北京交通大學
BEIJING JIAOTONG UNIVERSITY

高级程序设计训练

SPT-02 线性表

《实验一：顺序表的实现及基本操作》

- 创建有若干个元素（可以是整型数值）的顺序表，实现对顺序表的初始化，对已建立的顺序表插入操作、删除操作、遍历输出顺序表。
- 要求各个操作均以函数的形式实现，在主函数中调用各个函数实现以下操作：

① 创建顺序表21、18、30、75、42、56，并输出顺序表中的各元素值。

② 在顺序表的第3个位置插入67，并输出此时顺序表中的各元素值。

③ 删除顺序表中的第6个数据元素，并输出此时顺序表中的各元素值。

④ 查找顺序表中是否有75这个元素，如果有返回该元素在顺序表中的位序

《实验一：顺序表的实现及基本操作》

- 上一节课以定长顺序存储结构完成了该题目，那么如果使用动态分配顺序存储结构来实现相同功能，我们应该对代码作何修改？

```
#define LIST_SIZE   100    //线性表存储空间的分配量
typedef struct{
   ElemType  elem[LIST_SIZE]; //存放线性表元素的一维数组
     int length;  //线性
}SqList;
```

```
//------------线性表的动态分配顺序存储结构-----------//
#define LIST_SIZE   100          //线性表存储空间的初始分配量
#define LISTINCREMENT  10    //线性表存储空间的分配增量
typedef struct{
   ElemType  *elem;              //存储空间基址
     int      length;                    //当前长度
     int      listsize;                //当前分配的存储容量
}SqList;
```

实验1的动态分配顺序存储实现

结构体的修改

```
#include "stdio.h"

#define OK 1
#define ERROR 0
#define TRUE 1
#define FALSE 0

#define MAXSIZE 20 /* 存储空间初始分配量 */

typedef int Status;        /* Status是函数的类型,其值是函数结果状态代码，如OK等 */
typedef int ElemType;      /* Ele                    标情况而定，这里假设为int */

typedef struct
{
  ElemType data[MAXSIZE];          /* 数组，存储数据元素 */
  int length;                      /* 线性表当前长度 */
}SqList;
```

```
#define INIT_SIZE 100       /*初始分配空间的大小*/
#define LISTINCREMENT 10    /*分配增量*/
```

ElemType *elem;

int listsize;

实验1的动态分配顺序存储实现

- 初始化函数的修改

```
Status InitList(SqList *L)
{
    L->length=0;
    return OK;
}
```

实验1的动态分配顺序存储实现

初始化函数的修改

```
Status InitList(SqList *L)
{
    L->elem = (ElemType *)malloc(INIT_SIZE*sizeof(ElemType));
    if (!L->elem)
    {
        return ERROR;
    }

    L->length=0;
    return OK;
}
```

实验1的动态分配顺序存储实现

- 初始化函数的修改

```
Status InitList(SqList *L)
{
    L->elem = (ElemType *)malloc(INIT_SIZE*sizeof(ElemType));
    if (!L->elem)
    {
        return ERROR;
    }

    L->length=0;

    L->listsize = INIT_SIZE;
    /*申请的空间为初始大小*/

    return OK;
}
```

实验1的动态分配顺序存储实现

找元素位置函数的修改

```
/* 初始条件：顺序线性表L已存在 */
/* 操作结果：返回L中第1个与e满足关系的数据元素的位序。 */
/* 若这样的数据元素不存在，则返回值为0 */
int LocateElem(SqList L,ElemType e)
{
    int i;
    if (L.length==0)
            return 0;
    for(i=0;i<L.length;i++)
    {
            if ( L.elem [i] ==e)
                    break;
    }
    if(i>=L.length)
            return 0;

    return i+1;
}
```

实验1的动态分配顺序存储实现

插入元素函数的修改

```
/* 初始条件：顺序线性表L已存在,1≤i≤ListLength(L)， */
/* 操作结果：在L中第i个位置之前插入新的数据元素e，L的长度加1 */
Status ListInsert(SqList *L,int i,ElemType e)
{
  int k;

  if (i<1 || i>L->length+1)/* 当i比第一位置小或者比最后一位置后一位置还要大时 */
    return ERROR;

  if (L->length==MAXSIZE)  /* 顺序线性表已经满 */
    return ERROR;


  if (i<=L->length)        /* 若插入数据位置不在表尾 */
  {
    for(k=L->length-1;k>=i-1;k--)  /* 将要插入位置之后的数据元素向后移动一位 */
      L->data[k+1]=L->data[k];
  }
  L->data[i-1]=e;          /* 将新元素插入 */
  L->length++;

  return OK;
}
```

