第 5 章 递归

教材中练习题及参考答案

1. 有以下递归函数：

```
void fun(int n)
{   if (n==1)
        printf("a:%d\n",n);
    else
    {   printf("b:%d\n",n);
        fun(n-1);
        printf("c:%d\n",n);
    }
}
```

分析调用 fun(5)的输出结果。

解：调用递归函数 fun(5)时，先递推到递归出口，然后求值。这里的递归出口语句是 printf("a:%d\n",n)，递推时执行的语句是 printf("b:%d\n",n)，求值时执行的语句是 printf("c:%d\n",n)。调用 fun(5)的输出结果如下：

```
b:5
b:4
b:3
b:2
a:1
c:2
c:3
c:4
c:5
```

2. 已知 $A[0..n-1]$为整数数组，设计一个递归算法求这 n 个元素的平均值。

解：设avg(A，i)返回$A[0..i]$共$i+1$个元素的平均值，则递归模型如下：

avg(A，i)=A[0]　　当$i=0$

avg(A，i)=(avg(A，$i-1$)*i+$A[i]$)/(i+1)　　其他情况

对应的递归算法如下：

```
float avg(int A[],int i)
{   if (i==0)
        return(A[0]);
    else
        return((avg(A,i-1)*i+A[i])/(i+1));
}
```

求 $A[n]$中 n 个元素平均值的调用方式为：avg(A，n−1)。

3. 设计一个算法求正整数 n 的位数。

解：设 $f(n)$为整数 n 的位数，其递归模型如下：

$f(n)=1$　　当 $n<10$ 时

$f(n)=f(n/10)+1$　　其他情况

对应的递归算法如下：

```
int fun(int n)
{    if (n<10)
         return 1;
     else
         return fun(n/10)+1;
}
```

4. 上楼可以一步上一阶，也可以一步上两阶。设计一个递归算法，计算共有多少种不同的走法。

解：设 $f(n)$表示 n 阶楼梯的不同的走法数，显然 $f(1)=1$，$f(2)=2$（两阶有一步一步走和两步走 2 种走法）。$f(n-1)$表示 $n-1$ 阶楼梯的不同的走法数，$f(n-2)$表示 $n-2$ 阶楼梯的不同的走法数，对于 n 阶楼梯，第 1 步上一阶有个 $f(n-1)$种走法，第 1 步上两阶有个 $f(n-2)$种走法，则 $f(n)=f(n-1)+f(n-2)$。对应的递归算法如下：

```
int fun(int n)
{    if (n==1 || n==2)
         return n;
     else
         return fun(n-1)+fun(n-2);
}
```

5. 设计一个递归算法，利用顺序串的基本运算求串 s 的逆串。

解：经分析，求逆串的递归模型如下：

$f(s)=s$　　若$s=\Phi$

$f(s)=\text{Concat}(f(\text{SubStr}(s，2，\text{StrLength}(s)-1))，\text{SubStr}(s，1，1))$　　其他情况

递归思路是：对于 $s=$“$s_1s_2\cdots s_n$”的串，假设“$s_2s_3\cdots s_n$”已求出其逆串即 f(SubStr(s，2，StrLength(s)−1))，再将 s_1（为 SubStr(s，1，1)）单个字符构成的串连接到最后即得到 s 的逆串。对应的递归算法如下：

```
#include "sqstring.cpp"      //顺序串的基本运算算法
SqString invert(SqString s)
{    SqString s1,s2;
     if (StrLength(s)>0)
     {    s1=invert(SubStr(s,2,StrLength(s)-1));
          s2=Concat(s1,SubStr(s,1,1));
     }
     else
          StrCopy(s2,s);
     return s2;
```

```
}
```

6. 设有一个不带表头结点的单链表 *L*，设计一个递归算法 count(*L*)求以 *L* 为首结点指针的单链表的结点个数。

解：对应的递归算法如下：

```
int count(LinkNode *L)
{   if (L==NULL)
        return 0;
    else
        return count(L->next)+1;
}
```

7. 设有一个不带表头结点的单链表 *L*，设计两个递归算法，traverse(*L*)正向输出单链表 *L* 的所有结点值，traverseR(*L*)反向输出单链表 *L* 的所有结点值。

解：对应的递归算法如下：

```
void traverse(LinkNode *L)
{   if (L==NULL) return;
    printf("%d ",L->data);
    traverse(L->next);
}
void traverseR(LinkNode *L)
{   if (L==NULL) return;
    traverseR(L->next);
    printf("%d ",L->data);
}
```

8. 设有一个不带表头结点的单链表 *L*，设计两个递归算法，del(*L*，*x*)删除单链表 *L* 中第一个值为 *x* 的结点，delall(*L*，*x*)删除单链表 *L* 中所有值为 *x* 的结点。

解：对应的递归算法如下：

```
void del(LinkNode *&L,ElemType x)
{   LinkNode *t;
    if (L==NULL) return;
    if (L->data==x)
    {   t=L; L=L->next; free(t);
        return;
    }
    del(L->next,x);
}
void delall(LinkNode *&L,ElemType x)
{   LinkNode *t;
    if (L==NULL) return;
    if (L->data==x)
    {   t=L; L=L->next;
        free(t);
    }
    delall(L->next,x);
}
```

9. 设有一个不带表头结点的单链表 *L*，设计两个递归算法，maxnode(*L*)返回单链表 *L*

中最大结点值，minnodel(*L*)返回单链表 *L* 中最小结点值。

解：对应的递归算法如下：

```
ElemType maxnode(LinkNode *L)
{    ElemType max;
     if (L->next==NULL)
          return L->data;
     max=maxnode(L->next);
     if (max>L->data) return max;
     else return L->data;
}
ElemType minnode(LinkNode *L)
{    ElemType min;
     if (L->next==NULL)
          return L->data;
     min=minnode(L->next);
     if (min>L->data) return L->data;
     else return min;
}
```

10. 设计一个模式匹配算法，其中模板串 *t* 含有通配符'*'，它可以和任意子串匹配。对于目标串 *s*，求其中匹配模板 *t* 的一个子串的位置（'*'不能出现在 *t* 的最开头和末尾）。

解：采用 BF 模式匹配的思路，当是 *s*[*i*]和 *t*[*j*]比较，而 *t*[*j*]为'*'时，取出 *s* 中对应'*'的字符之后的所有字符构成的字符串，即 SubStr(*s*，*i*+2，*s.length*−*i*−1)，其中 *i*+2 是 *s* 中对应'*'字符后面一个字符的逻辑序号。再取出 *t* 中'*'字符后面的所有字符构成的字符串，即 SubStr(*t*，*j*+2，*t.length*−*j*−1)，递归对它们进行匹配，若返回值大于-1，表示匹配成功，返回 *i*。否则返回-1。对应的递归算法如下：

```
#include "sqstring.cpp"     //顺序串的基本运算算法
findpat(SqString s,SqString t)
{    int i=0,j=0,k;
     while (i<s.length && j<t.length)
     {    if (t.data[j]=='*')
          {    k=findpat(SubStr(s,i+2,s.length-i-1),SubStr(t,j+2,t.length-j-1));
               j++;
               if (k>-1)
                    return i-1;
               else
                    return -1;
          }
          else if (s.data[i]==t.data[j])
          {    i++;
               j++;
          }
          else
          {    i=i-j+1;
               j=0;
          }
     }
     if (j>=t.length)
```

```
            return i-1;
        else
            return -1;
    }
```