实习报告示例:1.3题　集合的并、交和差运算

实习报告

题目:编制一个演示集合的并、交和差运算的程序

班级:计算机95(1) 姓名:丁一　　学号:954211 完成日期:1997.9.14

一、需求分析

1. 本演示程序中,集合的元素限定为小写字母字符['a'..'z'],集合的大小$n<27$。集合输入的形式为一个以"回车符"为结束标志的字符串,串中字符顺序不限,且允许出现重复字符或非法字符,程序应能自动滤去。输出的运算结果字符串中将不含重复字符或非法字符。

2. 演示程序以用户和计算机的对话方式执行,即在计算机终端上显示"提示信息"之后,由用户在键盘上输入演示程序中规定的运算命令;相应的输入数据(滤去输入中的非法字符)和运算结果显示在其后。

3. 程序执行的命令包括:

1) 构造集合1; 2) 构造集合2; 3) 求并集; 4) 求交集; 5) 求差集;6)结束。

"构造集合1"和"构造集合2"时,需以字符串的形式键入集合元素。

4. 测试数据

(1) Set1 = "magazine", Set2 = "paper",

Set1∪Set2 = "aegimnprz",Set1∩Set2 = "ae", Set1−Set2 = "gimnz";

(2) Set1 = "012oper4a6tion89", Set2 = "error data",

Set1∪Set2 = "adeinoprt", Set1∩Set2 = "aeort", Set1−Set2 = "inp"。

二、概要设计

为实现上述程序功能,应以有序链表表示集合。为此,需要两个抽象数据类型:有序表和集合。

1. 有序表的抽象数据类型定义为:

ADT OrderedList {

数据对象:$D=\{ a_i \mid a_i \in CharSet,\ i=1,2,\ldots,n,\ n\geqslant 0 \}$

数据关系:$R1=\{ <a_{i-1},a_i> \mid a_{i-1},a_i \in D,\ a_{i-1}<a_i,\ i=2,\ldots,n \}$

基本操作:

InitList(&L)

操作结果:构造一个空的有序表L。

DestroyList(&L)

初始条件:有序表L已存在。

操作结果：销毁有序表 L。

ListLength(L)

初始条件：有序表 L 已存在。

操作结果：返回有序表 L 的长度。

ListEmpty(L)

初始条件：有序表 L 已存在。

操作结果：若有序表 L 为空表，则返回 True，否则返回 False 。

GetElem(L, pos)

初始条件：有序表 L 已存在。

操作结果：若 1 ≤pos≤Length(L)，则返回表中第 pos 个数据元素。

LocateElem(L, e, &q)

初始条件：有序表 L 已存在。

操作结果：若有序表 L 中存在元素 e，则 q 指示 L 中第一个值为 e 的元素的位置，并返回函数值 TRUE；否则 q 指示第一个大于 e 的元素的前驱的位置，并返回函数值 FALSE。

Append(&L, e)

初始条件：有序表 L 已存在。

操作结果：在有序表 L 的末尾插入元素 e。

InsertAfter(&L, q, e)

初始条件：有序表 L 已存在，q 指示 L 中一个元素。

操作结果：在有序表 L 中 q 指示的元素之后插入元素 e。

ListTraverse(q, visit())

初始条件：有序表 L 已存在，q 指示 L 中一个元素。

操作结果：依次对 L 中 q 指示的元素开始的每个元素调用函数 visit()。

} **ADT** OrderedList

2. 集合的抽象数据类型定义为：

ADT Set {

数据对象：D={ a_i | a_i 为小写英文字母且互不相同，i=1,2,...,n，0≤n≤26 }

数据关系：R1={}

基本操作：

CreateSet(&T, Str)

初始条件：Str 为字符串。

操作结果：生成一个由 Str 中小写字母构成的集合 T。

DestroySet(&T)

初始条件：集合 T 已存在。

操作结果：销毁集合 T 的结构。

Union(&T, S1, S2)