实验1的动态分配顺序存储实现

插入元素函数的修改

```
/* 初始条件：顺序线性表L已存在,1≤i≤ListLength(L)， */
/* 操作结果：在L中第i个位置之前插入新的数据元素e，L的长度加1 */
Status ListInsert(SqList *L,int i,ElemType e)
{
  int k;

  if (i<1 || i>L->length+1)/* 当i比第一位置小或者比最后一位置后一位置还要大时 */
    return ERROR;

if (L->length >= L->listsize)  /*当前空间已满，增加分配空间*/
{
    newbase=(ElemType *)realloc(L->elem,(L->listsize+LISTINCREMENT)*sizeof(ElemType));
    if (!newbase)
      return(ERROR);

    L->elem = newbase;
    L->listsize = L->listsize + LISTINCREMENT;
}
```

实验1的动态分配顺序存储实现

- 插入元素函数的修改

```
if (i<=L->length)          /* 若插入数据位置不在表尾 */
{
  for(k=L->length-1;k>=i-1;k--)  /* 将要插入位置之后的数据元素向后移动一位 */
    L->elem[k+1] = L->elem[k];
}
L->data[i-1]=e;            /* 将新元素插入 */
L->length++;

return OK;
}
```

实验1的动态分配顺序存储实现

删除元素函数的修改

```
Status ListDelete(SqList *L,int i,ElemType *e)
{
   int k=0;
   if (L->length==0)                    /* 线性表为空 */
      return ERROR;

   if (i<1 || i>L->length)              /* 删除位置不正确 */
      return ERROR;

   *e=L->data[i-1];

   if (i<L->length)                     /* 如果删除不是最后位置 */
   {
      for(k = i;k <L->length; k++)/* 将删除位置后继元素前移 */
         L->data[k-1] = L->data[k];
   }

   L->length--;
   return OK;
}
```

实验1的动态分配顺序存储实现

删除元素函数的修改

```
Status DeleteList_Sq(SqList *L, int i, ElemType *e)
/* 删除线性表中的第i个元素 */
{
    int k=0;
    if(L->length == 0)
        return ERROR;

    if ((i<1)||(i>L->length))
        return ERROR;

    *e = L->elem[i-1];

    if(i < L->length)
    {
        for (k=i; k < L->length; k++)
            L->elem[k-1]=L->elem[k];
    }

    L->length --;

    return OK;
}
```

