实验题第 1 题

第 1 小题

(1) 设计思路

本题要求从顺序表中删除最小值，并以最后一个元素来填补该最小值的位置。为了达成这一目标，首先需要通过比较找到最小值，然后用最后一个元素覆盖它。在寻找最小值时，使用两个变量分别记录最小值及其所在位置，初始化为顺序表中的第一个元素及位置，然后从顺序表的第 2 个元素开始依次与该最小值比较，记录更小者，从而找到顺序表的最小值和位置。在用最后一个元素覆盖最小值时，需要注意两点：①若该最小值本身就是最后一个元素，则只需直接将之删除即可，无需进行覆盖；②在用最后一个元素覆盖最小值后，相当于将该元素移动到了最小值位置，此时需要将位于原本最后一个位置上的元素删除，以免重复存储。

(2) 源代码

```
//1-1 从顺序表中删除具有最小值的元素（假设顺序表中元素都不相同），并由函数返回被删元素的值，空出的位置由最后一个元素填补。
template <class DataType>
DataType DelMinElem(SqList<DataType> &list){
  DataType min,tmp;//min 标记当前的最小值，tmp 为临时变量
  if (list.GetLength() == 0) return UNDER_FLOW;//如果是空表，返回下溢信息
  list.GetElem(1, min);//初始化 min
  int flag = 1;//定义标记变量 flag 来标记最小值的序号
  for (int i = 2 ; i <= list.GetLength(); i++){//遍历顺序表，如果有更小值，则更新 min，并记录其序号
    list.GetElem(i,tmp);//获取第 i 个数据的值，记录到 tmp 中
    if (tmp < min){//比较 min 与 tmp，进行最小值的更新并标记其序号
      min=tmp;
      flag = i;
    }
  }
  list.GetElem(list.GetLength(),tmp);//获取最后一个元素的值
  list.SetElem(flag,tmp);//用最后一个元素的值替换最终标记的最小值
  list.DeleteElem(list.GetLength(),tmp);//删除最后一个元素
    return min;//返回最小值
}
```

(3) 说明

上述代码可以应用测试程序进行测试并得到正确结果。

第 2 小题

(1)设计思路

本题要求从顺序表中删除与给定值 e 相等的所有元素。为了达成这一目的，需要通过数据比对，寻找数据域与给定值 e 数值相同的元素的位置，并对其进行删除处理。在寻找数据域与给定值 e 数值相同的元素时，需要对顺序表进行遍历，一旦出现目标元素，就利用循环次数 i 与其序号相同的特点，立刻对其进行删除，并获取删除数据信息。整个过程需要一次完整的遍历，以删除所有数据域与给定值 e 相等的元素。需要注意的是，如果选择顺序（正向）遍历，在删除元素后会导致整体序号的前移，而记录循环次数的 i 不会前移，因此需要对此进行处理，方法有二：1. 在每次删除后对计数器 i 进行一次自减操作(i--)，以保证 i 和数据序号的一致性（未测试）；2. 对目标顺序表进行反向遍历，这样删除数据就不会影响未经遍历的数据次序，数据的序号和计数器 i 就不会发生错位（测试并采纳）。相较于方法 1，方法 2 的语法更简单，执行的操作更少，操作更安全。

(2)源代码

```
//1-2 从顺序表中删除与给定值 e 相等的所有元素。
template <class DataType>
Status DelValue(SqList<DataType> &list, int e){
  DataType tmp;
  if (list.LocateElem(e) == 0){//如果是空表，返回下溢信息
    return UNDER_FLOW;
  } else {
    for(int i = list.GetLength(); i >= 1; i--){//反向遍历顺序表，避免了顺序删除后元素相对位置改变造成的数据遗漏
      list.GetElem(i,tmp);
      if (tmp == e){//如果遇到 data 域数值与目标数相同的元素，则进行删除
        list.DeleteElem(i,tmp);
      }
    }
    return SUCCESS;//返回成功信息
  }
}
```