初始条件：集合 S1 和 S2 存在。

操作结果：生成一个由 S1 和 S2 的并集构成的集合 T。

Intersection(&T，S1，S2)

初始条件：集合 S1 和 S2 存在。

操作结果：生成一个由 S1 和 S2 的交集构成的集合 T。

Difference(&T，S1，S2)

初始条件：集合 S1 和 S2 存在。

操作结果：生成一个由 S1 和 S2 的差集构成的集合 T。

PrintSet(T)

初始条件：集合 T 已存在。

操作结果：按字母次序顺序显示集合 T 的全部元素。

}**ADT** Set

3. 本程序包含四个模块：

1）主程序模块：

```
void main() {
    初始化;
    do {
        接受命令;
        处理命令;
    } while ("命令"="退出");
}
```

2）集合单元模块——实现集合的抽象数据类型；

3）有序表单元模块——实现有序表的抽象数据类型；

4）结点结构单元模块——定义有序表的结点结构。

各模块之间的调用关系如下：

三、详细设计

1. 元素类型、结点类型和指针类型

```
typedef  char  ElemType ;                    // 元素类型
typedef  struct  NodeType  {
            ElemType  data;
```

```
        NodeType    * next;
    } NodeType, * LinkType;     //  结点类型，指针类型

status MakeNode( LinkType &p,  ElemType e)
{
  // 分配由 p 指向的数据元素为 e、后继为"空"的结点，并返回 TRUE，
  // 若分配失败，则返回 FALSE
  p = (LinkType)malloc(sizeof(NodeType));
  if (!p) return FALSE;
  p-> data = e;  p->next = NULL;  return TRUE;
}
void FreeNode(LinkType &p )
{   // 释放 p 所指结点
}
LinkType Copy ( LinkType p )
{
  // 复制生成和指针 p 所指结点有同值元素的新结点并返回，
  // 若分配空间失败，则返回空指针。新结点的指针域为 NULL
  s = (LinkType)malloc(sizeof(NodeType));
  if (!s) return NULL;
  s-> data = p-> data;  s-> next = NULL;  return s;
}
ElemType Elem( LinkType p )
{
  // 若指针 p!=NULL，则返回 p 所指结点的数据元素，否则返回'#'
}
LinkType SuccNode( LinkType p )
{
  // 若指针 p!=NULL，则返回指向 p 所指结点的后继元素的指针，
  // 否则返回 NULL
}
```

2. 根据有序表的基本操作的特点，有序表采用有序链表实现。链表设头、尾两个指针和表长数据域，并附设头结点，头结点的数据域没有实在意义。

```
typedef  struct {
           LinkType  head, tail;     // 分别指向线性链表的头结点和尾结点
           int       size;           // 指示链表当前的长度
         } OrderedList;              // 有序链表类型
```

有序链表的基本操作设置如下：

```
bool InitList( OrderedList &L );
  // 构造一个带头结点的空的有序链表 L ,并返回 TRUE;
  // 若分配空间失败,则令 L.head 为 NULL, 并返回 FALSE
void DestroyList( OrderedList &L );
  // 销毁有序链表 L
bool ListEmpty( OrderedList L );
  // 若 L 不存在或为"空表",则返回 TRUE, 否则返回 FALSE
int ListLength( OrderedList L );
  // 返回链表的长度
LinkType GetElemPos( OrderedList L,  int pos );
  // 若 L 存在且 0<pos<L.size+1, 则返回指向第 pos 个元素的指针,
  // 否则返回 NULL
bool LocateElem( OrderedList L,  ElemType e,  LinkType &q );
  // 若有序链表 L 存在且表中存在元素 e,则 q 指示 L 中第一个值为 e 的
  // 结点的位置,并返回 TRUE;否则 q 指示第一个大于 e 的元素的前驱的
  // 位置,并返回 FALSE
void Append( OrderedList &L,  LinkType s );
  // 在已存在的有序链表 L 的末尾插入指针 s 所指结点
void InsertAfter(OrderList &L,  LinkType q,  LinkType s );
  // 在已存在的有序链表 L 中 q 所指示的结点之后插入指针 s 所指结点
void ListTraverse ( LinkType p,  status ( *visit)(LinkType q) );
  // 从 p(p!=NULL)指示的结点开始,依次对每个结点调用函数 visit
```