实验1的动态分配顺序存储实现

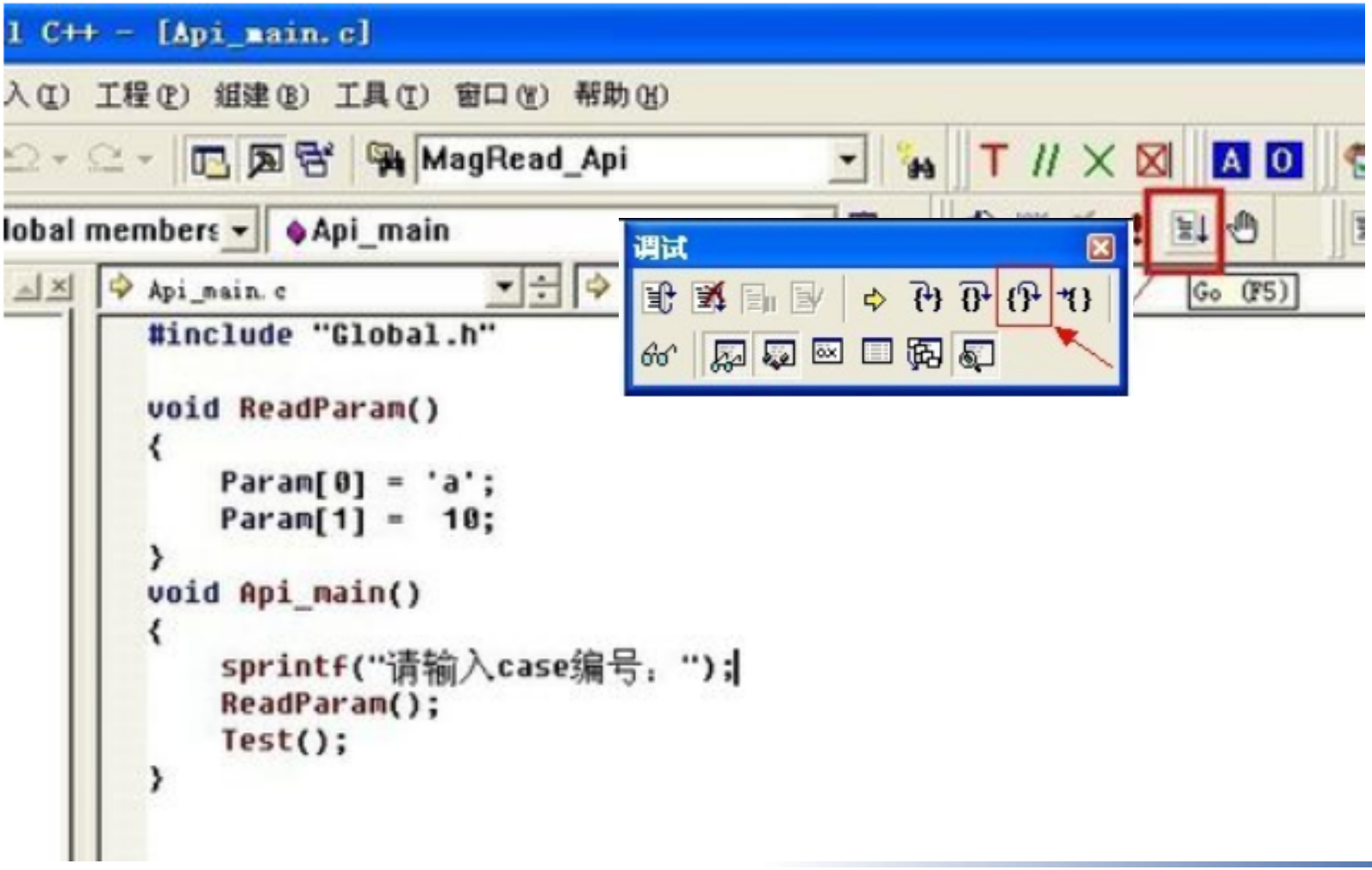
- 遍历函数的修改

```
/* 初始条件：顺序线性表L已存在 */
/* 操作结果：依次对L的每个数据元素输出 */
Status ListTraverse(SqList L)
{
  int i;
    for(i=0;i<L.length;i++)
            visit(L.data[i]);
    return OK;
}
```

```c
int main() {

    SqList Lq;
    ElemType e=0;
    Status i = 0;
    int j = 0;

    i = InitList(&Lq);
    ListInsert(&Lq, 1, 21);
    ListInsert(&Lq, 2, 18);
    ListInsert(&Lq, 3, 30);
    ListInsert(&Lq, 4, 75);
    ListInsert(&Lq, 5, 42);
    ListInsert(&Lq, 6, 56);
    printf("初始顺序表为\n");
    ListTraverse(Lq);

    ListInsert(&Lq, 3, 67);
    printf("插入67后顺序表为\n");
    ListTraverse(Lq);

    ListDelete(&Lq, 6, &e);
    printf("删除第6个元素后顺序表为\n");
    ListTraverse(Lq);

    j = LocateElem(Lq, 75);
    if (j >= 1)
        printf("元素75的位置为%d\n", j);
    else
        printf("不存在元素75\n");

    return 0;
}
```

