§2. 线性表

2.1. 线性表的类型定义

2.1.1. 线性结构的特点

★ 线性结构：元素间存在一对一的关系

- 存在唯一一个称为"第一"的数据元素
- 存在唯一一个称为"最后"的数据元素
- 除最后一个外，每个元素仅有一个后继
- 除第一个外，每个元素仅有一个前驱

2.1.2. 线性表的含义

具有相同特征的数据元素的有限序列

★ 数据元素可以是任意形式，较复杂的一般表示为一个记录，由若干数据项组成，则整个线性表又可称为文件

2.1.3. 线性表的长度

序列中所含的元素的个数称为线性表的长度，用n(n≥0)表示

★ 当n=0时，表示一个空表，即表中不含任何元素

§2. 线性表

2.1. 线性表的类型定义

2.1.4. 线性表的表示形式

设序列中第i个元素为a_i（$1 \leqslant i \leqslant n$），则线性表一般表示为：

$(a_1, a_2, \ldots, a_i, \cdots, a_n)$

★ a_1为第1个元素，称为表头元素，a_n为最后一个元素，称为表尾元素

★ 一个线性表可以用一个标识符来命名，例：

$L=(a_1, a_2, \ldots, a_i, \ldots a_n)$

★ 线性表中的元素在位置上是有序的，即第i个元素a_i处在第i-1个元素a_{i-1}后面和第i+1个元素a_{i+1}的前面，这种位置上的有序性就是一种线性关系

★ 称i为数据元素a_i在线性表中的位序

2.1.5. 序偶

$\langle a_{i-1}, a_i \rangle$称为一个序偶，表示线性表中数据元素的相邻关系

★ a_{i-1}称为序偶的第一元素，a_i称为第二元素

a_{i-1}称为a_i的直接前驱，a_i称为a_{i-1}的直接后继

★ 当$i=1, 2, \ldots, n-1$时， a_i有且仅有一个直接后继

当$i=2, 3, \ldots, n$ 时， a_i有且仅有一个直接前驱

2.1.6. 抽象数据类型的线性表的定义(P. 19 - 20)

★ 在写算法时，假设这些基本操作均已实现，可以直接使用

§2. 线性表

2.2. 线性表的顺序表示和实现

2.2.1. 顺序表示的特点

用一组地址连续的存储单元依次存储线性表的数据元素，借助元素在存储器中的相对位置来表示元素间的逻辑关系

★ 假设线性表的每个元素需占用L个存储单元，并以所占的第一个单元的存储地址作为数据元素的起始存储位置，则线性表中第i+1个数据元素的存储位置$Loc(a_{i+1})$和第i个数据元素的存储位置$Loc(a_i)$之间满足下列关系：

$$Loc(a_{i+1}) = Loc(a_i) + L$$

★ 线性表的第i个元素a_i的存储位置和a_1的关系为：

$$Loc(a_i) = Loc(a_1) + (i-1)*L$$

★ a_1(表头元素)通常称作线性表的起始位置或基地址

★ 每个元素的存储位置和起始位置相差一个和数据元素在线性表中的位序成正比的常数（即L）

★ 只要确定了线性表的起始位置，即可随机存取表中任一元素

★ C/C++语言中数组具备顺序存储的特点，但数组大小必须固定，因此不直接使用数组，而是用动态申请空间的方法模拟数组，方便线性表的扩大

★ 形式化定义中线性表从1..n，C/C++中数组从0..n-1

假设数据元素为int型，则：

（1）采用数组形式：

```
#define MAX_NUM 100
int a[MAX_NUM];
```

可用a[i]形式访问，当线性表中元素满100后，无法再增加，如果初始值设置很大，则会造成巨大的浪费

（2）采用C动态申请空间模拟数组形式：

```
#define MAX_NUM 100
int *a;
a = (int *)malloc(MAX_NUM * sizeof(int));
```

也可用a[i]形式访问，当线性表中元素满100后，可以再增加，方法：

```
a = (int *)realloc(a, (MAX_NUM+10)*sizeof(int));
```