其中部分操作的伪码算法如下：

```
BOOL InitList( OrderedList &L )
{
   if (MakeNode(head, ' ')) {   // 头结点的虚设元素为空格符' '
       L.tail = L.head;   L.size = 0;   return TRUE;
        }
   else {   L.head = NULL;   return FALSE; }
} //InitList

void DestroyList( OrderedList &L )
{
     p = L.head;
     while ( p ) { q = p;   p = SuccNode(p);   FreeNode(q); }
     L.head = L.tail = NULL;
```

```
} //DestroyList

LinkType GetElemPos( OrderedList L,  int pos )
{
   if ( ! L.head || pos<1 || pos>L.size )  return NULL;
   else if ( pos == L.size ) return L.tail;
   else {
      p = L.head->next;  k = 1;
      while ( p && k<pos) { p = SuccNode(p);  k++; }
      return p;
   }
} //GetElemPos

status LocateElem( OrderedList L,  ElemType e,  LinkType &p )
{
  if ( L.head ) {
     pre = L.head;  p = pre->next;
                    //pre 指向 *p 的前驱，p 指向第一个元素结点
     while ( p && p->data<e) { pre = p;  p = SuccNode(p); }
     if ( p && p->data == e )  return TRUE;
     else { p = pre;  return FALSE; }
   }
  else  return FALSE;
} //LocateElem
void Append( OrderedList &L,  LinkType s )
{
  if ( L.head && s )  {
     if ( L.tail != L.head )  L.tail->next = s;
     else  L.head->next = s;
     L.tail = s;  L.size++;
   }
} //Append

void InsertAfter( OrderList &L,  LinkType q,  LinkType s )
{
   if ( L.head && q && s ) {
      s->next = q->next;  q->next = s;
      if ( L.tail == q ) L.tail = s;
```

```
            L. size++;
        }
    } //InsertAfter

void ListTraverse( LinkType p,  status ( *visit)(LinkType q) )
{
    while (p) { visit(p);  p = SuccNode(p); }
} //ListTraverse
```

3. 集合 Set 利用有序链表类型 OrderedList 来实现，定义为有序集 OrderedSet：

```
typedef  OrderedList  OrderedSet;
```

