第一章作业

- 5. 试编写一个递归函数，用来输出*n* 个元素的所有子集。例如，三个元素{*a*, *b*, *c*} 的所有子集是：{ }（空集），{*a*}, {*b*}, {*c*}, {*a*, *b*}, {*a*, *c*}, {*b*, *c*} 和{*a*, *b*, *c*}。

- 基本思想：
- 用一个一维数组x[1:n]表示大小为n的数组的一个子集。
- 如果第j个元素包含在子集中，那么x[j]=1 ，否则x[j]=0；
- 例如 原数组为{a,b},那么他的子集为{0，0}，{0，1}，{1，0}，{1，1}。分别对应子集 {Ø}，{b},{a},{a,b}.

函数实现：

```
#include <iostream.h>
// 定义全局变量， n 在主函数种初始化
int x[20], // 子集向量，假设大小为20
    n;     // 数组元素个数
void Subsets(int i，int n)
{// 输出数组 a[i:n].的所有子集
 // 只有 x[i:n] 在每次递归调用时改变，x[1:i-1],已经被确定为了 0 或1
   if (i == n) {// x[n] 可以是0或1
                // 输出不包含元素 n的子集
                x[n] = 0;
for (int j = 1; j <= n; j++)
   cout << x[j] << " ";
cout << endl;

//输出包含元素 n的子集
x[n] = 1;
for (j = 1; j <= n; j++)
   cout << x[j] << " ";
cout << endl;
return;
}
```

```
// 子集中不包含元素 i 的情况
    x[i] = 0;
// 递归调用产生不含有元素 i 的所有子集
    Subsets(i+1,n);

    //子集中包含元素 i 的情况
    x[i] – 1;
//递归调用产生含有元素 i 的所有子集
    Subsets(i+1,n);
}
```

```
#include <iostream>
int x[100]; // 子集向量，假设大小为100
template <class T>
void Subsets(int i ,int n ,T a[])
{// 输出数组 a[i:n].的所有子集
 // 只有 x[i:n] 在每次递归调用时改变，x[1:i-1],已经被确定为了 0 或1
   if (i == n)  {// x[n] 可以是0或1
                 // 输出不包含元素 n的子集
              x[n] = 0;
              int temp=0;
 for (int j = 1; j <= n; j++)
         { if(x[j]!=0)  {cout << a[j] << " ";temp=1;}          }
if(temp==0)                cout<<"空集";
   cout << endl;

    //输出包含元素 n的子集
    x[n] = 1;
    for (j = 1; j <= n; j++)   { if(x[j]!=0)  cout << a[j] << " ";      }
cout << endl;
    return;
}
```

```
// 子集中不包含元素 i 的情况
    x[i] = 0;
// 递归调用产生不含有元素 i 的所有子集
    Subsets(i+1,n,a);

    //子集中包含元素 i 的情况
    x[i] – 1;
//递归调用产生含有元素 i 的所有子集
    Subsets(i+1,n,a);
}
```

```
void main(void)
{
    cout<<"输入数组大小n=";
    int n;
    cin>>n;
    while(n>100){

          cout<<"请输入一个在1到100内的数"<<endl;
          cin>>n;
    }
    cout<<"输入"<<n<<"个数组元素： ";
    char y[n+1];//实例化
    for(int i=1;i<=n;i++)
       cin>>y[i] ;
    Subsets(1,n,y);
}
```

- 第三章习题
- 2.
- 假设一个线性表的描述满足公式（3-1）
- 1)扩充LinearList类的定义，增加一个函数Reverse，该函数将表中元素的次序变反。反序操作是就地进行的（即在数组element本身的空间内）。注意，在反序操作进行之前，表中第k个元素（如果存在）位于element[k-1]，完成反序之后，该元素位于element[length-k]。
- 2)证明上述函数的复杂性与线性表的长度成线性关系。
- 4)请编写另外一个就地处理的反序函数，它能对LinearList类型的对象进行反序操作。该函数不是LinearList类的成员函数，但它应利用成员函数来产生反序线性表。
- 5)上述函数的时间复杂性是多少？

- (1)
- template<class T>
- LinearList<T>& LinearList<T>::Reverse(){
- for(int i=1;i<length/2;i++)
- Swap(element[i-1],element[length-i]);
- return *this;
- }

- (2)O(length)

- (4)
- template<class T>
- void Reverse(LinearList<T>& L){
- int n=L.length();
- T x;
- for(int i=0;i<n-1;i++)
- {
- L.Delete(n,x);
- L.Insert(i,x);
- }
- }
- (5)O(n^2)