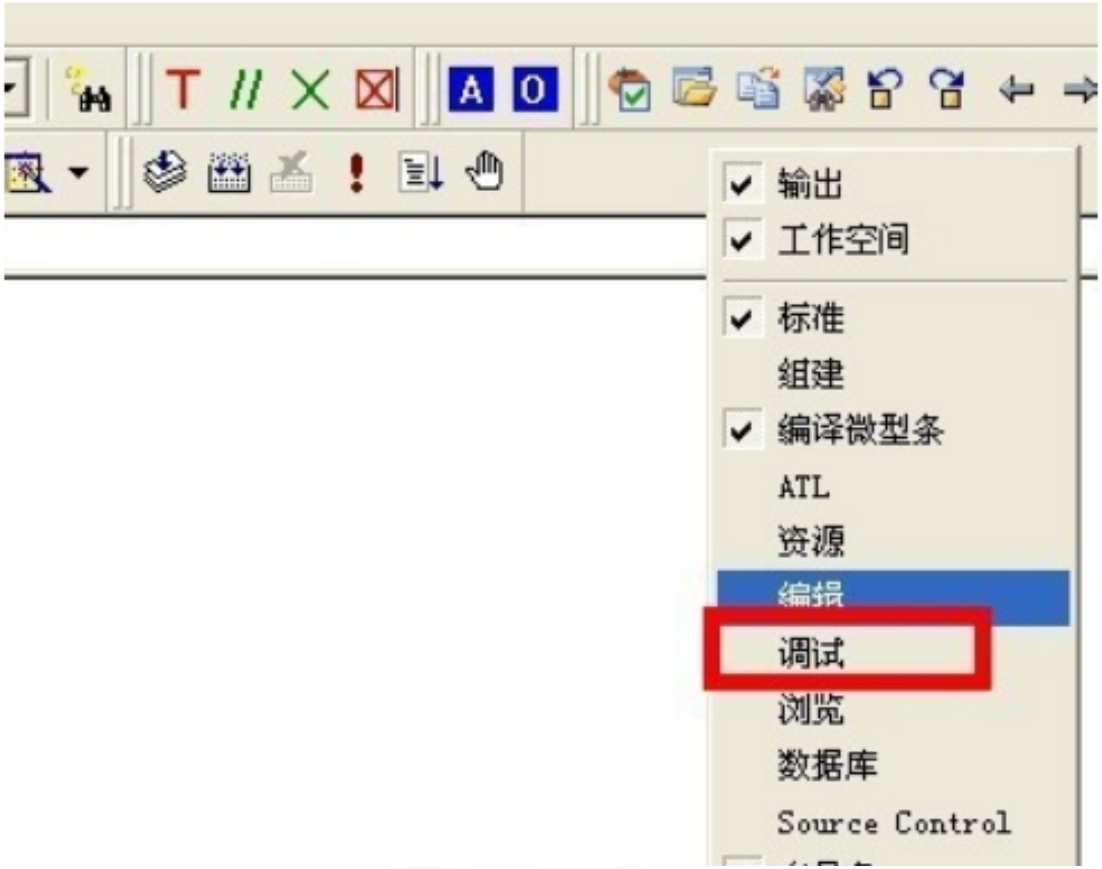
VC++6.0 的程序调试方法

按快捷键F5或点击以下图片上标记的图标进入调试模式。



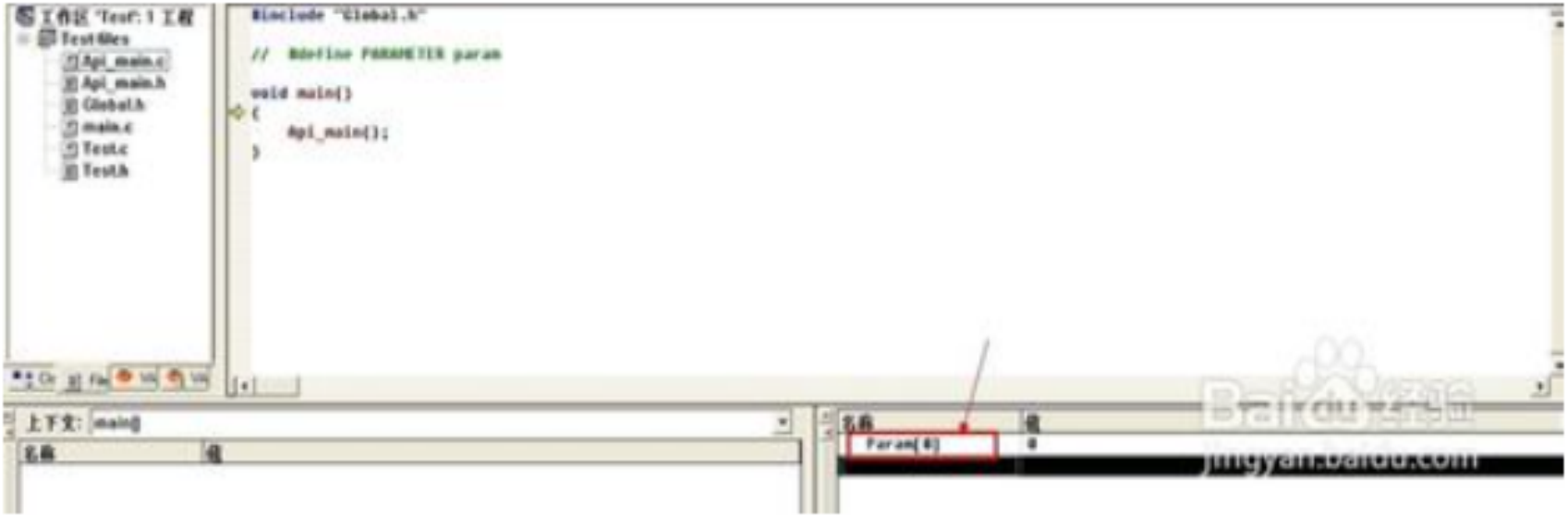
VC++6.0 的程序调试方法

打开调试工具条，一般情况下当你按F5键后会自动弹出，如果没有弹出的话，右击工具栏空白处，会弹出下图，选中调试，就会出现调试工具条。



