```
////////////////////////////////////////////////////////////////////////////////
//////////////////////////
//   线性顺序表类的实现
//   Author：Melissa M.CAO
//   Belong：Section of software theory，School of Computer Engineering & Science，
Shanghai University
//   Version： 1.0
////////////////////////////////////////////////////////////////////////////////
//////////////////////////

#include "StdAfx.h"
#include "SeqList.h"

////////////////////////////////////////////////////////////////////////////////
//////////////////////////
//*****************************************************************************
*****************************
//以下部分为线性顺序表类的最基本的操作
////////////////////////////////////////////////////////////////////////////////
//////////////////////////
//*****************************************************************************
*****************************


////////////////////////////////////////////////////////////////////////////////
//////////////////////////
//     构造函数，空表
////////////////////////////////////////////////////////////////////////////////
//////////////////////////
template<class   datatype>   SeqList<datatype>   ::SeqList(int   size):len(0),
maxsize(size)
{
    data = new datatype[maxsize];
    if (data == NULL)
    {
        cout << "动态存储分配失败" << endl;
        exit(1);
    }
}

////////////////////////////////////////////////////////////////////////////////
//////////////////////////
//     构造函数，参数含义：
//                          size：表的最大长度；curlrn：当前长度；a：存放实际值的
```

数组
//   t：申请空间类型，为 1：按照最大长度申请空间；2：按照实际长度申请空间。
```
//////////////////////////////////////////////////////////////////////////////////////////
template<class datatype> SeqList<datatype> ::SeqList(int size, int curlen, datatype *a, int t)
{
    int i;
    maxsize = size;
    len = curlen;
    if (t == 1)
        data = new datatype[maxsize];
    else
        data = new datatype[curlen];
    if (data == NULL)
    {
        cout << "动态存储分配失败" << endl;
        exit(1);
    }

    for (i = 0; i< len; i++)
        data[i] = a[i];
}

//////////////////////////////////////////////////////////////////////////////////////////
//     取元素
//     返回值：位置 i 上的元素；找不到，返回空
//     参数：i--整型，取元素的位置
//////////////////////////////////////////////////////////////////////////////////////////
template <class datatype> datatype SeqList<datatype>::Get(int i)const
{
        if ( i>= 1 && i <= len )
            return data[i-1];
        cout << "位置不合理";
        return NULL;
}

//////////////////////////////////////////////////////////////////////////////////////////
//     定位函数：
//     返回值：所查元素的位置
```

```
//     参数：所查的元素，抽象类型
//////////////////////////////////////////////////////////////////////////////
//////////////////////////
template <class datatype> int SeqList<datatype>::Locate(datatype &item) const
{
    int i;
    for (i = 1; i <= len; i++)
        if (data[i-1] == item)
            break;
    if ( i > len )
        return 0;
    return i;
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////////
//    查找函数：在表中查找元素值 x
//    返回值：查找成功，返回 x 的存储下标；否则，返回-1。
//    参数：i，整型，表示插入位置；item，抽象类型，表示欲插入的值
//////////////////////////////////////////////////////////////////////////////
//////////////////////////
template <class datatype> int SeqList<datatype> :: find(datatype &x) const
{
    int i = 0;
    while( i < len && data[i] != x)
        i++;
    if (i == len)
        return -1;
    else
        return i;
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////////
//    插入函数：在位置 i 插入元素值
//    返回值：插入成功，返回 1；否则，返回 0。
//    参数：i，整型，表示插入位置；item，抽象类型，表示欲插入的值
//////////////////////////////////////////////////////////////////////////////
//////////////////////////
template <class datatype> int SeqList<datatype>::Insert(const datatype &item,int i)
{
      if (IsFull())
      {
          cout << "线性表已满！" << endl;
```

```
        return 0;
    }
    if (i < 0 || i > len)
    {
        cout << "位置不合理！" << endl;
        return 0;
    }
    for (int j = len; j > i; j--)
        data[j] = data[j-1];
    data[i] = item;
    len++;
    return 1;
}

//////////////////////////////////////////////////////////////////////////////
////////////////////////
//    追加函数：在表的最后位置插入元素值
//    返回值：插入成功，返回 1；否则，返回 0。
//    参数：item，抽象类型，表示欲插入的值
//////////////////////////////////////////////////////////////////////////////
////////////////////////