附：realloc函数的原型定义及使用

```
void *realloc(void *ptr, unsigned int newsize);
```

★ 表示为指针ptr重新申请newsize大小的空间

★ ptr必须是malloc/calloc/realloc返回的指针

★ 新老空间可重合，也可能不重合，若不重合，原空间原有内容会被复制到新空间，再释放原空间

★ 更多内容可自行查阅（Linux下 man realloc 查询）

```
#include <iostream>
#include <stdlib.h> //VS2015可省略
using namespace std;
int main()
{
    void *p;
    p = malloc(1024 * 1024);
    //省略申请是否成功的判断
    cout << p << endl;

    p = realloc(p, 1024 * 1024 + 1024); //换为+4096再试
    //省略申请是否成功的判断
    cout << p << endl;

    free(p);
    return 0;
}
```

用 VS2015
VC++6.0
CodeBlocks 分别编译

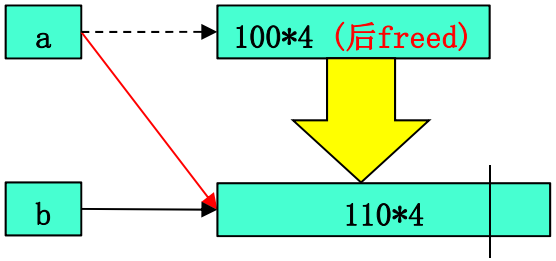
（3）采用C++的动态申请空间模拟数组形式：

```
#define MAX_NUM 100
int *a;
a = new int[MAX_NUM];
```

也可用a[i]形式访问，当线性表中元素满100后，可以再增加，方法：

```
int *b = new int[MAX_NUM+10]; //申请新
for(i=0; i<MAX_NUM; i++)  //原=>新
    b[i] = a[i];
delete a;  //释放原空间
a = b;     //原指针指向新空间
```

思考：如果新空间小于原空间，应如何？



§2. 线性表

2.2. 线性表的顺序表示和实现

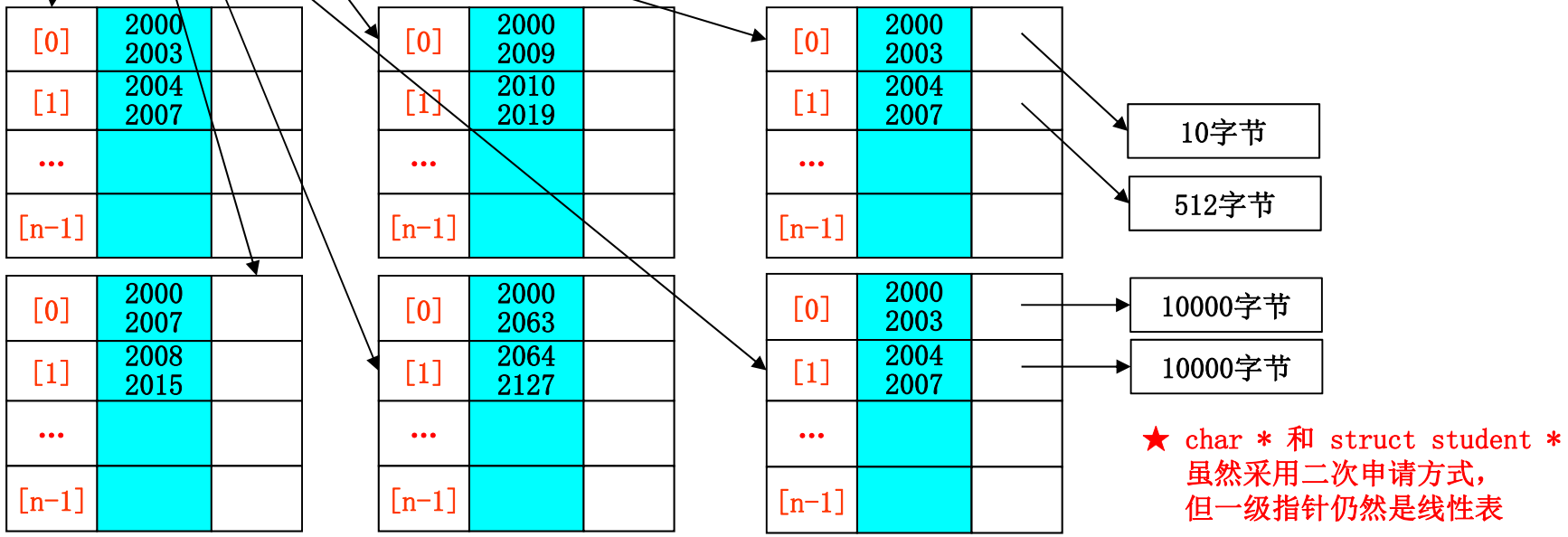
2.2.2. 线性表顺序表示的基本操作的实现

2.2.2.1. C语言版

★ 线性表的数据类型

线性表允许存放任何类型的数据，不失一般性，讨论以下四种类型(六种形式)：

int	：整形数据	
double	：浮点型数据	
char []	：定长或不定长字符串	（图示中假设定长10字节）
char *	：定长或不定长字符串（二次申请）	（图示中假设不定长）
struct student	：复杂结构体（几十 ~ 几千字节）	（图示中假设定长64字节）
struct student *	：复杂结构体（二次申请）	（图示中假设定长10000字节）