VC++6.0 的程序调试方法

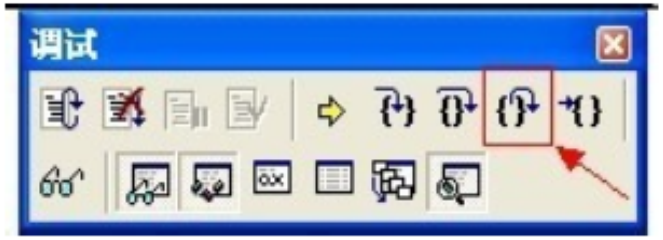
接下来，在监视窗口中添加你要监视数据变化的变量。



VC++6.0 的程序调试方法

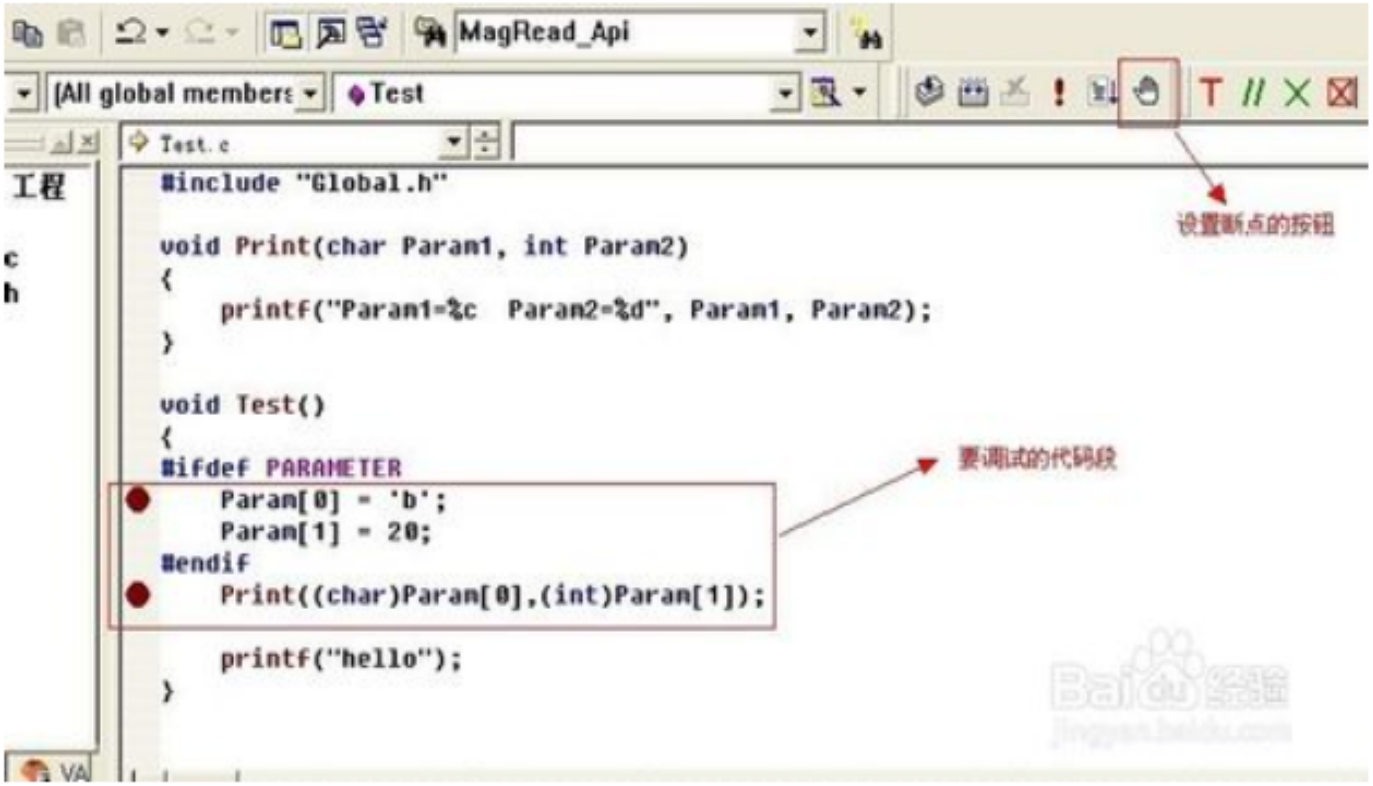
按F11逐语句的调试代码，如果某一语句是一函数，你不希望进入该函数时，F10逐过程来查看，在代码的调试过程中，通过监视窗口查看变量值的变化，从而确定代码是否有问题。

