实验六 顺序表的查找实验

【实验目的】

掌握以顺序表组织待查数据元素的查找方法，尤其是二分查找方法。

【实验背景】

给定一个值 K，在一组具有相同数据类型的数据元素中找出关键字等于给定值 K 的数据元素(节点)，这个操作过程称为**查找**。若找到，则查找成功，输出该数据元素(节点)的相关信息；否则查找失败，输出查找失败的信息。

为方便查找算法实现，定义用于查找操作的顺序表的数据类型如下：

```
＃define LIST_SIZE 20
typedef struct{
    KeyType key;   //key 为关键字
    OtherType other_data;
} RecordType;
typedef struct{
    RecordType r[LIST_SIZE+1]; // r[0]为工作单元
    int length;          // length 为顺序表的长度
}RecordList;
```

简单顺序查找是从顺序表的一端开始顺序扫描，将给定值 K 依次与顺序表中各数据元素(节点)的关键字比较，若当前扫描到的节点的关键字与给定值 K 相等，则查找成功；若扫描结束后，仍未找到关键字等于 K 的节点，则查找失败。算法具体实现时，将 0 号单元 r[0]作为监视哨，存放给定值 K：

```
int SeqSearch(RecordList L，KeyType k){
//在顺序表 L 中顺序查找其关键字等于 k 的元素，
//若找到，则函数值为该元素在表中的位置，否则为 0。
    L.r[0].key = k;//   0 号单元作为监视哨
    int i = L.length;
    while( L.r[i].key != k )   --i;
    return(i);
}
```

二分查找也称折半查找，它要求待查找的数据元素必须是按关键字大小有序排列的顺序表。其查找过程如下：

① 将表中间位置记录的关键字与给定 K 值比较，如果两者相等，则查找成功；

② 否则 ，利用中间位置记录将表分成前、后两个子表，如果中间位置记录的关键字大于给定 K 值，则进一步查找前一子表，否则进一步查找后一子表。

③ 重复以上过程，直到找到满足条件的记录，则查找成功，或者直到分解出的子表不存在为止，此时查找不成功。

算法实现如下：

```
int BinSrch(RecordList *L，KeyType k){
//在有序表 L 中二分查找其关键字等于 k 的元素，
//若找到，则函数值为该元素在表中的位置
    low=0；i=0；high=L->length-1；//置区间初值
    while (low <= high){
```

```
            mid=(low+high) / 2;
            if (k= = L->r[mid].key){       //找到待查元素
                  i=mid ; break;
            }
            elseif(k< L->r[mid]. key)
                  high=mid-1;       //未找到，则继续在前半区间进行查找
            else   low=mid+1;              //继续在后半区间进行查找
      }
      return (i);
   }
```

分块查找也称索引顺序查找。它主要针对这样的**分块有序**表进行查找：整个列表由若干块(子表)组成，每块内元素排列无序，但每一块中所有元素均小于(大于)其后面块中所有元素，即块间有序。此时，可以为该顺序表建立一个索引表。

索引表中为每一块设置一个索引项，每个索引项记录该块的起始位置，以及该块中的最大关键字(或最小关键字)。索引表按关键字有序排列。

分块查找算法需分两步进行：首先，应用二分查找算法或简单顺序查找算法，将给定值 key 与索引表中的关键字进行比较，以确定待查元素所在的块。然后，应用简单顺序查找算法，在相应块内查找关键字为 key 的数据元素。

【实验任务】

1、程序验证

（1）建立含有若干个整数的顺序表，在此基础上实现顺序表的简单顺序查找算法。

（2）下列函数是以顺序表的最后一个单元作为监视哨的简单顺序查找，阅读该程序，写出运行结果，并通过运行来验证。

```
#define LIST_SIZE 20
typedef struct{
      char   r[LIST_SIZE];
      int length;            // length 为表中元素的个数
}RecordList;
RecordList   *SqLset( ){
       RecordList *L;
       int   i=0;   char   c;
       L=malloc( sizeof(RecordList));
       scanf("%c", &c);       //输入一个字符
       L->length = 0;
      while(c!='#') {   //'#'为输入结束标志
          L->r[i] =c ;
          L->length++;   i++;
          if(L->length = = LIST_SIZE-1)     break;
          scanf("%c", &c);
      }
      return ( L);
}
int   SequenSearch(RecordList   *L，char k,   int   n){
//在长度为 n 的顺序表 L 中顺序查找其关键字等于 k 的元素
```

```
        int   i=0,   position=-1;   // position 为查找到的位置
        L->r[n].key = k;   //最后一个单元作为监视哨
        while(n!=i) {
            if(L->r[i].key = =k) {
                position = i+1;
                break;
              }
            i++;
        }
        return   position;
    }
   main( ) {
     RecordList   *A;
     int   m;
     char   c;
     A = SqLset( );
     scanf("%c", &c);     //输入待查元素
     m = SequenSearch(A，c,   A->length);
     if(m = = -1)
         printf("查找不成功！");
   else   printf("m=%d, 查找成功！",   m);
  }
```