★ char * 和 struct student * 虽然采用二次申请方式，但一级指针仍然是线性表

§2.线性表

2.2.线性表的顺序表示和实现

2.2.2.线性表顺序表示的基本操作的实现

2.2.2.1.C语言版

★ 程序的组成

● linear_list_sq.h	：头文件
● linear_list_sq.c	：具体实现
● linear_list_sq_main.c	：使用（测试）示例

说明：从思维上把这个程序理解为两个人完成，其中一个人完成linear_list_sq.h和.c（提供线性表基本功能的实现），另一个人完成 linear_list_sq_main.c（用他人提供的线性表基本操作函数要实现自己的应用目的），两人层次不同（底层向上层提供支持），适合团队合作和分工

★ 与书上算法的区别

- C语言无引用，需要用指针代替
- 临时变量算法中无定义，程序要补齐
- 某些形式化定义和实际表示之间有区别

§2.线性表

2.2.线性表的顺序表示和实现

2.2.2.线性表顺序表示的基本操作的实现

2.2.2.1.C语言版

★ 程序的组成

★ 与书上算法的区别

- C语言无引用，需要用指针代替
- 临时变量算法中无定义，程序要补齐
- 某些形式化定义和实际表示之间有区别

★ linear_list_sq.h 中各定义项的解析

```
/* linear_list_sq.h 的组成 */
#define TRUE          1
#define FALSE         0
#define OK            1
#define ERROR         0
#define INFEASIBLE   -1
#define OVERFLOW     -2

typedef int Status;
```

P.10 预定义常量和类型

(1) 预定义常量和类型：

```
// 函数结果状态代码
#define  TRUE          1
#define  FALSE         0
#define  OK            1
#define  ERROR         0
#define  INFEASIBLE    -1
#define  OVERFLOW      -2
// Status 是函数的类型,其值是函数结果状态代码
typedef  int  Status;
```

```
/* linear_list_sq.h 的组成 */
#define LIST_INIT_SIZE      100 //初始大小为100(可按需修改)
#define LISTINCREMENT       10  //空间分配增量(可按需修改)



typedef struct {
    int *elem;              //存放动态申请空间的首地址
    int length;             //记录当前长度
    int listsize;           //当前分配的元素的个数
} sqlist;
```

```
/* 相当于两步
   1、先定义结构体类型
   2、用typedef声明为新类型 */
struct _sqlist_ {
    int *elem;
    int length;
    int listsize;
};

typedef struct _sqlist_ sqlist;
```

/* linear_list_sq.h 的组成 */

```
#define LIST_INIT_SIZE      100
#define LISTINCREMENT       10
typedef struct {
    int *elem;
    int length;
    int listsize;
} sqlist;
```

```
Status    InitList(sqlist *L);
Status    DestroyList(sqlist *L);
Status    ClearList(sqlist *L);
Status    ListEmpty(sqlist L);
int       ListLength(sqlist L);
Status    GetElem(sqlist L, int i, int *e);
int       LocateElem(sqlist L, int e, Status (*compare)(int e1, int e2));
Status    PriorElem(sqlist L, int cur_e, int *pre_e);
Status    NextElem(sqlist L, int cur_e, int *next_e);
Status    ListInsert(sqlist *L, int i, int e);
Status    ListDelete(sqlist *L, int i, int e);
Status    ListTraverse(sqlist L, Status (*visit)(int e));
```

ADT List {
数据对象：D = { a_i | a_i ∈ ElemSet, i = 1,2,…,n, n≥0 }
数据关系：Rl = { <a_{i-1},a_i> | a_{i-1},a_i ∈ D, i = 2,…,n }
基本操作：