(3)说明

上述代码可以应用测试程序进行测试并得到正确结果。

第 3 小题

(1) 设计思路

本题要求在一个顺序表中如果一个数据值有重复出现，则留下第一个这样的数据值，并删除其他所有重复的元素，使表中所有元素的值均不相同。为了达成这一目的，需要对每个元素进行检测，测试是否有数据域与之相同的元素。如果发现相同元素，则进行删除，但需要留下第一个这样的元素。在检测时，需要使用一个嵌套循环，首先获取每一个元素作为目标数据，再针对这个元素进行一轮比对，并删除重复元素。需要注意的是，题目要求保留第一个这样的元素，最好的解决方法是在获取目标时采用正向遍历，而遍历比对删除时采用反向遍历，这样就可以将比对的范围锁定在 i+1 到 length 之间。这样做始终把第一次出现的重复数据作为目标数据，而又避免了与目标数据本身进行比对删除的操作。

(2) 源代码

```
//1-3 在一个顺序表中如果一个数据值有重复出现，则留下第一个这样的数据值，并删除其他所有重复的元素，使表中所有元素的值均不相同。
template <class DataType>
Status DelRepElem(SqList<DataType> &list){
  if (list.GetLength() == 0) return UNDER_FLOW;//如果为空表，返回下溢信息
  else {
    for (int i = 1; i <= list.GetLength()-1; i++){//顺序遍历，每一次循环获取一个目标数据
      int tmp,opn = 0;
      list.GetElem(i,opn);
      for (int j = list.GetLength(); j >= i+1; j--){//反向遍历，每一次循环进行一次比对
        list.GetElem(j,tmp);
        if (tmp == opn) {
          list.DeleteElem(j,tmp);//如果比对数据与目标数据的数据域相同，说明发生了重复，则删除
        }
      }
    }
    return SUCCESS;//所有循环结束，返回成功信息
  }
}
```

(3)说明

上述代码可以应用测试程序进行测试并得到正确结果。

实验题第 2 题

第 1 小题

(1) 设计思路

本题要求把给定值 e 插入有序表中，使得插入后的表仍然有序。为了达成这一目的，需要将 e 按顺序与所有元素进行比较，插入到符合顺序的位置上。为与所有元素顺序比较，需要进行一次顺序遍历。一旦在遍历比对过程中发现比 e 大的元素，则在这个位置插入 e，并跳出循环，返回成功信息，操作结束。如果没有发现比 e 大的元素，则在表尾执行插入。

(2) 源代码

```
//2.1 把给定值 e 插入有序表中，使得插入后的表仍然有序。
template <class DataType>
Status OrdListInsertElem(SqList<DataType> &list,int e){
    DataType big;//定义一个 big 量，作为比对数据
    for(int i = 1; i <= list.GetLength(); i++){//对顺序表元素进行顺序遍历
        list.GetElem(i,big);//获取比对元素的数据域
        if(big >= e){//如果比对元素的数据域大于等于给定值 e，则进行插入
            list.InsertElem(i,e);//在序号为 i 的元素处插入给定值 e
            return SUCCESS;//返回成功信息，结束循环
        }
    }list.InsertElem(list.GetLength()+1,e);return SUCCESS;//循环结束未返回成功，在表尾插入
}
```

(3)说明

上述代码可以应用测试程序进行测试并得到正确结果。

第 2 小题

(1) 设计思路

本题要求删除值为 e 的所有数据元素。为了达成这一目的，需要找到所有数据域与目标值 e 相等的元素并全部进行删除。为此，对顺序表元素进行遍历，并获取每一个元素的数据域数据，如果与 e 值相等，则进行删除。与 1-2 相同，需要考虑到删除数据导致的序号与计数器错位问题，故采用反向（倒序）遍历来规避此问题。

(2) 源代码

```
//2-2 删除值为 e 的所有数据元素。
template <class DataType>
Status OrdListDeleteElem(SqList<DataType> &list, int e){
    DataType tmp;//定义 tmp 变量，存放比对数据
    for (int i = list.GetLength(); i >= 1; i--){//反向遍历顺序表
        list.GetElem(i,tmp);//获取比对元素的数据域数据
        if(tmp == e) list.DeleteElem(i,tmp);//将 tmp 与给定值 e 比对，若相等则删除 tmp 对应元素
    }
    return SUCCESS;//返回成功信息
}
```

(3)说明

上述代码可以应用测试程序进行测试并得到正确结果。

第 3 小题

(1) 设计思路

本题要求合并两个有序表，得到一个新的有序表。为了达成这一目的，有两种处理方案：1. 先将一个表中的元素全部复制到新表中，再将另一个表中的每个元素有序地插入到新表中；2. 同时将两个表的数据插入到新表中，过程中通过比较数据大小来决定是否切换另一张表来插入。方法 2 相较于方法 1，比较和插入同步进行，节约了一定的时间，总体只需要进行 length1 + length2 步操作，而方法 1 显然每次插入另一张表元素时，都要再次进行遍历，寻找应该插入的位置，整体比方法 2 多了一个循环。但由于没有产生新的嵌套，时间复杂度相等，故此处采用更加符合直觉的方法 1。每一次插入的具体操作同 2-1 类似，只需添加循环即可。