2、算法填空

请在下面算法的空格处填入适当内容，以使算法能够递归实现二分查找过程。

```
 # define LIST_SIZE 20
 typedef struct{
        int   r[LIST_SIZE];
        int length;              // length 为表中元素的个数
 }RecordList;
 RecordList   *SqLset( ){
         RecordList *L;
         int   i=0,   a;
         L=malloc( sizeof(RecordList));
         printf("输入一个整型有序序列，输入 0 时结束！")
         scanf("%d", &a);       //输入一个整数
         L->length = 0;
        while(a!=0) {   //整数 0 为输入结束标志，输入一个有序的整数序列
            L->r[i] =a ;
            L->length++;   i++;
            if(L->length = = LIST_SIZE-1)     break;
            scanf("%d", &a);
        }
        return ( L);
 }
```

```
int  BinSrch(RecordList *L，int  k,  int  low,  int  high){
//在有序表 L 中二分查找其关键字等于 k 的元素，
      int  mid;
      if(low > high)  return  -1；
      else  {
            mid=(low+high) / 2；
            if ( _____ )          //找到待查元素
                  return  mid ；
            else  if(k< L->r[mid])
                      return    _____ ；
                  else
                      return    _____ ；
          }
   }
main( ) {
   RecordList  *A;
   int  k,  low,  high;
   A = SqLset( );
   low = 0;     high = A->length-1
   scanf(“%d”, &k);    //输入待查元素
   m = BinSrch(A, k, low, high);
   if(m = = -1)
       printf(“查找不成功！”)；
  else  printf(“m=%d, 查找成功！”,  m)；
}
```

3、算法设计

（1）为二分查找的非递归算法设计主函数，完善该算法，并通过运行来验证。

（2）设顺序表中的关键字是递增有序的，将监视哨设在高下标端，设计算法实现简单顺序查找。

4、实例演练

[问题描述]

将实验二实例演练中的通讯录索引存储，应用顺序查找和二分查找技术实现查询功能。

[基本要求]

① 建立两个顺序表，一个为索引表，索引存储姓氏首字母及地址信息；另一个顺序表，按姓氏首字母不同分块存储通讯录信息。在索引表应用二分查找技术，查找姓氏首字母及该类姓氏在通讯录表中的起始位置和末位置；在通讯录表中应用顺序查找技术完成查询。

② 设计算法完成问题求解；

③ 分析算法的时间复杂度和空间复杂度。

[实现提示]

①定义索引表的结点类型：

```
typedef  struct {
    char  firstname;      // 姓名首字母
    int     address;          // 该姓氏在通讯录表中的起始位置
} IndexData;
IndexData  Index[26];      // 该姓氏在通讯录表中的起始位置
```

②建立索引表和通讯录表

首先对索引表中的 firstname 分量递增有序分别赋值 a～z，address 分量均赋值为 0；录入通讯者信息时，按照姓氏首字母递增顺序将其插入到通讯录表中，同时判断是否修改索引表中相应的 address 分量值，并将其后所有元素的 address 分量值加 1。

③按姓氏首字母查询信息

在索引表中应用二分查找算法，查找该类姓氏在通讯录表中的起始位置 Index[i].adress、末位置 Index[i＋1].adress－1；在通讯录表中相应的块中应用简单顺序查找算法，实现信息查询。

④功能函数设计

功能函数包括通讯者的插入、建立索引表和通讯录表、通讯者的查询等 3 个函数；在建立通讯录表函数中调用插入函数。

[思维扩展]

考虑按其它关键字进行查询时索引表的建立方法和查询方法。