C 。

A．(qu.rear+1)%maxsize= =(qu.front+1)%maxsize

B．(qu.rear+1)%maxsize= =qu.front+1

B．(qu.rear+1)%maxsize= =qu.front+1

D．qu.rear==qu.front

19．最适合用作链队列的不带表头结点链表是 D 。

A．带首结点指针和尾结点指针的循环单链表

B．只带尾结点指针的非循环单链表

C．只带首结点指针的非循环单链表

D．只带尾结点指针的循环单链表

20．若用一个大小为 6 的数组来实现循环队列，且当前 rear 和 front 的值分别是 0 和 3，当从队列中删除一个元素，再加入两个元素后，rear 和front 的值分别是 B 。

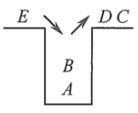
A．1 和5　　B．2 和4　　C．4 和2　　D．5 和1

二、问答题

1．有 5 个元素，其入栈次序为 A,B,C,D,E，在各种可能的出栈次序中，第一个出栈元素为 C 且第二个出栈元素为 D 的出栈序列有哪几个？

1．解答：

CD 出栈后的状态如右图所示。



此时有如下 3 种操作：① *E* 进栈后出栈，出栈序列为 *CDEBA*；② *B* 出栈，*E* 进栈后出栈，出栈序列为 *CDBEA*；③ *B* 出栈，*A* 出栈，*E* 进栈后出栈，出栈序列为 *CDBAE*。

所以，以 *CD* 开头的出栈序列有 *CDEBA*、*CDBEA*、*CDBAE* 三种。

2.【2019 统考真题】请设计一个队列，要求满足：①初始时队列为空；②入队时，允许增加队列占用空间；③出队后，出队元素所占用的空间可重复使用，即整个队列所占用的空间只增不减；④入队操作和出队操作的时间复杂度始终保持为 $O(1)$。试回答下列问题。

（1）该队列是应选择链式存储结构，还是应选择顺序存储结构？

（2）画出队列的初始状态，并给出判断队空和队满的条件。

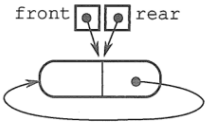
（3）画出第一个元素入队后的队列状态。

（4）给出入队操作和出队操作的基本过程。

1）顺序存储无法满足要求②的队列占用空间随着入队操作而增加。根据要求来分析：要求①容易满足；链式存储方便开辟新空间，要求②容易满足；对于要求③，出队后的结点并不真正释放，用队头指针指向新的队头结点，新元素入队时，有空余结点则无须开辟新空间，赋值到队尾后的第一个空结点即可，然后用队尾指针指向新的队尾结点，这就需要设计成一个首尾相接的循环单链表，类似于循环队列的思想。设置队头、队尾指针后，链式队列的入队操作和出队操作的时间复杂度均为 $O(1)$，要求④可以满足。

因此，采用链式存储结构（两段式单向循环链表），队头指针为 `front`，队尾指针为 `rear`。

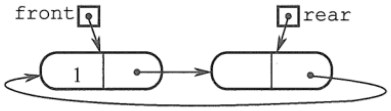
2）该循环链式队列的实现可以参考循环队列，不同之处在于循环链式队列可以方便地增加空间，出队的结点可以循环利用，入队时空间不够也可以动态增加。同样，循环链式队列也要区分队满和队空的情况，这里参考循环队列牺牲一个单元来判断。初始时，创建只有一个空闲结点的循环单链表，头指针 `front` 和尾指针 `rear` 均指向空闲结点，如下图所示。



队空的判定条件：`front==rear`。

队满的判定条件：`front==rear->next`。

3）插入第一个元素后的状态如下图所示。



4）操作的基本过程如下：

入队操作：
若 (front==rear->next)　　//队满 　　则在rear后面插入一个新的空闲结点； 入队元素保存到rear所指结点中；rear=rear->next；返回。
出队操作：
若 (front==rear)　　//队空 　　则出队失败，返回； 取front 所指结点中的元素e；front=front->next；返回e。