InitList(&L)
操作结果：构造一个空的线性表 L。

DestroyList(&L)
初始条件：线性表 L 已存在。
操作结果：销毁线性表 L。

ClearList(&L)
初始条件：线性表 L 已存在。
操作结果：将 L 重置为空表。

ListEmpty(L)
初始条件：线性表 L 已存在。
操作结果：若 L 为空表，则返回 TRUE，否则返回 FALSE。

ListLength(L)
初始条件：线性表 L 已存在。
操作结果：返回 L 中数据元素个数。

GetElem(L, i, &e)
初始条件：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。

LocateElem(L, e, compare())
初始条件：线性表 L 已存在，compare()是数据元素判定函数。
操作结果：返回 L 中第 1 个与 e 满足关系 compare()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。

Status InitList(sqlist *L);

算法转程序时：
★ 引用都表示为指针(C无引用)
★ 每个形参都要有类型定义

★ P.19-20 抽象数据类型定义转换为实际的C语言定义的函数原型说明
- 引用都表示为指针
- 每个形参都要有类型定义，其中compare和visit区别较大

问：当类型是double时，
需要做什么改变？

```
/* linear_list_sq.h 的组成 */
#define LIST_INIT_SIZE      100 //初始大小为100(可按需修改)
#define LISTINCREMENT       10  //空间分配增量(可按需修改)

typedef struct {
    double *elem;           //存放动态申请空间(当数组用)的首地址
    int length;             //记录当前长度
    int listsize;           //当前分配的元素的个数
} sqlist;

Status   InitList(sqlist *L);
Status   DestroyList(sqlist *L);
Status   ClearList(sqlist *L);
Status   ListEmpty(sqlist L);
int      ListLength(sqlist L);
Status   GetElem(sqlist L, int i, double *e);
int      LocateElem(sqlist L, double e, Status (*compare)(double e1, double e2));
Status   PriorElem(sqlist L, double cur_e, double *pre_e);
Status   NextElem(sqlist L, double cur_e, double *next_e);
Status   ListInsert(sqlist *L, int i, double e);
Status   ListDelete(sqlist *L, int i, double *e);
Status   ListTraverse(sqlist L, Status (*visit)(double e));
```

为什么不用换？

问：当类型是char[]时，
需要做什么改变？

```
/* linear_list_sq.h 的组成 */
#define LIST_INIT_SIZE      100 //初始大小为100(可按需修改)
#define LISTINCREMENT       10  //空间分配增量(可按需修改)

typedef struct {
    char (*elem)[10];       //存放动态申请空间(当数组用)的首地址
    int length;             //记录当前长度
    int listsize;           //当前分配的元素的个数
} sqlist;

Status   InitList(sqlist *L);
Status   DestroyList(sqlist *L);
Status   ClearList(sqlist *L);
Status   ListEmpty(sqlist L);
int      ListLength(sqlist L);
Status   GetElem(sqlist L, int i, char *e);
int      LocateElem(sqlist L, char *e, Status (*compare)(char *e1, char *e2));
Status   PriorElem(sqlist L, char *cur_e, char *pre_e);
Status   NextElem(sqlist L, char *cur_e, char *next_e);
Status   ListInsert(sqlist *L, int i, char *e);
Status   ListDelete(sqlist *L, int i, char *e);
Status   ListTraverse(sqlist L, Status (*visit)(char *e));
```

```
double cur_e, double *pre_e);
main:
double d1=5.2, d2;
PriorElem(L,  d, &d2);

main:
char s1[10], s2[10];
PriorElem(L, s1,  s2);

问题1：调用方式不一致，是否正确？
```

问：当类型是char[]时，
需要做什么改变？

```
/* linear_list_sq.h 的组成 */
#define LIST_INIT_SIZE      100 //初始大小为100(可按需修改)
#define LISTINCREMENT       10  //空间分配增量(可按需修改)

typedef struct {
    char (*elem)[10];       //存放动态申请空间(当数组用)的首地址
    int length;             //记录当前长度
    int listsize;           //当前分配的元素的个数
} sqlist;

Status   InitList(sqlist *L);
Status   DestroyList(sqlist *L);
Status   ClearList(sqlist *L);
Status   ListEmpty(sqlist L);
int      ListLength(sqlist L);
Status   GetElem(sqlist L, int i, char (*e)[10]);
int      LocateElem(sqlist L, char *e, Status (*compare)(char *e1, char *e2));
Status   PriorElem(sqlist L, char *cur_e, char (*pre_e)[10]);
Status   NextElem(sqlist L, char *cur_e, char (*next_e)[10]);
Status   ListInsert(sqlist *L, int i, char *e);
Status   ListDelete(sqlist *L, int i, char (*e)[10]);
Status   ListTraverse(sqlist L, Status (*visit)(char *e));
```

```
double cur_e, double *pre_e);
main:
double d1=5.2, d2;
PriorElem(L,  d, &d2);

main:
char s1[10], s2[10];
PriorElem(L, s1, &s2);

问题2：若要求保持一致，如何做？
```

