Ackerman 函数的定义如下：

$$A(m,n)=\begin{cases} n+1 & m=0 \\ A(m-1,1) & m\neq 0, n=0 \\ A(m-1,A(m,n-1)) & m\neq 0, n\neq 0 \end{cases}$$

试写出递归和非递归算法

【解】对于递归算法，直接按照公式即可。其实现见代码清单 3-17。

代码清单 3-17 Ackerman 函数的递归实现

```
1.  int A( int m, int n )
2.  {
3.      if ( m!=0 && n!=0 ) return A( m-1, A( m, n-1 ) );
4.      if ( m!=0 && n==0 ) return A( m-1, 1 );
5.      return n+1;
6.  }
```

对于非递归算法，可以用栈来消除递归。栈中存放着要计算的函数的参数。Ackerman 函数有两个整型参数，我们设置了一个整型栈存放着两个参数。初始时，把要计算的 Ackerman 函数的两个参数依次进栈。然后每次从栈中取出两个元素，即 Ackerman 函数的两个参数，根据不同的情况进行处理，处理结果再存入栈中。当 m 和 n 都不等于 0 时，Ackerman 函数的计算要分两个阶段，先计算 A(m, n-1)，然后再计算 A(m-1, A(m, n-1))。于是将 m-1，m 和 n-1 依次进栈，表示先计算 A(m, n-1)，如果结果为 x，再将 x 进栈，那么下一次计算的就是 A(m-1, x)，这正是递归函数所要求的。当 m 不等于 0，n 等于 0 时，将 m-1 和 1 进栈。当 m 等于 0 时，直接计算出结果 n+1，并进栈。当栈内只剩下一个元素时，就是计算的结果。其实现见代码清单 3-18。

代码清单 3-18 Ackerman 函数的非递归实现

```
7.  int A( int m, int n )
8.  {
9.      seqStack<int> a;    //定义一个整型的顺序栈，存放 Ackerman 函数的参数
10.     a.push( m );        //将参数进栈
11.       a.push( n );
12.
13.     while ( true )
14.     {
15.         // 取栈内头两个元素，第一个是 n，第二个是 m
16.         n = a.pop();
17.             if (a.isEmpty()) return n;  //栈中只有元素，就是计算结果
18.
19.             m = a.pop();
20.
21.         //  按公式分情况讨论
22.         if ( m != 0 && n != 0 )
```

```
23.          {    a.push( m-1 );
24.             a.push( m );
25.             a.push( n-1 );
26.          }
27.          else if ( m!=0 && n==0 )
28.             {   a.push( m-1 );
29.                 a.push( 1 );
30.             }
31.          else a.push( n+1 );
32.     }
33.  }
```