三、算法设计题

1. 假设一个算术表达式中包含小括号、方括号和大括号 3 种括号，编写一个算法来判别表达式中的括号是否配对，以字符“\0”作为算术表达式的结束符。

括号匹配是栈的一个典型应用，给出这道题是希望读者好好掌握栈的应用。算法的基本思想是扫描每个字符，遇到花、中、圆的左括号时进栈，遇到花、中、圆的右括号时检查栈顶元素是否为相应的左括号，若是，退栈，否则配对错误。最后栈若不为空也为错误。

```
bool BracketsCheck(char *str){
    InitStack(S);              //初始化栈
     int i=0;
    while(str[i]!='\0'){
       switch(str[i]){
          //左括号入栈
          case '(': Push(S,'('); break;
          case '[': push(S,'['); break;
          case '{': push(S,'{'); break;
          //遇到右括号，检测栈顶
          case ')': Pop(S,e);
             if(e!='(') return false;
          break;
          case ']': Pop(S,e);
             if(e!='[') return false;
          break;
          case '}': Pop(S,e);
             if(e!='{') return false;
          break;
          default:
          break;
       }//switch
       i++;
    }//while
    if(!IsEmpty(S)){
       printf("括号不匹配\n");
       return false;
    }
    else {
       printf("括号匹配\n");
       return true;
    }
}
```

2. 设有两个栈 *s*1、*s*2 都采用顺序栈方式，并共享一个存储区[0,…,*Maxsize*-1]；为了尽量利用空间、减少溢出的可能，这里可采用栈顶相向、迎面增长的存储方式。试设计 *s*1、*s*2 有关入栈和出栈的操作算法。

两个栈共享向量空间，将两个栈的栈底设在向量两端，初始时，s1 栈顶指针为-1，s2 栈顶指针为 maxsize。两个栈顶指针相邻时为栈满。两个栈顶相向、迎面增长，栈顶指针指向栈顶元素。

```
#define  maxsize   100              //两个栈共享顺序存储空间所能达到的最多元素数，
                                    //初始化为 100
#define  elemtp int                 //假设元素类型为整型
typedef  struct{
   elemtp stack[maxsize];           //栈空间
   int   top[2];                    //top 为两个栈顶指针
}stk;
stk   s;                            //s 是如上定义的结构类型变量，为全局变量
```

本题的关键在于，两个栈入栈和退栈时的栈顶指针的计算。s1 栈是通常意义下的栈；而 s2 栈入栈操作时，其栈顶指针左移（减 1），退栈时，栈顶指针右移（加 1）。

此外，对于所有栈的操作，都要注意“入栈判满、出栈判空”的检查。

（1）入栈操作

```
int push(int i, elemtp x){
//入栈操作。i 为栈号，i=0 表示左边的 s1 栈，i=1 表示右边的 s2 栈，x 是入栈元素
//入栈成功返回 1，否则返回 0
   if(i<0||i>1){
      printf("栈号输入不对");
      exit(0);
   }
   if(s.top[1]-s.top[0]==1){
      printf("栈已满\n");
      return  0;
   }
   switch(i){
      case 0: s.stack[++s.top[0]]=x; return  1; break;
      case 1: s.stack[--s.top[1]]=x; return  1;
   }
}
```

（2）退栈操作

```
elemtp pop(int i){
//退栈算法。i 代表栈号，i=0 时为 s1 栈，i=1 时为 s2 栈
//退栈成功返回退栈元素，否则返回-1
   if(i<0||i>1){
      printf("栈号输入错误\n");
      exit(0);
   }
   switch(i){
      case 0:
         if(s.top[0]==-1){
            printf("栈空\n");
            return  -1;
         }
         else
            return  s.stack[s.top[0]--];
      case 1:
         if(s.top[1]==maxsize){
            printf("栈空\n");
            return  -1;
         }
         else
            return  s.stack[s.top[1]++];
   }//switch
}
```

第四章

一、选择题

1．设主字符串为T ="abcacababbc" ，模式字符串为S="ababb"，采用KMP 算法进行模式匹配，需要进行 B 次单个字符间的比较。

A．12　B．13　C．14　D．15

2．设有一个10×10 的对称矩阵$\boldsymbol{M}$，将其上三角部分的元素a_{ij}（1≤i≤j≤10）按行序优先存入大小为55 的 C 语言一维数组N 中，元素a_{36} 在N 中的下标是 A 。