- 7. 设A和B均为LinearList对象。假定A和B中的元素都是按序排列的（如从左至右按递增次序排列）。

  1) 试编写一个成员函数Merge(A, B)，用以创建一个新的有序线性表，该表中包含了A和B的所有元素。

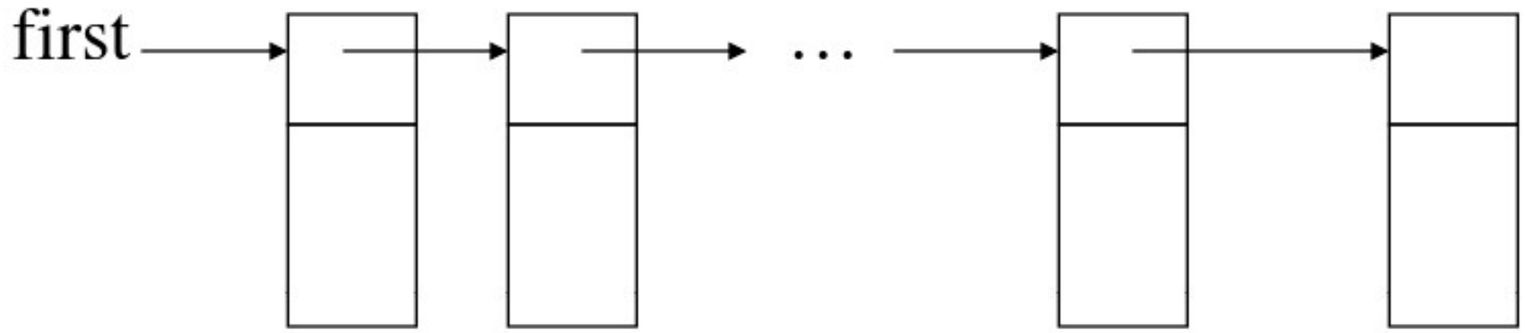
  2) 考察所编写的函数的时间复杂性。

```
template <class T>
LinearList<T>& LinearList<T>::
Merge(const LinearList<T>& A, const LinearList<T>& B)
{// Merge the two sorted lists A and B
    int al = A.Length();
    int bl = B.Length();
    length = al + bl; // length of result
    if (length > MaxSize) throw NoMem();
    // inadequate space for result
    int ca = 0; // cursor for A
    int cb = 0; // cursor for B
    int ct = 0; // cursor for *this
```

```
while ((ca < al) && (cb < bl)) {
    if (A.element[ca] <= B.element[cb])
    element[ct++] = A.element[ca++];
    else element[ct++] = B.element[cb++];
    }
// take care of left overs
if (ca == al) // A is finished
    for (int q = cb; q < bl; q++)
    element[ct++] = B.element[q];
Else  for (int q = ca; q < al; q++)
          element[ct++] = A.element[q];
return *this;
}
```

- 时间复杂性：$\Theta(al+bl)$.

- 27.
- 1)扩充Chain的类定义，增加函数Reverse，用于对x中的元素反序。要求反序操作就地进行，不需要分配任何新的节点。
- 2)函数的时间复杂性是多少？



- 算法思想：
- 设pr指针first 所指节点的前驱; p指针first 所指节点的后继。
- 重复：
- 改变first 所指节点中的link指针指向其前驱
- pr，first，p沿链后移一位置

```
template <class T>
Chain<T>& Chain<T>::Reverse(){
if （first= =0） return *this;
ChainNode<T> *p= first->link; *pr=0;
  while(p){
        first->link=pr;
        pr= first ;
        first=p;
        p=p->link;
   }
first->link=pr;
Return  *this;
}
```

- 28.
- 完成练习27，区别是Reverse不作为Chain的成员函数。要求利用Chain的成员函数来完成反序操作。新的函数将拥有两个参数A和B，A作为输入的链表，B是把A反序后得到的链表。在反序完成时，A变成一个空的链表。

```
template<class T>
Void Reverse(Chain<T>& A,Chain<T>& B){
T x;
 B.first=0;
While (!A.IsEmpty())
    {A.delete(1,x);
     B.Insert(0,x);
    }
}
```

- 31.
- 令A和B都是Chain类型，假定A和B的元素都是按序排列的（即从左至右按递增次序排列），编写一个函数Merge，用以创建一个新的有序线性表C，该表中包含了A和B的所有元素。

```
template<class T>
Merge(Chain<T>  A, Chain<T>  B, Chain<T>&  C){
        ChainNode<T>* PA=A.first;
        ChainNode<T>* PB=B.first;

        int i=0;
        while(PA&&PB){
                if(PA->data<=PB->data){
                        C.Insert(i,PA->data);
                        PA=PA->link;
                }
                else{
                        C.Insert(i,PB->data);
                        PB=PB->link;
                }
                i++;
        }
```

```
while(PA){
        C.Insert(i,PA->data);
        PA=PA->link;
        i++;
}
while(PB){
  C.Insert(i,PB->data);
        PB=PB->link;
        i++;
    }

}
```

- 32
- 重做练习31，要求函数是Chain的一个成员函数，并使用两个输入链表中的物理节点来建立新链表C，在Merge执行完之后，两个输入链表均变成空表。

```
template<class T>
Chain<T>& Chain<T>::Merge(Chain<T>& A, Chain<T>& B){
        ChainNode<T>* pa=A.first;
        ChainNode<T>* pb=B.first;
        ChainNode<T>* pc=0;    //pc指向当前新链表的尾结点
         first=0;
        while(pa&&pb){  //q指向当前两链表中的较小元素，并从链表中删除
                if(pa->data<=pb->data){
                        q=pa;
                        pa=pa->link;
                  }
                else{
                         q=pb;
                        pb=pb->link;
                   }
                if  (pc==0)  { first=q;  pc=q;}  //第一个结点需设置first
                    else  {pc ->link=q;  pc=q;}
                }
    if (pa)   pc->link=pa;      //将剩余链表链入新链表
           else  pc->link=pb;
     A.first=0;
     B.first=0;
     return  *this;
  }
```