(2) 源代码

```
//2-3 合并两个有序表，得到一个新的有序表。
template <class DataType>
Status OrdListMerge(SqList<DataType> &list1, SqList<DataType> &list2,
SqList<DataType> &list3){
    for(int i = 1; i <= list2.GetLength(); i++){//将 list2 中的元素按顺序一一复制到 list3 中
        DataType tmp;
        list2.GetElem(i,tmp);
        list3.InsertElem(i,tmp);
    }
    for(int j = 1; j <= list1.GetLength(); j++){//遍历 list1，获取其中每一个元素，对其进行顺
序插入
        DataType big,tmp;
        list1.GetElem(j,tmp);//将获取元素存入 tmp 中
        for(int i = 1; i <= list3.GetLength(); i++){//遍历检索 list3 中元素，与 tmp 进行比较
            list3.GetElem(i,big);
            if(big >= tmp){//如果在遍历过程中检索出更大的元素，则在此处插入 tmp 元素
                list3.InsertElem(i,tmp);
                break;//插入完成，跳出检索循环
            } else if(i == list3.GetLength()) {//如果在遍历完后仍未出现更大元素，则在 list3
的最后一位后插入 tmp 元素
        list3.InsertElem(list3.GetLength()+1,tmp);
            }
        }
    }
    return SUCCESS;//循环结束，返回成功信息
}
```

(3)说明

上述代码可以应用测试程序进行测试并得到正确结果。

注: 其中，在 list1 检索插入 list3 时，循环语句 `for(int i = 1; i <= list3.GetLength(); i++){}`可以优化。其起始值 i 可以用一个变量记录下来，或者将 i 定义为成员变量，下一次进入循环时可以利用这个标记，直接从上一次结束循环时的位次开始往后检索，以省去一部分冗余的比较操作。

第 4 小题

(1) 设计思路

本题要求从有序顺序表中删除其值在给定值 s 与 t 之间（s＜t）的所有元素，即删除取值在[s,t]之间的所有元素；如果 s≥t 或顺序表为空，则显示出错信息，并退出运行。为了达成这一目的，需要对顺序表进行遍历，判断每一个元素是否符合题设规则，如果符合，进行删除操作。同上，使用反向遍历可以有效规避顺序删除引起的次序错位。需要注意的是，不要将返回成功信息的操作语句放在循环内，以免提前结束循环，漏删剩余符合条件的元素。

(2) 源代码

```
//2-4 从有序顺序表中删除其值在给定值 s 与 t 之间（s＜t）的所有元素，即删除取值在[s,t]之间的所有元素；
如果 s≥t 或顺序表为空，则显示出错信息，并退出运行。
template <class DataType>
Status OrdListIntervalDelete(SqList<DataType> &list,int s,int t){
    DataType tmp;
    if(s >= t || list.GetLength() == 0) return FAILED;//如果为空表或 s<t，返回错误信息
    else{
        for (int i = list.GetLength(); i >= 1; i--){//涉及多个删除操作，使用反向遍历
            list.GetElem(i,tmp);//获取第 i 位数据，存放到 tmp 中
            if(tmp >=s && tmp <= t){//判断 tmp 是否在闭区间[s,t]中
                list.DeleteElem(i,tmp);//删除第 i 位元素，并将数据存放到 tmp 中
            }
        }
        return SUCCESS;//循环结束，返回成功信息
    }
}
```

(3)说明

上述代码可以应用测试程序进行测试并得到正确结果。

实验题第 3 题

第 1 小题

(1) 设计思路

本题要求在 LinkList 类中增加一个定位函数：在单链表中寻找第 i 个元素节点。若找到，则返回第 i 个元素节点的地址，否则返回空指针。为了达成这一目的，需要对链表中的元素进行遍历，直到次序为 i，获取此节点并返回。因此需要创建一个计数器变量 count，从 1 开始随循环次数自增，同时伴随着指针后移，直到 count 与 i 相等，结束循环，此时 p 正好存放目标节点的地址，返回 p 即可。如果遍历完全部元素，count 都没有与 i 相等而结束循环，则 p 已经后移到了尾指针位置上，存放的地址为 NULL，此时返回 NULL 即可。