A．22　B．23　C．24　D．25

3．数组A 中的元素a_{ij}（1≤i≤6，1≤j≤5）以列序优先依次存储在起始地址为1000 的连续存储空间中，每个元素占2 个存储单元，则a_{34} 的地址是 C 。

A．1026 B．1038 C．1040 D．1046

4．设有一个10 行10 列的矩阵$\boldsymbol{A}$，采用行序优先存储方式。如果a_{00} 为第一个元素，其存储地址为1000，a_{23} 的存储地址为1069，则存储一个元素需要的单元数是 C 。

A．1　B．2　C．3　D．4

5．下列不能够对数据元素进行随机访问的物理结构是 B 。

A．数组的顺序存储　　B．三元组顺序表

C．三对角矩阵的压缩存储　D．对称矩阵的压缩存储

6．某稀疏矩阵$\boldsymbol{A}$ 采用三元组顺序表作为存储结构，对于矩阵元素的赋值运算Assign(A,e,i,j)，不可能 C （在Assign(A,e,i,j)中，e 是元素的值，i 和j 分别为矩阵$\boldsymbol{A}$ 中元素a_{ij} 的行号和列号，完成将e 赋予a_{ij} 的操作）

A．插入一个新的三元组　　B．修改某个三元组的元素值

C．修改某个三元组的行号或列号　　D．删除一个三元组

7．有一个10 阶的三对角矩阵$\boldsymbol{M}$，其元素a_{ij}（1≤i≤10，1≤j≤10）按行优先次序压缩存入一个大小为28、下标从0 开始的一维数组N 中，N 中的元素值依次为1～28，则$\boldsymbol{M}$ 中元素a_{54} 和元素a_{57} 的值分别是 D 。

A．0, 0　　B．12,15　　C．13,0　　D．12,0

8．对稀疏矩阵进行压缩存储的方法一般有两种，分别为 A 。

A．三元组顺序表和十字链表　　B．对角矩阵和散列表

C．三元组和对称矩阵　　D．散列和十字链表

9．对广义表*G*=((*a*, (( ),*b*)), ((( ),(*c*,*d*)),( )))执行tail(head(head(tail(G))))操作的结果是 D 。

A．( )　B．*c*　C．(*c*,*d*)　D．((*c*,*d*))

10．广义表((*a*,()),(*b*,(*c*)),(()))的深度是 A 。

A．3　B．4　C．5　D．6

二、 填空题

1. 字符串是一种特殊的线性表，其特殊性体现在 数据元素是字符 。

2. 设目标串 S="*xyzabxabacaa*"，模式串 T="*abac*"，使用朴素模式匹配算法，匹配成功需要 12 次单字符的比较。

3. 数组 A[l_1..h_1][l_2..h_2]按行序优先进行存放，元素 A[*i*][*j*]前有（i- l_1）*（h_2- l_2+1）+ *j*- l_2 个元素。

4. 通常多维数组的顺序存储有行序优先 和 列序优先 两种存储次序。

5. 设有 n 阶的对称矩阵 A，按照行的顺序将矩阵下三角中的元素（包括对角线上元素）存放在 n(n+1)个连续的存储单元中，则 A[i][j]与 A[0][0]之间有 j(j-1)/2+i-2 个数据元素，这里 i<j。

6. 设有一个 5 行 5 列的矩阵 A，采用行序优先存储方式，存储全部数据需要 100 个字节的空间。如果 $a_{0,0}$ 为第一个元素，其存储地址为 1000，则 $a_{2,4}$ 的地址为 1056 。

7. 特殊矩阵 指具有相同值的非零元素或零元素的分布有一定规律的矩阵。

8. 设 10*10 的对称矩阵下三角保存 SA[1. . 55]中，其中 A[1][1]保存在 SA[1]中，A[5][3] 保存在 SA[*k*]中，这里 *k* 等于 13 。

