在本章实现的顺序表类中，增加连接两个线性表的功能，即将线性表（$a_0, a_1,\ldots, a_{n-1}$）和（$b_0, b_1,\ldots,b_{m-1}$）合并成一个线性表（$a_0, a_1,\ldots, a_{n-1}, b_0, b_1,\ldots,b_{m-1}$）。顺序表类的用户可以使用表达式 C = A + B 将线性表 A 和 B 归并后存入线性表 C。

【解】顺序表类的用户可以用表达式 A+B 来归并两个顺序表，因此归并操作必须用+运算符重载来实现。我们将这个函数重载成友元函数。该函数的实现非常简单。首先定义一个长度为 A、B 长度之和的顺序表 C。接着将 A 表的 data 中的数据复制到 C 表的 data 中，然后再将 B 表的 data 中的数据复制到 C 表的剩余部分。该函数的定义见代码清单 2-9。

代码清单 2-9 合并两个顺序表

```
1. template <class elemType>
2. seqList<elemType> operator+(const seqList<elemType> & a,
3.                         const seqList<elemType> & b)
4. {
5.       seqList<elemType> c(a.length() + b.length());  //c 用于存放归并结果
6.
7.       for (int i = 0; i < a.currentLength; ++i)  //复制 a 的数据
8.             c.data[i] = a.data[i];
9.       for ( int j = 0; j < b.currentLength; ++j) //复制 b 的数据
10.            c.data[i++] = b.data[j];
11.    c.currentLength = i;                    /设置归并后的表长
12.
13.     return c;
14.}
```

如果此时用户直接在程序中用 C=A+B，发现程序会异常终止。程序为什么会异常终止？是因为顺序表类中缺少拷贝构造函数和赋值运算符重载函数。我们分析一下 operator+函数。operator+函数中保存归并结果的是局部对象 c，函数执行结束前会返回 c 的值。对象返回时，系统会用 c 作为参数创建一个临时对象，然后析构 c。由于我们没有定义拷贝构造函数，创建临时对象时系统调用了缺省的拷贝构造函数。缺省的拷贝构造函数是将对象成员互相赋值。于是，临时对象的 currentLength 和 maxSize 的值与 c 的这两个数据成员值相同，临时对象的 data 值也与 c 的 data 值相同，即两个指针指向同一块空间。创建完临时对象，系统就析构对象 c，将 c 的 data 指向的空间释放了。这就造成了临时对象的 data 指向的是一块非法的空间。要使得 c 的值被正确返回，我们需要添加一个如代码清单 2-10 所示的拷贝构造函数，让临时对象有自己的存放线性表元素的空间。

代码清单 2-10 顺序表类的拷贝构造函数

```
1. template <class elemType>
2. seqList(const seqList<elemType> &other)
3. {
4.    currentLength = other.currentLength;
5.    maxSize = other.maxSize;
6.    data = new elemType[maxSize];
7.    for (int i = 0; i < currentLength; ++i)
```

```
8.          data[i] = other.data[i];
9.  }
```

执行 C = A + B 时，要将 A + B 的结果，也就是临时对象的值赋给 C，此时又会出现程序异常终止。原因与缺少拷贝构造函数类似。由于顺序表类没有定义赋值运算符重载函数，在赋值时系统会调用缺省的赋值运算符重载函数。该函数与缺省的拷贝构造函数一样将赋值号右面的对象的值原式原样地赋给了赋值号左边的对象，使得 C 对象与临时对象有相同的 data 值。赋值完成后，系统析构临时对象，将临时对象的 data 指向的空间释放了。于是 C 的 data 又指向了非法的空间。要使得能正确地赋值，我们必须重载赋值运算符。该函数的实现如代码清单 2-11 所示。

代码清单 2-11 顺序表类的赋值运算符重载函数的实现

```
1.  template <class elemType>
2.  seqList<elemType> &operator=(const seqList<elemType> &other)
3.  {
4.     if (this == &other) return *this;  //检查自我赋值
5.     delete [] data;                    //释放当前对象原来指向的空间
6.
7.     //构建一个和 other 一样的对象
8.     currentLength = other.currentLength;
9.     maxSize = other.maxSize;
10.    data = new elemType[maxSize];
11.    for (int i = 0; i < currentLength; ++i)
12.          data[i] = other.data[i];
13.
14.    return *this;
15.}
```