(2) 源代码

```
//3-1 定位函数：在单链表中寻找第 i 个元素节点。若找到，则返回第 i 个元素节点的地址，否则返回空指针。
template <class DataType>
Node<DataType>* LinkList<DataType>::LocateAddress(const int i) {
  Node<DataType> *p = head->next;//p 指向第一个元素节点
  int count = 1;//初始化计数器为 1
  while(p != NULL && count != i) {//从第一个元素节点开始依次进行比较元素的值，直到到第 i 个元素或遍历结束(P=NULL)为止
    p = p->next;//移动指针 p 使之指向后继节点
    count++;//计数器进行自增
  }
  return p ? p : NULL;//使用三元运算符判断，如果找到第 i 个元素的地址(p 不为 NULL)，返回 p,否则返回 NULL
}
```

(3)说明

上述代码可以应用测试程序进行测试并得到正确结果。

第 2 小题

(1) 设计思路

本题要求增加一个统计函数：统计单链表中等于给定值 e 的元素个数。为了达成这一目的，需要对链表所有元素进行比较，如果与目标值相等，曾记录一次，最终返回总记录次数。使用一个 for 循环遍历整个链表，利用 LinkList 的类函数 GetElem 逐个获取数据并记录即可。

(2) 源代码

```
//3-2 统计函数：统计单链表中等于给定值 e 的元素个数。
template <class DataType>
int Frequency(LinkList<DataType> &list, DataType &e){
  int count = 0;//初始化计数器为 0
  DataType tmp;
  for(int i = 1; i <= list.GetLength(); i++){//遍历整个链表
    list.GetElem(i,tmp);//将第 i 个元素存放到 tmp 中
    if(tmp == e) count++;//将给定值 e 与 tmp 进行比对，如果相等，计数加 1
  }
  return count;//循环结束，返回计数器 count 的值
}
```

(3)说明

上述代码可以应用测试程序进行测试并得到正确结果。

实验题第 4 题

第 1 小题

(1) 设计思路

本题要求插入函数：把元素值 e 作为数据元素插入表中，使得插入后的表仍然有序。为达成这一目的，需要找到符合要求的位次，并将给定值 e 插入其中。利用 for 循环遍历，获取节点数据与 e 比较，如果大于等于 e 则在此处执行插入。循环结束无返回值，则在链表尾部插入。

(2) 源代码

```
//4-1 插入函数：把元素值 e 作为数据元素插入表中，使得插入后的表仍然有序。
template <class DataType>
Status OrdListInsertElem(LinkList<DataType> &list,DataType e){
    DataType tmp;
    for(int i = 1; i <= list.GetLength(); i++){//循环遍历数组
        list.GetElem(i,tmp);//获取第 i 个元素数据存放到 tmp 中
        if(e <= tmp){//判断给定值 e 是否小于等于 tmp
            list.InsertElem(i,e);//在第 i 个节点处插入给定值 e
            return SUCCESS;//退出循环，返回成功信息
        }
    }list.InsertElem(list.GetLength()+1,e);//循环结束，在链表尾插入给定值 e
}
```

(3)说明

上述代码可以应用测试程序进行测试并得到正确结果。

第 2 小题

(1) 设计思路

本题要求删除数据元素等于 e 的节点。为了达成这一目的，需要判断每个元素是否与 e 相同，相同的执行删除操作。为此，使用一个 for 循环进行反向遍历，获取第 i 个节点数据至 tmp，与 e 比较，相等则进行删除。循环结束，执行完全部删除操作，返回成功信息。

(2) 源代码

```
//4-2 删除数据元素等于 e 的节点。
template <class DataType>
Status OrdListDeleteElem(LinkList<DataType> &list, DataType e){
    DataType tmp;
    for(int i = list.GetLength(); i >= 1; i--){//涉及多个删除，反向遍历
        list.GetElem(i,tmp);//获取第 i 个节点的数据，存放在 tmp 中
        if(tmp == e){//判断 tmp 是否与 e 相等
            list.DeleteElem(i,tmp);//删除第 i 个节点数据并存放在 tmp 中
        }
    }
    return SUCCESS;//循环结束，返回成功信息
}
```

(3)说明

上述代码可以应用测试程序进行测试并得到正确结果。