9. 广义表 (*a*,(*b*,*c*),*d*) 的表长是 3 。

10．广义表((*a*,*b*),(*c*),(*d*))的表尾是((*c*),(*d*)) 。

三、问答题

1. 设字符串 s="I AM A STUDENT"，t="GOOD"，q="WORKER"。求：

① StrLength(s)、　② StrLength(t)、　③ StrSub (s,8,7)、

④ StrSub (t,2, 1)、　⑤ StrIndex(s, "A")、 ⑥ StrIndex(s,t)、

⑦ StrConcat(StrSub(s,6,2), Concat(t,StrSub (s,7,8)))

⑧ StrReplace(s,"STUDENT",q) 的值。

（这里 StrSub 返回取出的子串）

答案：

① 14 ② 4 ③ ‘STUDENT’ ④ ‘O’ ⑤ 3

⑥ 0 ⑦ ‘I AM A WORKER’ ⑧ ‘A GOOD STUDENT’

2. 简述多维数组和线性表的区别。

多维数组是线性表的一种推广

线性表长度可变，数组长度不能改变

多维数组的元素可以是还是一个数组，而线性表元素只是数据元素

3. 简述广义表和线性表的区别。

广义表是线性表的一种推广，元素既可以是原子，也可以是广义表。

4. 如图 4.25 所示 $n*(2n-1)$的矩阵，以第 n 列为中线对称，即 $a_{i\ n-k}=a_{i\ n+k}$，这里（$1\le i\le n$，$1\le k\le n-1$），现需要该矩阵其压缩存储到一维数组 SA[0…m]中。

（1）试描述压缩存储方案，m 的最小值是什么？

（2）对在三角内的数组元素 a_{ij}，试写出 i，j 应满足的条件；

（3）假定待压缩的数组元素按行序优先保存在 SA[k]中，其中 b 单独保存在 SA[0]中。试写出下标(i，j)到 k 的转换公式。

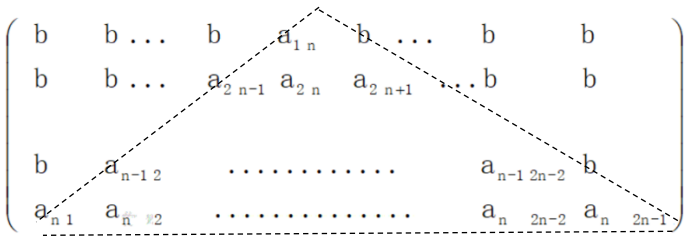


图 4.25 特殊矩阵示例

（1） 该特殊矩阵只需要保存一个 b 值，另外虚线三角形的一半，所以 m 为： n(n+1)/2

（2） n-i<j<n+i 并且 1<=i<=n

（3）两种方案：

(a)如果保存中线以后的，满足 n<=j<n+i 需要保存

K=i（i-1）/2+j-n+1 满足 n<=j<n+i，直接保存

K=i（i-1）/2+n-j+1 满足 n-i<j<n 时，保存的对称元素为（i， 2n-j）

或者以上两种合并成：K=i（i-1）/2+|n-j|+1

K=0　　其它

(b)如果保存中线以前的，满足 n-i<j<=n 需要保存

K=i（i-1）/2+j-n+i　　满足 n-i<j<=n，直接保存

K=i（i-1）/2+n-j+i　　满足 n<=j<n+i 时，保存的对称元素为（i， 2n-j）

或者以上两种合并成：K=i（i-1）/2+|n-j|+i

K=0　　其它

5. 特殊矩阵和稀疏矩阵在压缩存储后，是否能进行随机访问？

特殊矩阵压缩后存储后可以随机访问，稀疏矩阵不行。

6. 试分别画出图 4.26 所示稀疏矩阵的三元组顺序表和十字链表。

$$M=\begin{bmatrix} 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 & 19 \\ 0 & 0 & 8 & 0 & 0 & 5 \end{bmatrix}$$