集合类型的基本操作的类 C 伪码描述如下：

```
void CreateSet( OrderedSet &T,  char *s )
{
  // 生成由串 s 中小写字母构成的集合 T,IsLower 是小写字母判别函数
  if ( InitList(T) )   // 构造空集 T
    for ( i = 1;  i<=length(s);  i++ )
      if ( islower(s[i]) && ! LocateElem(T, s[i], p) )
        // 过滤重复元素并按字母次序大小插入
        if ( MakeNode(q, s[i]) ) InsertAfter(T, p, q);
} // CreateSet

void DestroySet( OrderedSet &T )
{
  // 销毁集合 T 的结构
  DestroyList( T );
} // DestroyList

void Union( OrderedSet &T,  OrderedSet S1,  OrderedSet S2 )
{
  // 求已建成的集合 S1 和 S2 的并集 T,即:S1.head!=NULL 且 S2.head!=NULL
  if ( InitList(T) ) {
    p1 = GetElemPos(S1, 1);  p2 = GetElemPos(S2, 1);
    while ( p1 && p2 ) {
      c1 = Elem(p1);  c2 = Elem(p2);
      if ( c1<=c2 ) {
        Append(T, Copy(p1));  p1 = SuccNode(p1);
        if ( c1== c2 )  p2 = SuccNode(p2);
```

```
    }
    else { Append(T, Copy(p2));  p2 = SuccNode(p2); }
}
while ( p1 )
  { Append(T, Copy(p1));  p1 = SuccNode(p1); }
while ( p2 )
  { Append(T, Copy(p2));  p2 = SuccNode(p2); }
} // Union

void Intersection( OrderedSet &T,  OrderedSet S1,  OrderedSet S2)
{
  // 求集合 S1 和 S2 的交集 T
  if ( ! InitList(T) )  T.head = NULL;
  else {
    p1 = GetElemPos(S1, 1);  p2 = GetElemPos(S2, 1);
    while (p1 && p2 ) {
      c1 = Elem(p1);  c2 = Elem(p2);
      if ( c1<c2 ) p1 = SuccNode(p1);
      else if ( c1>c2 ) p2 = SuccNode(p2);
      else  {  // c1== c2
        Append(T, Copy(p1));
        p1 = SuccNode(p1);  p2 = SuccNode(p2);
      } // else
    } // while
  } // else
} // Intersection

void Difference( OrderedSet &T,  OrderedSet S1,  OrderedSet S2 )
{
  // 求集合 S1 和 S2 的差集 T
  if ( ! InitList(T) )  T.head = NULL;
  else {
    p1 = GetElemPos(S1, 1);  p2 = GetElemPos(S2, 1);
    while ( p1 && p2 ) {
      c1 = Elem(p1);  c2 = Elem(p2);
      if ( c1<c2 )  { Append(T, Copy(p1));  p1 = SuccNode(p1); }
      else if ( c1>c2 )  p2 = SuccNode(p2);
```

```
        else   // c1 == c2
            { p1 = SuccNode(p1);   p2 = SuccNode(p2); }
      } // while
      while ( p1 )
        { Append(T, Copy(p1));   p1 = SuccNode(p1); }
    } // else
} // Difference

void WriteSetElem( LinkType p )
{
    // 显示集合的一个元素
    printf(',');   WriteElem(Elem(p));
} //WriteSetElem

void PrintSet( OrderedSet T )
{
    // 显示集合的全部元素
    p = GetElemPos(T, 1);
    printf('[');
    if ( p )  { WriteElem(Elem(p));   p = SuccNode(p); }
    ListTraverse(p, WriteSetElem);
    printf(']');
} //PrintSet
```

4. 主函数和其他函数的伪码算法

```
void main( )
{
    // 主函数
    Initialization( );    // 初始化
    do {
        ReadCommand(cmd);   // 读入一个操作命令符
        Interpret(cmd);         // 解释执行操作命令符
    } while ( cmd != 'q' && cmd != 'Q' );
} // main

void Initialization( )
{
    // 系统初始化
```