问：当类型是char *时，
需要做什么改变？

```
/* linear_list_sq.h 的组成 */
#define LIST_INIT_SIZE      100 //初始大小为100(可按需修改)
#define LISTINCREMENT       10  //空间分配增量(可按需修改)

typedef struct {
    char **elem;            //存放动态申请空间(当数组用)的首地址
    int length;             //记录当前长度
    int listsize;           //当前分配的元素的个数
} sqlist;

Status   InitList(sqlist *L);
Status   DestroyList(sqlist *L);
Status   ClearList(sqlist *L);
Status   ListEmpty(sqlist L);
int      ListLength(sqlist L);
Status   GetElem(sqlist L, int i, char **e);
int      LocateElem(sqlist L, char *e, Status (*compare)(char *e1, char *e2));
Status   PriorElem(sqlist L, char *cur_e, char **pre_e);
Status   NextElem(sqlist L, char *cur_e, char **next_e);
Status   ListInsert(sqlist *L, int i, char *e);
Status   ListDelete(sqlist *L, int i, char **e);
Status   ListTraverse(sqlist L, Status (*visit)(char *e));
```

问：当类型是struct student时，需要做什么改变？

注：纯C编译器，struct student e 不能写成 student e

```
/* linear_list_sq.h 的组成 */
#define LIST_INIT_SIZE      100 //初始大小为100(可按需修改)
#define LISTINCREMENT       10  //空间分配增量(可按需修改)
struct student {
    ...
};
typedef struct {
    struct student *elem; //存放动态申请空间(当数组用)的首地址
    int length;           //记录当前长度
    int listsize;         //当前分配的元素的个数
} sqlist;
Status   InitList(sqlist *L);
Status   DestroyList(sqlist *L);
Status   ClearList(sqlist *L);
Status   ListEmpty(sqlist L);
int      ListLength(sqlist L);
Status   GetElem(sqlist L, int i, struct student *e);
int      LocateElem(sqlist L, struct student e, Status (*compare)( struct student e1, struct student e2));
Status   PriorElem(sqlist L, struct student cur_e, struct student *pre_e);
Status   NextElem(sqlist L, struct student cur_e, struct student *next_e);
Status   ListInsert(sqlist *L, int i, struct student e);
Status   ListDelete(sqlist *L, int i, struct student *e);
Status   ListTraverse(sqlist L, Status (*visit)(struct student e));
```

问：当类型是struct student *时，需要做什么改变？

注：纯C编译器，struct student e 不能写成 student e

```
/* linear_list_sq.h 的组成 */
#define LIST_INIT_SIZE      100 //初始大小为100(可按需修改)
#define LISTINCREMENT       10  //空间分配增量(可按需修改)
struct student {
    ...
};
typedef struct {
    struct student **elem; //存放动态申请空间(当数组用)的首地址
    int length;            //记录当前长度
    int listsize;          //当前分配的元素的个数
} sqlist;
Status   InitList(sqlist *L);
Status   DestroyList(sqlist *L);
Status   ClearList(sqlist *L);
Status   ListEmpty(sqlist L);
int      ListLength(sqlist L);
Status   GetElem(sqlist L, int i, struct student **e);
int      LocateElem(sqlist L, struct student *e, Status (*compare)( struct student *e1, struct student *e2));
Status   PriorElem(sqlist L, struct student *cur_e, struct student **pre_e);
Status   NextElem(sqlist L, struct student *cur_e, struct student **next_e);
Status   ListInsert(sqlist *L, int i, struct student *e);
Status   ListDelete(sqlist *L, int i, struct student **e);
Status   ListTraverse(sqlist L, Status (*visit)(struct student *e));
```

```
/* linear_list_sq.h 的组成 */
#define LIST_INIT_SIZE      100 //初始大小为100(可按需修改)
#define LISTINCREMENT       10  //空间分配增量(可按需修改)


typedef struct {
    int *elem;     //存放动态申请空间(当数组用)的首地址
    int length;    //记录当前长度
    int listsize; //当前分配的元素的个数
} sqlist;
```