template <class datatype> int SeqList<datatype>::AppendItem(const datatype &item)
{
    if (IsFull())
    {
        cout << "线性表已满！" << endl;
        return 0;
    }
    data[len] = item;
    len++;
    return 1;
}

//////////////////////////////////////////////////////////////////////////////
////////////////////////
//    删除函数：删除位置 i 元素值
//    返回值：   删除入成功，返回删除的元素值；否则，返回 NULL。
//    参数：     i，整型，表示删除位置；
//               增加参数 f，目的是避免 datatype 为 int 时，实参为整数时无法确认删除
的是位置还是元素 item。故增加该参数，
//               为 true 时表示按照位置删除；为 false 时暂时未考虑含义，实现时暂且报
错。
//////////////////////////////////////////////////////////////////////////////
////////////////////////
```

```
template <class datatype> datatype SeqList<datatype>::Delete(const int i, bool f)
{
    if (f == false)
    {
        cout << "error!";
        return NULL;
    }
      if (IsEmpty())
      {
          cout << "线性表为空，无元素可删!! " << endl;
          return NULL;
      }
      if ( i < 1 || i > len )
      {
          cout << "位置不合理! " << endl;
          return NULL;
      }
      datatype t = data[i-1];
      for( int j = i; j < len; j++ )
          data[j-1] = data[j];
      len--;
      return t;
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////////
//    删除函数：删除指定的元素值
//    返回值：删除入成功，返回删除的元素值；否则，返回 NULL。
//    参数：item，抽象类型，表示欲删除的值；
//////////////////////////////////////////////////////////////////////////////
//////////////////////////
template <class datatype> datatype SeqList<datatype>::Delete(datatype &item)
{
     int position = Locate(item);
     return Delete(position, true);

}

//////////////////////////////////////////////////////////////////////////////
//////////////////////////
//     判断线性顺序表是否为空
//////////////////////////////////////////////////////////////////////////////
//////////////////////////
template <class datatype> int SeqList<datatype>::IsEmpty( ) const
```

```
{
      if (len == 0)
          return 1;
      return 0;
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////
//     判断线性顺序表是否满
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template <class datatype> int SeqList<datatype>::IsFull( ) const
{
      if (len == maxlen)
          return 1;
      return 0;
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////
//     清空顺序表
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template <class datatype> void SeqList<datatype> :: Clear( )
{
      len = 0;
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////
//     显示函数
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template <class datatype> void SeqList<datatype> :: Display()
{
    int i;
    for (i = 0; i < len; i++)
        cout << data[i] << "; ";
    cout << endl;
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////
//     显示部分数据的函数
```

```
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template <class datatype> void SeqList<datatype> :: Display(int low, int high)
{
    int i;
    for (i = low; i <= high; i++)
        cout << data[i] << "; ";
    cout << endl;
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////////
//*************************************************************************
*****************************
//以下部分为线性顺序表类的“基本”操作的扩展
//////////////////////////////////////////////////////////////////////////////
//////////////////////////
//*************************************************************************
*****************************

//////////////////////////////////////////////////////////////////////////////
//////////////////////
//    求 x 的前驱位置函数
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template <class datatype> int SeqList<datatype> :: Prior(datatype &x)
{
    int i = Locate(x);
    if (i > 1 && i <= len)
        return i-1;
    else
        if (i == 1)
        {
            cout << x << "是第一个元素，因此返回-1" << endl;
            return -1;
        }
        else
        {
            cout << x << "在线性表中不存在，因此返回-1" << endl;
            return -1;
        }
}

//////////////////////////////////////////////////////////////////////////////
```

```
//////////////////////////
//      求 x 的后继位置函数
////////////////////////////////////////////////////////////////////////////////
//////////////////////////
template <class datatype> int SeqList<datatype> :: Next(datatype &x)
{
    int i = Locate(x);
    if (i < len)
        return i+1;
    else
        if (i == len)
        {
