些编译器能够自动完成。但是即使如此，最好还是不要让你的程序带着尾递归。

```
1       /**
2        * Print container from itr.
3        */
4       public static <AnyType> void printList( Iterator<AnyType> itr )
5       {
6           if( !itr.hasNext( ) )
7               return;
8
9           System.out.println( itr.next( ) );
10          printList( itr );
11      }
```

图 3-35 递归的不当使用：打印一个链表

```
1       /**
2        * Print container from itr.
3        */
4       public static <AnyType> void printList( Iterator<AnyType> itr )
5       {
6           while( true )
7           {
8               if( !itr.hasNext( ) )
9                   return;
10
11              System.out.println( itr.next( ) );
12          }
13      }
```

图 3-36 不用递归而打印一个表：编译器可以做到

递归总能够被彻底去除(编译器是在转变成汇编语言时完成递归去除的)，但是这么做是相当冗长乏味的。一般方法是要求使用一个栈，而且仅当你能够把最低限度的最小值放到栈上时这个方法才值得一用。我们将不对此做进一步的详细讨论，只是指出，虽然非递归程序一般说来确实比等价的递归程序要快，但是速度优势的代价却是由于去除递归而使得程序清晰性受到了影响。

3.7 队列 ADT

像栈一样，**队列**(queue)也是表。然而，使用队列时插入在一端进行而删除则在另一端进行。

3.7.1 队列模型

队列的基本操作是 enqueue(入队)，它是在表的末端(叫做队尾(rear))插入一个元素，和 dequeue(出队)，它是删除(并返回)在表的开头(叫做队头(front))的元素。图 3-37 显示一个队列的抽象模型。

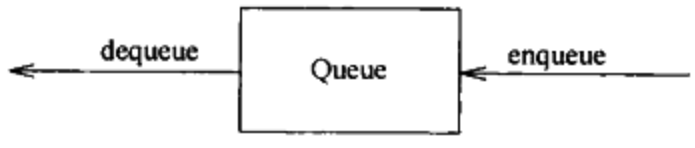

图 3-37 队列模型