设集合元素存储在一个数组中。试设计一个集合类，使用户能通过运算符*、+、-执行集合的并、交、差运算。设集合元素是无序存储的

【解】集合类的定义见代码清单 7-5。我们将实现并、交、差操作的函数定义为集合类的友元函数。

代码清单 7-5 集合类的定义

```
1.  template<class T>
2.  class Set {
3.          friend Set<T> operator*(const Set<T> &a, const Set<T> &b); // 交
4.          friend Set<T> operator+(const Set<T> &a, const Set<T> &b); // 并
5.          friend Set<T> operator-(const Set<T> &a, const Set<T> &b);  // 差
6.
7.    private:
8.          T *elem;                          // 存放集合元素的数组
9.          int size, volume;                  // 当前集合元素个数与容量
10.         void double_space();                        //扩展空间函数
11.         bool exist(T x)const;                  //查找 x 是否存在函数
12.
13.   public:
14.
15.         Set();
16.         Set(const Set<T> &a);
17.         ~Set() { delete [] elem; }
18.
19.         Set &operator=(const Set<T> &a); // 重载等号
20.         int Get_Size(){return size;}    // 返回集合元素个数
21.         bool insert(T x);              // 插入函数
22.         bool erase(T x);               // 删除函数
23.         void display();                // 输出函数
24. };
```

保存一个集合需要一个动态数组。我们用 elem 表示动态数组，size 是集合中的元素个数，volume 是数组的容量。与顺序表一样，在元素的添加过程中可能引起数组的溢出，于是在集合类中定义了一个扩展数组空间的私有的成员函数 double_space。在实现并、交、差运算时经常需要检查某个元素在集合中是否存在，于是在集合类中有定义了一个私有的成员函数 exist。

集合类用动态数组保存集合的数据，因此需要构造函数、拷贝构造函数和析构函数。由于数据成员中有动态数组，因此也需要赋值运算符重载函数，另外我们还为集合类设计了常用的插入、删除和显示成员函数。这些函数的实现非常简单，我们不再解释。它们的定义见代码清单 7-6。

代码清单 7-6 集合类成员函数的实现

```
1.  template<class T>    // 构造函数
2.  Set<T>::Set(){
3.      size = 0;
4.      volume = 20;
5.      elem = new T[volume];
6.  }
7.
8.  template<class T>   // 复制构造函数
9.  Set<T>::Set(const Set<T> &a){
10.     size = a.size;
11.     volume = a.volume;
12.     elem = new T[volume];
13.     for(int i = 0; i < size ;++i) elem[i] = a.elem[i];
14. }
15.
16. template<class T>  // 扩展空间函数
17. void Set<T>::double_space(){
18.     volume *= 2;
19.     T *tmp = new T[volume];
20.     for(int i = 0 ;i < size; ++i) tmp[i] = elem[i];
21.     delete []elem;
22.     elem = tmp;
23. }
24.
25. template<class T>    // 查找函数
26. bool Set<T>::exist(T x)const{
27.     for(int i = 0 ;i < size; ++i)
28.         if(elem[i] == x) return true;
29.     return false;
30. }
31.
32. template<class T>   // 插入函数
33. bool Set<T>::insert(T x){
34.     if(exist(x)) return false;        // x 已经存在，不能再插入
35.     if(size == volume) double_space();   // 空间不足，扩大空间
36.     elem[size++] = x;
37.     return true;
38. }
39.
40. template<class T>  // 删除函数
41. bool Set<T>::erase(T x){
42.     bool flag = false;
43.     int i;
44.     for(i = 0; i < size; ++i)
```

```
45.             if(elem[i] == x){
46.                 flag = true;
47.                 break;
48.             }
49.         if(flag)
50.             for(; i<size-1; ++i) elem[i] = elem[i+1];
51.         return flag;
52.  }
53.
54.  template<class T>    // 等号重载
55.  Set<T> &Set<T>::operator=(const Set<T> &a){
56.         size = a.size;
57.      delete [] elem;
58.      elem = new T[size];
59.         for(int i = 0 ;i < size; ++i) elem[i] = a.elem[i];
60.         return *this;
61.  }
62.
63.  template<class T>    // 显示函数
64.  void Set<T>::display(){
65.         for(int i = 0 ; i < size; ++i)    cout << elem[i] << " ";
66.         cout<<endl;
67.  }
```

并、交、差函数的实现见代码清单 7-7。交函数将 a 集合与 b 集合的交集存放在 c 集合中。该函数对 a 集合中的每个元素检查它在 b 集合中是否存在。如果存在，则将它添加到 c 集合。最后返回 c 集合。并函数首先将集合 a 的元素复制到 c 集合，再将 b 集合的元素一个个添加到 c。差函数对集合 a 中的每个元素检查它在集合 b 中是否出现。如果没有出现，则添加到集合 c。

代码清单 7-7 并、交、差函数的实现

```
68.  template<class T>                              // 交函数的实现
69.  Set<T> operator*(const Set<T> &a, const Set<T> &b)
70.  {     Set<T> c;
71.         for(int i = 0 ; i < a.size; ++i)          // 检查 a 集合的每个元素
72.             if(b.exist(a.elem[i])) c.insert(a.elem[i]);
73.         return c;
74.  }
75.
76.  template<class T>                              // 并函数的实现
77.  Set<T> operator+(const Set<T> &a, const Set<T> &b)
78.  {     Set<T> c = a;
79.         for(int i = 0 ; i < b.size; ++i) c.insert(b.elem[i]);
80.         return c;
```

```
}

template<class T>                                   // 差函数的实现
Set<T> operator-(const Set<T> &a, const Set<T> &b)
{     Set<T> c;
      for(int i = 0 ; i < a.size; ++i)
            if(!b.exist(a.elem[i]))  c.insert(a.elem[i]);
      return c;
}
```