如果希望跳出某一函数时，按快捷键Shift+F11，或者直接点击调试工具条上的按钮就可以跳出该函数了。



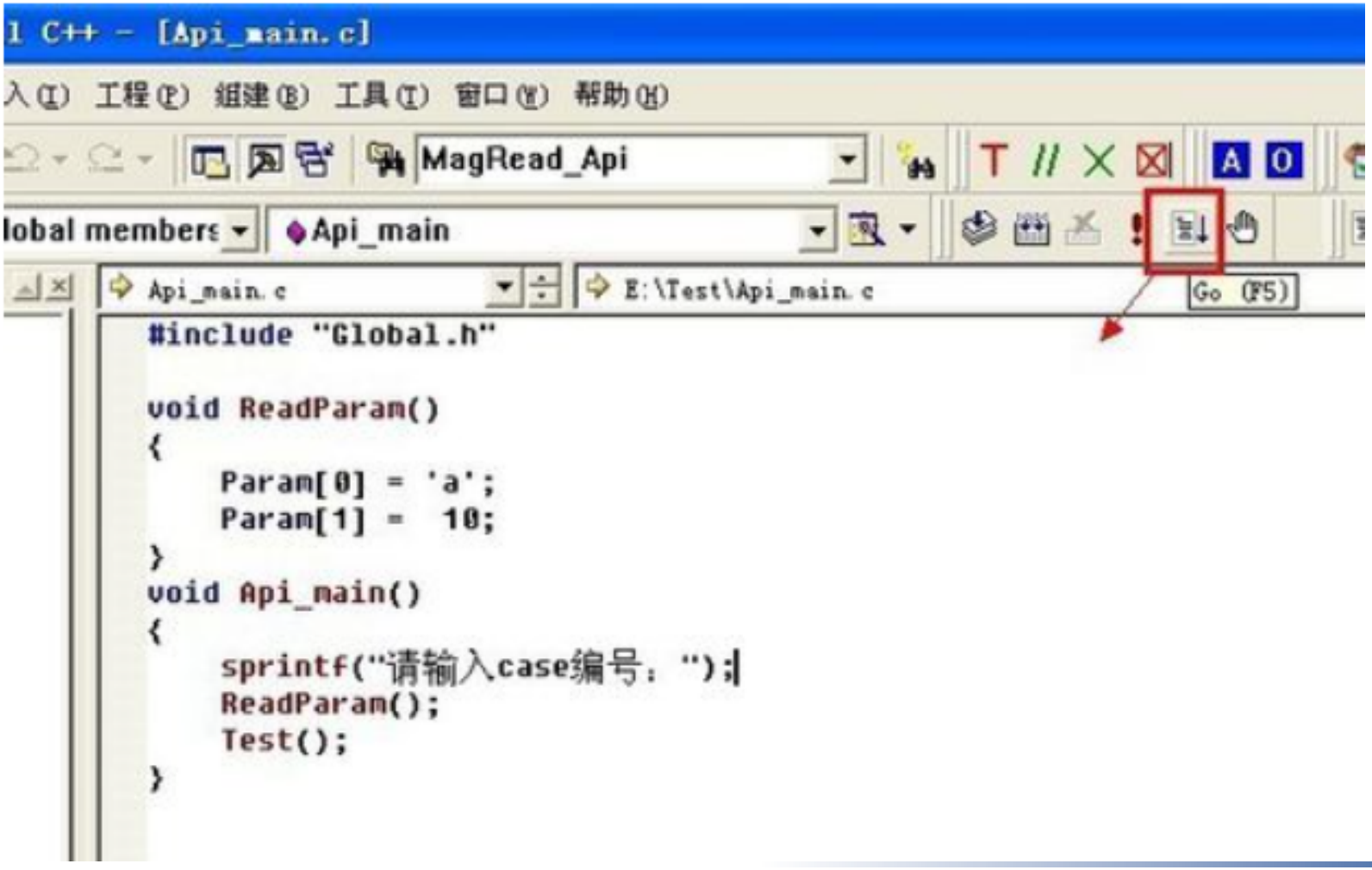
VC++6.0 的程序调试方法

如果只是希望调试某一部分代码的话，可以设置断点来调试，调试方法如上，只是在要调试的代码之间用断点来分开。设置断点时，光标放在你要设置断点的那行，按F9或点击编译微型条上面的类似手状的按钮即可。



VC++6.0 的程序调试方法

按快捷键F5或点击以下图片上标记的图标进入调试模式。



例1 线性表合并

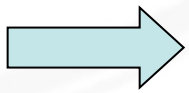
问题描述：

假设利用两个线性表LA和LB分别表示两个集合A和B,现要求一个新的集合

$$A=A\cup B$$

LA=(3, 5, 8, 11)

LB=(2, 6, 8, 9, 11, 15, 20)



LA=(3, 5, 8,11,2,6, 9, 15, 20)

```
void union(List &La, List Lb){
  La_len=ListLength(La);
  Lb_len=ListLength(Lb);
  for(i=1;i<=Lb_len;i++){
      GetElem(Lb,i,&e);
      if(!LocateElem(La,e,equal))
          ListInsert(&La,++La_len,e);
  }
} //union
```

```
/* 初始条件：顺序线性表L已存在。操作结果：返回L中数据元素个数 */
int ListLength(SqList L)
{
    return L.length;
}

/* 初始条件：顺序线性表L已存在，1≤i≤ListLength(L) */
/* 操作结果：用e返回L中第i个数据元素的值，注意i是指位置，第1个位置的数组是从0开始 */
Status GetElem(SqList L,int i,ElemType *e)
{
    if(L.length==0 || i<1 || i>L.length)
        return ERROR;
    *e=L.data[i-1];

    return OK;
}
```

```
void union(List &La, List Lb){
  La_len=ListLength(La);
  Lb_len=ListLength(Lb);
  for(i=1;i<=Lb_len;i++){
    GetElem(Lb,i, e);
    if(!LocateElem(La,e,equal))
      ListInsert(La,++La_len,e);
  }
} //union
```