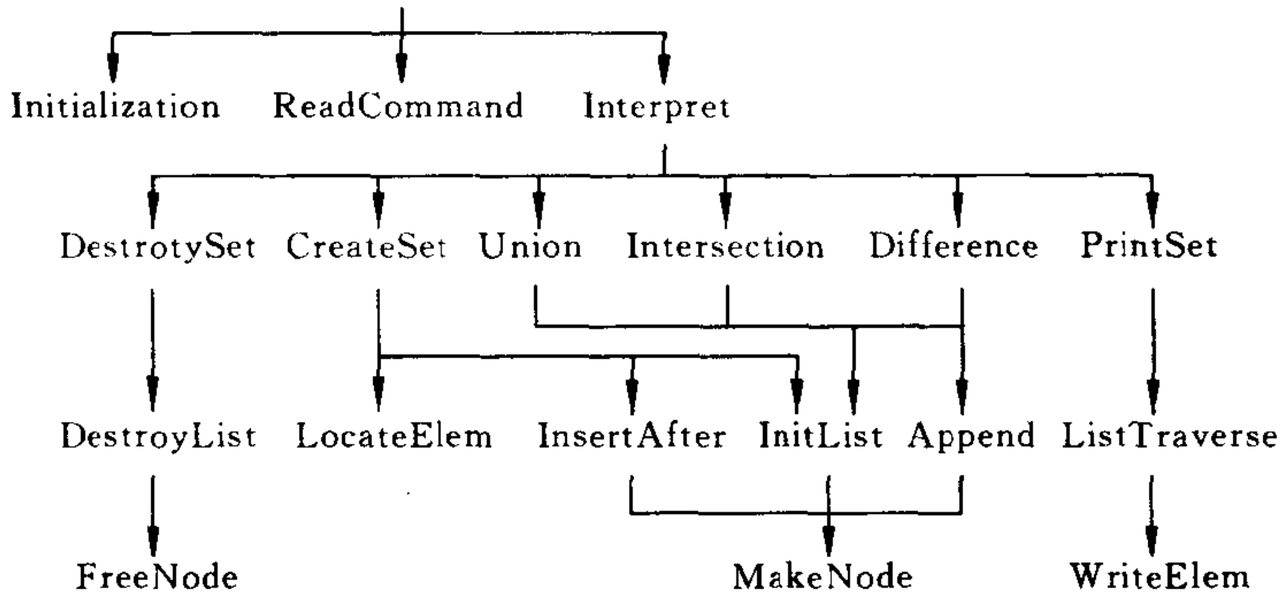
```
    clrscr( );  // 清屏
    在屏幕上方显示操作命令清单:MakeSet1--1  MakeSet2--2  Union--u
                          Intersaction--i  Difference--d  Quit--q ;
    在屏幕下方显示操作命令提示框;
    CreateSet(Set1,"");      PrintSet(Set1);  // 构造并显示空集 Set1
    CreateSet(Set2,"");      PrintSet(Set1);  // 构造并显示空集 Set2
} //Initialization

void ReadCommand( char cmd )
{
    // 读入操作命令符
    显示键入操作命令符的提示信息;
    do { cmd = getche( ); }
    while ( cmd ∉ ['1','2','u','U','i','I','d','D','q','Q'] ) );
}

void Interpret( char cmd )
{
    // 解释执行操作命令 cmd
    switch ( cmd ) {
    case '1': 显示以串的形式键入集合元素的提示信息;
              scanf(v);  // 读入集合元素到串变量 v
              CreateSet(Set1, v);  PrintSet(Set1); // 构造并显示有序集 Set1
              break;
    case '2': 显示以串的形式键入集合元素的提示信息;
              scanf(v);  // 读入集合元素到串变量 v
              CreateSet(Set2, v);  PrintSet(Set2); // 构造并显示有序集 Set2
              break;
    case 'u','U': Union(Set3, Set1, Set2);  //求有序集 Set1 和 Set2 的并集 Set3
                  PrintSet(Set3);           // 显示并集 Set3
                  DestroyList(Set3);        // 销毁并集 Set3
                  break;
    case 'i','I': Intersaction(Set3, Set1, Set2); //求有序集 Set1 和 Set2 的交集
                  Set3
                  PrintSet(Set3);
                  DestroyList(Set3);
                  break;
    case 'd','D': Difference(Set3, Set1, Set2);  //求集合 Set1 和 Set2 的差集
```

```
                    Set3
                    PrintSet(Set3);
                    DestroyList(Set3);
    }
} //Interpret
```

5. 函数的调用关系图反映了演示程序的层次结构：



四、调试分析

1. 由于对集合的三种运算的算法推敲不足，在有序链表类型的早期版本未设置尾指针和 Append 操作，导致算法低效。

2. 刚开始时曾忽略了一些变量参数的标识“&”，使调试程序时费时不少。今后应重视确定参数的变量和赋值属性的区分和标识。

3. 本程序的模块划分比较合理，且尽可能将指针的操作封装在结点和链表的两个模块中，致使集合模块的调试比较顺利。反之，如此划分的模块并非完全合理，因为在实现集合操作的编码中仍然需要判别指针是否为空。按理，两个链表的并、交和差的操作也应封装在链表的模块中，而在集合的模块中，只要进行相应的应用即可。