问：当类型是
int
double
char[]
char *
struct student
struct student *
时，能否使改动尽可能少？

答：在数据结构中一般不讨论具体类型，引入一个通用类型（Elemtype）来表示元素的类型即可

```
// - - - - - 线性表的动态分配顺序存储结构 - - - - -
#define  LIST_INIT_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(以 sizeof(ElemType)为单位)
}SqList;
```

P. 22

```
/* linear_list_sq.h 的组成 */
#define LIST_INIT_SIZE      100 //初始大小为100(可按需修改)
#define LISTINCREMENT       10  //空间分配增量(可按需修改)

typedef int ElemType;       //算法到程序的补充

typedef struct {
    ElemType *elem;         //存放动态申请空间的首地址
    int length;             //记录当前长度
    int listsize; //当前分配的元素的个数
} sqlist;

Status   InitList(sqlist *L);
Status   DestroyList(sqlist *L);
Status   ClearList(sqlist *L);
Status   ListEmpty(sqlist L);
int      ListLength(sqlist L);
Status   GetElem(sqlist L, int i, ElemType *e);
int      LocateElem(sqlist L, ElemType e, Status (*compare)(ElemType e1, ElemType e2));
Status   PriorElem(sqlist L, ElemType cur_e, ElemType *pre_e);
Status   NextElem(sqlist L, ElemType cur_e, ElemType *next_e);
Status   ListInsert(sqlist *L, int i, ElemType e);
Status   ListDelete(sqlist *L, int i, ElemType *e);
Status   ListTraverse(sqlist L, Status (*visit)(ElemType e));
```

问：当类型不同时，能否使改动尽可能少？

答：在数据结构中一般不讨论具体类型，引入一个通用类型（Elemtype）来表示元素的类型即可

问：算法转为程序时，如何对应实际类型？

答：用typedef声明新类型的方法来实现实际类型和通用类型间的映射

/* linear_list_sq.h 的组成 */

```
struct student {
    int   num;      //设学号为主关键字
    char  name[10];
    char  sex;
    float score;
    char  addr[30];
}   //算上填充，共52字节
```

问：当类型是
int
double
char[]
char *
struct student
struct student *
时，需要做什么改变？

答：实际使用时，6选1即可(只能打开其中一项，否则错)，函数声明部分不同类型完全一致

```
//typedef int ElemType;
typedef double ElemType;
//typedef char ElemType[10];
//typedef char* ElemType;
//typedef struct student ElemType;
//typedef struct student* ElemType;
```

```
typedef struct {
    ElemType *elem;
    int length;
    int listsize;
} sqlist;
```

不同类型一致，无任何变化

```
Status      InitList(sqlist *L);
Status      DestroyList(sqlist *L);
Status      ClearList(sqlist *L);
Status      ListEmpty(sqlist L);
int         ListLength(sqlist L);
Status      GetElem(sqlist L, int i, ElemType *e);
int         LocateElem(sqlist L, ElemType e,
                       Status (*compare)(ElemType e1, ElemType e2));
Status      PriorElem(sqlist L, ElemType cur_e, ElemType *pre_e);
Status      NextElem(sqlist L, ElemType cur_e, ElemType *next_e);
Status      ListInsert(sqlist *L, int i, ElemType e);
Status      ListDelete(sqlist *L, int i, ElemType *e);
Status      ListTraverse(sqlist L, Status (*visit)(ElemType e));
```

§2.线性表

2.2.线性表的顺序表示和实现

2.2.2.线性表顺序表示的基本操作的实现

2.2.2.1.C语言版

★ 程序的组成

★ 与书上算法的区别

- C语言无引用，需要用指针代替
- 临时变量算法中无定义，程序要补齐
- 某些形式化定义和实际表示之间有区别

★ linear_list_sq.h 中各定义项的解析

★ linear_list_sq.c 中各函数的具体实现

ElemType => int

```
/* linear_list_sq.c 的实现 */
#include <stdio.h>
#include <stdlib.h>                    //malloc/realloc函数
#include <unistd.h>                    //exit函数
#include "linear_list_sq.h"            //形式定义

/* 初始化线性表 */
Status InitList(sqlist *L)
{
    L->elem = (ElemType *)malloc(LIST_INIT_SIZE * sizeof(ElemType));
    if (L->elem == NULL)
        exit(OVERFLOW);
    L->length = 0;
    L->listsize = LIST_INIT_SIZE;
    return OK;
}
```