            cout << x << "是最后一个元素，因此返回-1" << endl;
            return -1;
        }
        else
        {
            cout << x << "在线性表中不存在，因此返回-1" << endl;
            return -1;
        }
}

////////////////////////////////////////////////////////////////////////////////
//////////////////////////
//      与表 B 合并，且数据不重复，即求并集。无序表。
//      合并成功，返回 1，否则，返回 0；
//      思考：如果是有序表，对算法有无影响？是否影响算法的时间、空间效率？
////////////////////////////////////////////////////////////////////////////////
//////////////////////////
template <class datatype> int SeqList<datatype> :: Union(SeqList<datatype> &B)
{
    int i, m;
    m = B.Length();
    datatype temp;
    if (len + m > maxsize)
    {
        cout << "合并后长度大于本线性表的最大长度，合并不成功！" << endl;
        return 0;
    }
    //请同学们自己完成

    return 1;
}
```

```
//////////////////////////////////////////////////////////////////////////////
//////////////////////
//      与无序表 B 求交集。
//      合并成功，返回 1，否则，返回 0；
//      思考：如果是有序表，对算法有无影响？是否影响算法的时间、空间效率？
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template <class datatype> int SeqList<datatype> :: InterSection(SeqList<datatype>
&B)
{
    int i;
    int m = len;

    datatype temp, temp1;

    i = 1;

    //请同学们自己完成

    return 1;
}
//////////////////////////////////////////////////////////////////////////////
//////////////////////
//*************************************************************************
*****************************
//以下部分为线性顺序表的习题
//////////////////////////////////////////////////////////////////////////////
//////////////////////
//*************************************************************************
*****************************

//////////////////////////////////////////////////////////////////////////////
//////////////////////
//      删除最大或最小元素,并用最后一个元素代替。表本身无序。有序就没有什么意义了。
//      select == 1, 删除最小值；select == 2, 删除最大值；
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template <class datatype> datatype SeqList<datatype> :: DelMinMaxElm(int select)
{
    datatype temp;

    if (len == 0)
    {
        cout << "空表，无须操作！" << endl;
```

```
        return NULL;
    }
    datatype aim = data[0];
    int i, pos = 1;

    //请同学们自己完成，查找最大值和最小值

    //找到
    if (pos != len)
    {
        //请同学们自己完成
        //提示：如果按书上答案，删除第 pos 个元素，再插入最后一个位置的元素到 pos
位置，会有大量移动，请优化算法
          }
    else //最后一个元素
    {  //请同学们自己完成  }
    return aim;
}

////////////////////////////////////////////////////////////////////////////////
//////////////////////
//    删除函数：删除所有指定的元素值，表本身无序
//    返回值：删除入成功，返回删除的元素值；否则，返回 NULL。
//    参数：item，抽象类型，表示欲删除的值；
//    思考：如果是有序表，对算法有无影响？是否影响算法的时间、空间效率？
////////////////////////////////////////////////////////////////////////////////
//////////////////////
template <class datatype> void SeqList<datatype>::DeleteALL(datatype &item,int
select)
{
    int position, i;

    if (select == 1)
    {
        //算法一
        //请同学们自己完成  }
    else
        if (select == 2)
        {
            //算法二
            //请同学们自己完成
        }
        else
        {
```

```
            cout << "您选择的操作不明确，无法实现！" << endl;
            return;
        }
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////
//     所有重复的元素只保留一个，表本身无序
//     思考：如果是有序表，对算法有无影响？是否影响算法的时间、空间效率？
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template <class datatype> void SeqList<datatype>::DeleteAllRepeat()
{
    if (IsEmpty())
    {
        cout << "Empty Seqence List, Nothing to do!" << endl;
        return;
    }
    int i, j;
    //请同学们自己完成
}

//////////////////////////////////////////////////////////////////////////////
//////////////////////
//  删除元素值在 s 和 t 之间的元素。有序表。
//  思考：如果是无序表，对算法有无影响？是否影响算法的时间、空间效率？
//////////////////////////////////////////////////////////////////////////////
//////////////////////
template <class datatype> void SeqList<datatype>::DeleteBetweenST(datatype s,
datatype t)
{
    if (s > t)
    {
        cout << "给定的范围不合理，无需删除任何元素！" << endl;
        return;
    }

    //请同学们自己完成

}
//////////////////////////////////////////
/*
习题解析中问题：
3-1 （1） 找到,直接替换.如果按书上答案,删除第 pos 个元素,再插入最后一个位置的元素
```

到 pos 位置,会有大量移动。

（3） 后面不要{}。无论是否是要删除的元素，都需要重新赋值 temp = Get（i)。区别仅在于 i 的定位。

*/

```
//////////////////////////////////////////////////////////////////////////////////////////
//(1) 所有重复的元素只保留一个，表本身有序
//////////////////////////////////////////////////////////////////////////////////////////
```