4. 算法的时空分析

1）由于有序表采用带头结点的有序单链表，并增设尾指针和表的长度两个标识，各种操作的算法时间复杂度比较合理。InitList，ListEmpty，Listlength，Append 和 InsertAfter 以及确定链表中第一个结点和之后一个结点的位置都是 $O(1)$的，DestroyList，LocateElem 和 TraverseList 及确定链表中间结点的位置等则是 $O(n)$的，n 为链表长度。

2）基于有序链表实现的有序集的各种运算和操作的时间复杂度分析如下：

构造有序集算法 CreateSet 读入 n 个元素，逐个用 LocateElem 判定不在当前集合中及确定插入位置后，才用 InsertAfter 插入到有序集中，所以时间复杂度是 $O(n^2)$。

求并集算法 Union 利用集合的“有序性”将两个集合的 $m+n$ 个元素不重复地依次利用 Append 插入到当前并集的末尾，故可在 $O(m+n)$时间内完成。

可对求交集算法 Intersection 和求差集算法 Difference 作类似地分析，它们也是 $O(m+n)$。

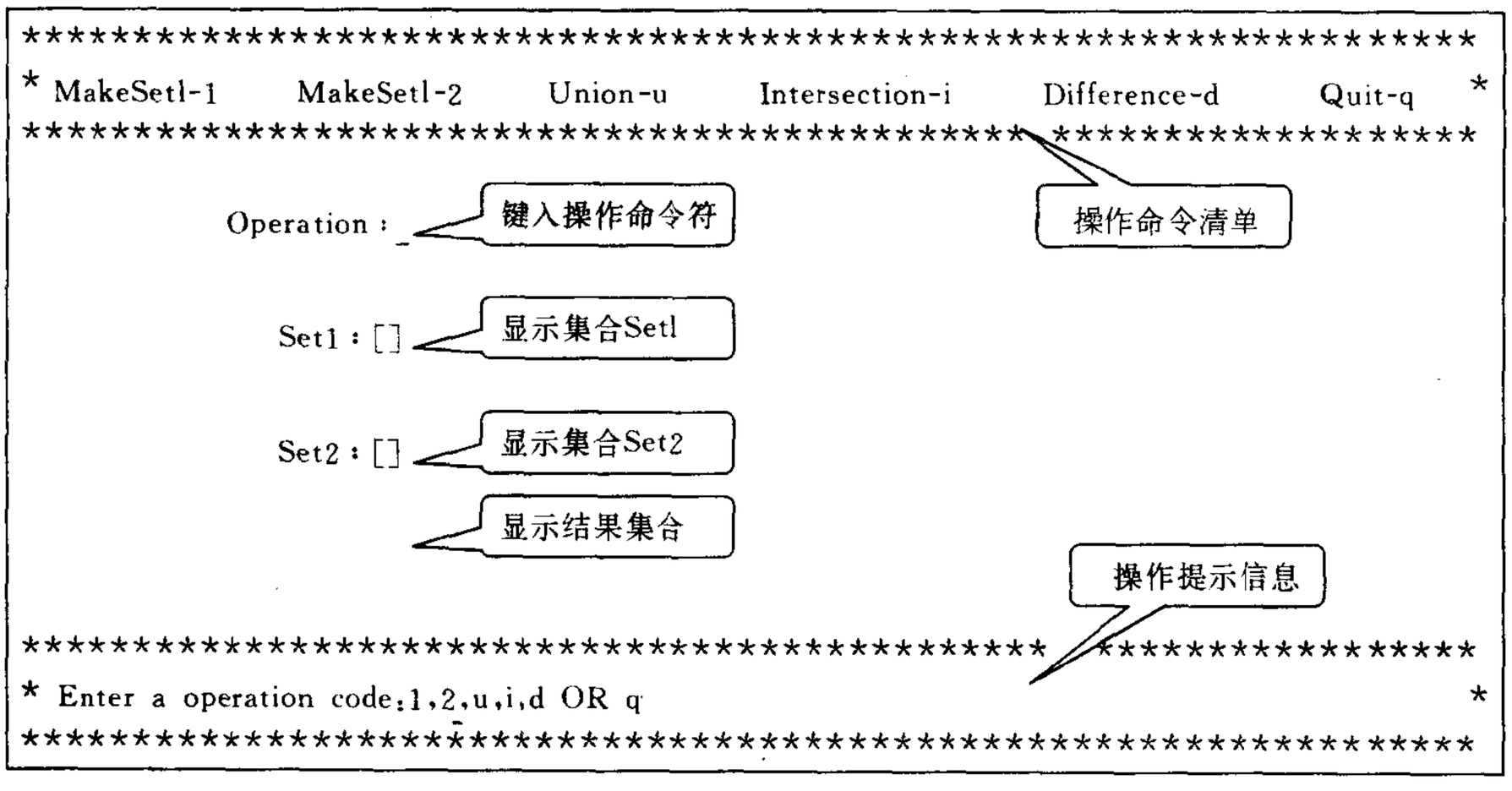
销毁集合算法 DestroySet 和显示集合算法 PrintSet 都是对每个元素调用一个 $O(1)$ 的函数，因此都是 $O(n)$的。