★ main函数中
声明为 sqlist L;
调用为 InitList(&L);

★ 形参为指针，因为函数中要改变并返回
★ 书上为引用，因此 L.elem形式应变为 L->elem形式

ElemType => int

```
/* linear_list_sq.c 的实现 */
#include <stdio.h>
#include <stdlib.h>                    //malloc/realloc函数
#include <unistd.h>                    //exit函数
#include "linear_list_sq.h"            //形式定义

/* 初始化线性表 */
Status InitList(sqlist L)
{
    L.elem = (ElemType *)malloc(LIST_INIT_SIZE * sizeof(ElemType));
    if (L.elem == NULL)
        exit(OVERFLOW);
    L.length = 0;
    L.listsize = LIST_INIT_SIZE;
    return OK;
}
```

★ main函数中
声明为 sqlist L;
调用为 InitList(L);

★ 形参为sqlist结构体变量，函数实现中
L-> 均改为 L. 是否正确？为什么？

```
/* linear_list_sq.c 的实现 */
/* 初始化线性表 */
Status InitList(sqlist L)
{
    L.elem = (ElemType *)malloc(LIST_INIT_SIZE * sizeof(ElemType));
    if (L.elem == NULL)
        exit(OVERFLOW);
    L.length = 0;
    L.listsize = LIST_INIT_SIZE;
    return OK;
}
```

错误分析

★ main函数中
声明为 sqlist L;
调用为 InitList(L);

Step1：实参传形参

实参 L(12字节)	2000 2011	值未定 ???
形参 L(12字节)	2100 2111	值未定 ???