图 4.26 稀疏矩阵示例

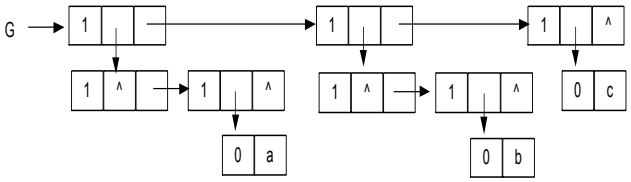


图 4.27　广义表存储结构示意图

答案：

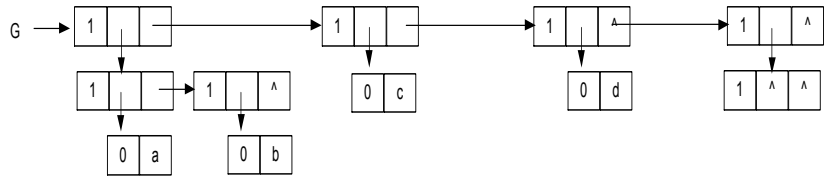
行	列	值
1	3	5
2	2	3
3	2	10
3	6	19
4	3	8
4	6	5
m:4	n:6	t:6

7. 试根据图 4.27 的广义表存储结构示意图给出广义表的定义。

答案：　　G=(((),a),((),b),c)

8. 试画出广义表 A=（(*a*，*b*)，*c*，d，(( ))）的存储结构示意图。

答案：



四、算法设计题

1. 编写算法实现定长存储表示的字符串的替换操作 Replace（&S,T,V）。

算法思想：

（1） 初始时设置 i=0，

（2） 使用 index 定位求 T 在 S 中位置 i 后出现的位置序号，赋值给 i，当 i 不等于-1 时，执行循环体：

① 删除 S 位置 i 开始的长度为 T 的字符串

② S 位置 i 前插入字符串 V；

③ i=i+V 的长度

```
int index(SeqString S,SeqString T,int pos)
{
    for (int i=pos,j=1;i<=S[0];i++,j++)
        if (S[i]!=T[j])
i=i-j+1,j=0;            //i 回溯到本次匹配起点，准备下次匹配
        else if (j==T[0])
 return i-j+1;      //匹配成功，返回本次匹配起始位置
    return 0;                          //匹配失败
}
```

```
void Replace(SeqString &S,SeqString T,SeqString V)
{
    int i=1,j,k;
```

```
    while(i=index(S,T,i))
    {
        if (T[0]>=V[0])
        {
            for(j=1;j<=V[0];j++)
                S[i+j-1]=V[j];
            for(j=i+V[0],k=i+T[0];k<S[0];j++,k++)
                S[j]=S[k];
        } else
        {
            for(j=i+T[0],k=S[0];k>=j;k--)
                S[k+V[0]-T[0]]=S[k];
            for(j=0;j<V[0];j++)
                S[i+j]=V[j+1];
        }
        S[0]-=T[0]-V[0];
        i+=V[0];
    }
}
```

2. 编写算法判断一个堆存储表示的字符串是否回文。

算法思想：

（1） 用 i 和 j 分别指向字符串 S 的第一个和最后一个字符；

（2） 当 i<j 时，重复下面操作：

① 如果 s.ch[i]!=s.ch[j] 返回 NO

② i++，j--；

（3）返回 YES

```
typedef struct {
    unsigned char *ch;
```

```
    int    length;
} HString;
status isPalindrome(HString s)
{
    for(int i=0,j=s.length-1;i<j;i++,j--)
        if (s.ch[i]!=s.ch[j])
            return NO;
    return YES;
}
```

3. 编写算法判断一个结点大小为 1 的链式串是否回文。

算法思想：用递归算法，递归出口为长度小于等于 1 时，返回 YES，否则将单链表拆分成 2 个等长的链表（如果长度为奇数，去掉中间的结点），将其中一个翻转过来，如果 2 个链表相等，返回 YES，否则返回 NO。

```
typedef struct node {
    unsigned char ch;
    struct node *next;
} C_NODE,*SLink;
void reverse(SLink &head)
{
    C_NODE *p, *q;
    if (!head)    return;
    p = head;
    head = NULL;
    while (p)       //利用首插法思想
    {
        q = p->next;
        p->next = head;
        head = p;
```

```
p = q;
```