除了构造有序集算法 CreateSet 用一个串变量读入 n 个元素，需要 $O(n)$的辅助空间外，其余算法使用的辅助空间与元素个数无关，即是 $O(1)$的。

5. 本实习作业采用数据抽象的程序设计方法，将程序划分为四个层次结构：元素结点、有序链表、有序集和主控模块，使得设计时思路清晰，实现时调试顺利，各模块具有较好的可重用性，确实得到了一次良好的程序设计训练。

五、用户手册

1. 本程序的运行环境为 DOS 操作系统，执行文件为：SetDemos.exe。
2. 进入演示程序后即显示文本方式的用户界面：



3. 进入“构造集合 1 (MakeSet1)”和“构造集合 2(MakeSet2)”的命令后，即提示键入集合元素串，结束符为“回车符”。

4. 接受其他命令后即执行相应运算和显示相应结果。

六、测试结果

执行命令 '1' ：键入 magazine 后，构造集合 Set1：[a,e,g,i,m,n,z]
执行命令 '2' ：键入 paper 后，构造集合 Set2：[a,e,p,r]
执行命令 'u' ：构造集合 Set1 和 Set2 的并集：[a,e,g,i,m,n,p,r,z]
执行命令 'i' ：构造集合 Set1 和 Set2 的交集：[a,e]
执行命令 'd' ：构造集合 Set1 和 Set2 的差集：[g,i,m,n,z]
执行命令 '1' ：键入 012oper4a6tion89 后，构造集合 Set1：[a,e,i,n,o,p,r,t]
执行命令 '2' ：键入 errordata 后，构造集合 Set2：[a,d,e,o,r,t]
执行命令 'u' ：构造集合 Set1 和 Set2 的并集：[a,d,e,i,n,o,p,r,t]
执行命令 'i' ：构造集合 Set1 和 Set2 的交集：[a,e,o,r,t]

执行命令 'd' ：构造集合 Set1 和 Set2 的差集：[i,n,p]

七、附录

源程序文件名清单：

```
Node.H        // 元素结点实现单元
OrdList.H     // 有序链表实现单元
OrderSet.H    // 有序集实现单元
SetDemos.C    // 主程序
```