LA=(3, 5, 8, 11)

LB=(2, 6, 8, 9, 11, 15, 20)

```
void union(List &La, List Lb){
  La_len=ListLength(La);
  Lb_len=ListLength(Lb);
  for(i=1;i<=Lb_len;i++){
     GetElem(Lb,i, e);
     if(!LocateElem(La,e,equal))
        ListInsert(La,++La_len,e);
  }
} //union
```

LA=(3, 5, 8, 11)

LB=(2, 6, 8, 9, 11, 15, 20)

```
void union(List &La, List Lb){
  La_len=ListLength(La);
  Lb_len=ListLength(Lb);
  for(i=1;i<=Lb_len;i++){
     GetElem(Lb,i, e);
     if(!LocateElem(La,e,equal))
        ListInsert(La,++La_len,e);
  }
} //union
```

LA=(3, 5, 8, 11)

LB=(2, 6, 8, 9, 11, 15, 20)

i=1

```
void union(List &La, List Lb){
  La_len=ListLength(La);
  Lb_len=ListLength(Lb);
  for(i=1;i<=Lb_len;i++){
      GetElem(Lb,i, e);
      if(!LocateElem(La,e,equal))
          ListInsert(La,++La_len,e);
  }
} //union
```

LA=(3, 5, 8, 11)

LB=(2, 6, 8, 9, 11, 15, 20)

i=1

```
void union(List &La, List Lb){
  La_len=ListLength(La);
  Lb_len=ListLength(Lb);
  for(i=1;i<=Lb_len;i++){
     GetElem(Lb,i, e);
     if(!LocateElem(La,e,equal))
        ListInsert(La,++La_len,e);
  }
} //union
```

LA=(3, 5, 8, 11)

LB=(2, 6, 8, 9, 11, 15, 20)

i=1

```
void union(List &La, List Lb){
  La_len=ListLength(La);
  Lb_len=ListLength(Lb);
  for(i=1;i<=Lb_len;i++){
     GetElem(Lb,i, e);
     if(!LocateElem(La,e,equal))
          ListInsert(La,++La_len,e);
  }
} //union
```

LA=(3, 5, 8, 11, 2)

LB=(2, 6, 8, 9, 11, 15, 20)

i=1

```
void union(List &La, List Lb){
  La_len=ListLength(La);
  Lb_len=ListLength(Lb);
  for(i=1;i<=Lb_len;i++){
      GetElem(Lb,i, e);
      if(!LocateElem(La,e,equal))
          ListInsert(La,++La_len,e);
  }
} //union
```

LA=(3, 5, 8, 11, 2)

LB=(2, 6, 8, 9, 11, 15, 20)

i=2

```
void union(List &La, List Lb){
  La_len=ListLength(La);
  Lb_len=ListLength(Lb);
  for(i=1;i<=Lb_len;i++){
      GetElem(Lb,i, e);
      if(!LocateElem(La,e,equal))
          ListInsert(La,++La_len,e);
  }
} //union
```

LA=(3, 5, 8, 11, 2)

LB=(2, 6, 8, 9, 11, 15, 20)

i=2

```
void union(List &La, List Lb){
  La_len=ListLength(La);
  Lb_len=ListLength(Lb);
  for(i=1;i<=Lb_len;i++){
     GetElem(Lb,i, e);
     if(!LocateElem(La,e,equal))
        ListInsert(La,++La_len,e);
  }
} //union
```

LA=(3, 5, 8, 11, 2)

LB=(2, 6, 8, 9, 11, 15, 20)

i=2

```
void union(List &La, List Lb){
  La_len=ListLength(La);
  Lb_len=ListLength(Lb);
  for(i=1;i<=Lb_len;i++){
     GetElem(Lb,i, e);
     if(!LocateElem(La,e,equal))
          ListInsert(La,++La_len,e);
  }
} //union
```

LA=(3, 5, 8, 11, 2, 6)

LB=(2, 6, 8, 9, 11, 15, 20)

i=2

```
void union(List &La, List Lb){
  La_len=ListLength(La);
  Lb_len=ListLength(Lb);
  for(i=1;i<=Lb_len;i++){
      GetElem(Lb,i, e);
      if(!LocateElem(La,e,equal))
          ListInsert(La,++La_len,e);
  }
} //union
```

LA=(3, 5, 8, 11, 2, 6)

LB=(2, 6, 8, 9, 11, 15, 20)

i=3

```
void union(List &La, List Lb){
  La_len=ListLength(La);
  Lb_len=ListLength(Lb);
  for(i=1;i<=Lb_len;i++){
      GetElem(Lb,i, e);
      if(!LocateElem(La,e,equal))
          ListInsert(La,++La_len,e);
  }
} //union
```

LA=(3, 5, 8, 11, 2, 6)

LB=(2, 6, 8, 9, 11, 15, 20)

i=3

```
void union(List &La, List Lb){
  La_len=ListLength(La);
  Lb_len=ListLength(Lb);
  for(i=1;i<=Lb_len;i++){
     GetElem(Lb,i, e);
     if(!LocateElem(La,e,equal))
        ListInsert(La,++La_len,e);
  }
} //union
```

LA=(3, 5, 8, 11, 2, 6)

LB=(2, 6, 8, 9, 11, 15, 20)

i=3

```
void union(List &La, List Lb){
  La_len=ListLength(La);
  Lb_len=ListLength(Lb);
  for(i=1;i<=Lb_len;i++){
      GetElem(Lb,i, e);
      if(!LocateElem(La,e,equal))
          ListInsert(La,++La_len,e);
  }
} //union
```

LA=(3, 5, 8, 11, 2, 6)

LB=(2, 6, 8, 9, 11, 15, 20)

i=4

```
void union(List &La, List Lb){
  La_len=ListLength(La);
  Lb_len=ListLength(Lb);
  for(i=1;i<=Lb_len;i++){
      GetElem(Lb,i, e);
      if(!LocateElem(La,e,equal))
          ListInsert(La,++La_len,e);
  }
} //union
```

LA=(3, 5, 8, 11, 2, 6)

LB=(2, 6, 8, 9, 11, 15, 20)

i=4

```
void union(List &La, List Lb){
  La_len=ListLength(La);
  Lb_len=ListLength(Lb);
  for(i=1;i<=Lb_len;i++){
     GetElem(Lb,i, e);
     if(!LocateElem(La,e,equal))
        ListInsert(La,++La_len,e);
  }
} //union
```

LA=(3, 5, 8, 11, 2, 6)

LB=(2, 6, 8, 9, 11, 15, 20)

i=4

```
void union(List &La, List Lb){
  La_len=ListLength(La);
  Lb_len=ListLength(Lb);
  for(i=1;i<=Lb_len;i++){
     GetElem(Lb,i, e);
     if(!LocateElem(La,e,equal))
          ListInsert(La,++La_len,e);
  }
} //union
```

LA=(3, 5, 8, 11, 2, 6, 9)

LB=(2, 6, 8, 9, 11, 15, 20)

i=4

```
void union(List &La, List Lb){
  La_len=ListLength(La);
  Lb_len=ListLength(Lb);
  for(i=1;i<=Lb_len;i++){
     GetElem(Lb,i, e);
     if(!LocateElem(La,e,equal))
        ListInsert(La,++La_len,e);
  }
} //union
```

$O(ListLength(LA) \times ListLength(LB))$

LA=(3, 5, 8, 11, 2, 6, 9, 15, 20)

LB=(2, 6, 8, 9, 11, 15, 20)

i=8

例2 有序表合并

问题描述：

已知线性表LA 和LB中的数据元素按值非递减有序排列,现要求将LA和LB归并为一个新的线性表LC,且LC中的数据元素仍按值非递减有序排列.

LA=(3, 5, 8, 11)

LB=(2, 6, 8, 9, 11, 15, 20)

LC=(2, 3, 5, 6, 8, 8, 9, 11, 11, 15, 20)

```
void MergeList(List La,List Lb,List &Lc){
  InitList(Lc);    i=j=1;k=0;
  La_len=ListLength(La); Lb_len=ListLength(Lb);
  while((i<=La_len)&&(j<=Lb_len)){
     GetElem(La, i, ai); GetElem(Lb, j, bj);
     if(ai<=bj) {ListInsert(Lc, ++k, ai); ++i;}
     else{ListInsert(Lc,++k, bj); ++j;}
  }
  while(i<=La_len){
    GetElem(La, i++, ai); ListInsert(Lc, ++k, ai);
  }
  while(j<=Lb_len){
    GetElem(Lb, j++, bj); ListInsert(Lc, ++k, bj);
} // MergeList
```

LA=(3, 5, 8, 11)
LB=(2, 6, 8, 9, 11, 15, 20)

LC=(2, 3, 5, 6, 8, 8, 9, 11, 11, 15, 20)

O(ListLength(LA) + ListLength(LB))