Step2：形参申请空间

实参 L(12字节)	2000 2011	值未定 ???
形参 L(12字节)	2100 2111	3000

错误：函数返回后，实参L
得不到申请的空间首址

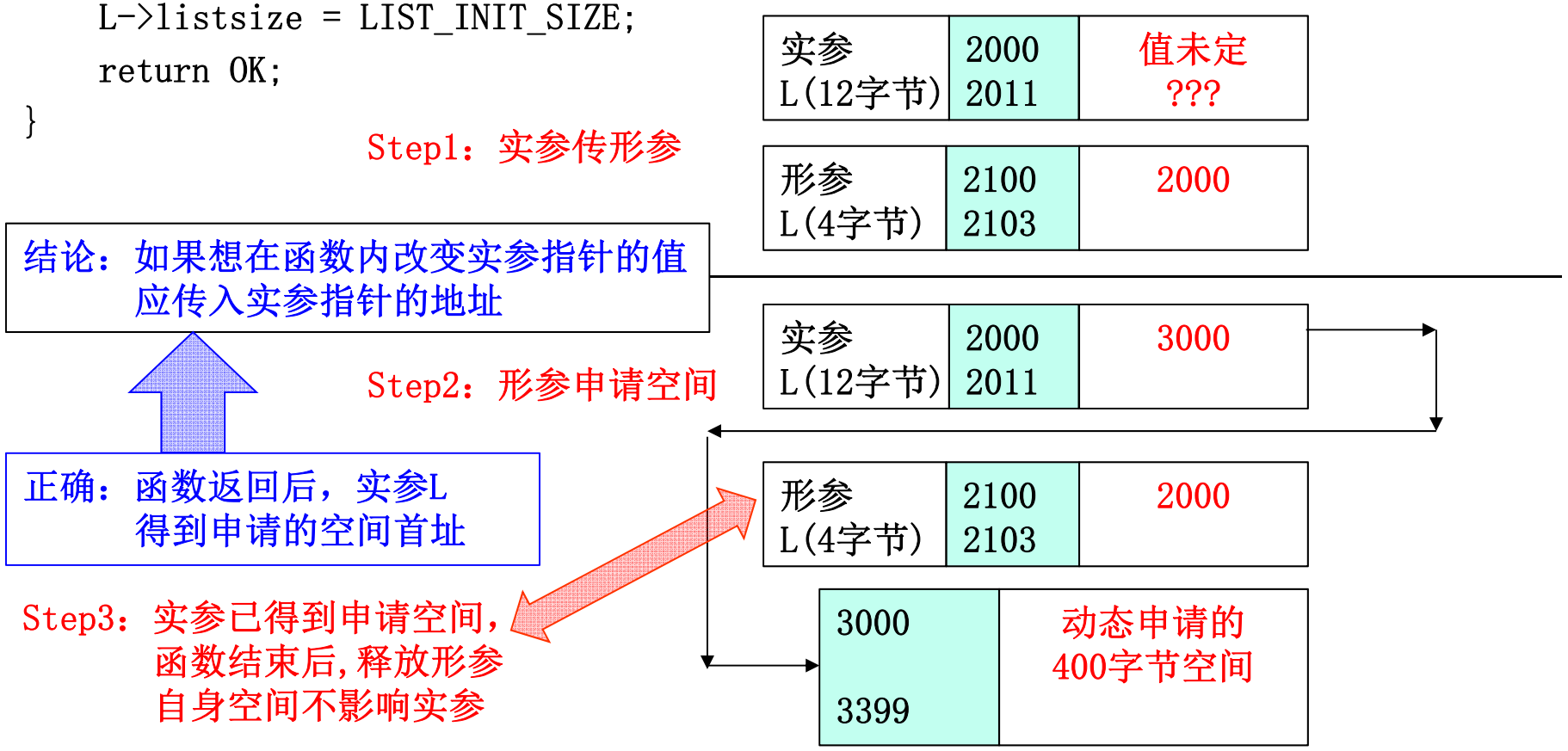
3000 3399	动态申请的 400字节空间

Step3：函数结束后，
释放形参自身空间，
实参并未得到申请空间

```
/* linear_list_sq.c 的实现 */
/* 初始化线性表 */
Status InitList(sqlist *L)
{
    L->elem = (ElemType *)malloc(LIST_INIT_SIZE * sizeof(ElemType));
    if (L->elem == NULL)
        exit(OVERFLOW);
    L->length = 0;
    L->listsize = LIST_INIT_SIZE;
    return OK;
}
```

正确分析

★ main函数中
声明为 sqlist L;
调用为 InitList(&L);



```
/* linear_list_sq.c 的实现 */
#include <stdio.h>
#include <stdlib.h>                    //malloc/realloc函数
#include <unistd.h>                    //exit函数
#include "linear_list_sq.h"            //形式定义

/* 初始化线性表 */
Status InitList(sqlist *L)
{
    L->elem = (ElemType *)malloc(LIST_INIT_SIZE * sizeof(ElemType));
    if (L->elem == NULL)
        exit(OVERFLOW);
    L->length = 0;
    L->listsize = LIST_INIT_SIZE;
    return OK;
}
```

★ main函数中
声明为 sqlist L;
调用为 InitList(&L);

★ 形参为指针，因为函数中要改变并返回
★ 书上为引用，因此 L.elem形式应变为 L->elem形式

问：当类型是
double
char[]
char *
struct student
struct student *
时，需要做什么改变？

答：不需要任何变化!!!

ElemType => int

```
/* linear_list_sq.c 的实现 */

/* 删除线性表 */
Status DestroyList(sqlist *L)
{
    /* 未执行 InitList，直接执行本函数，
       则可能出错，因为指针初值未定 */
    if (L->elem)
        free(L->elem);
    L->length   = 0; //可以不要
    L->listsize = 0; //可以不要

    return OK;
}
```

问：当类型是
double
char[]
char *
struct student
struct student *
时，需要做什么改变？

/* linear_list_sq.c 的实现 */

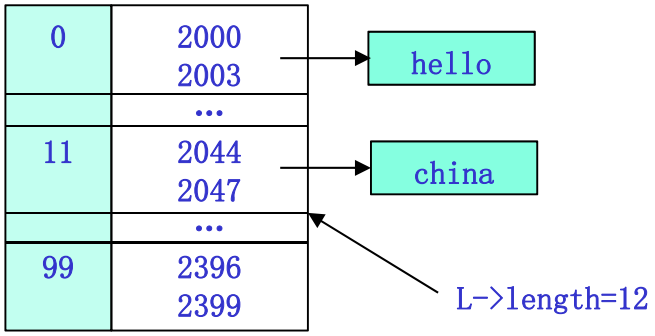
```
/* 删除线性表 */
Status DestroyList(sqlist *L)
{



    /* 未执行 InitList，直接执行本函数，
       则可能出错，因为指针初值未定 */
    if (L->elem)
            free(L->elem);
    L->length   = 0; //可以不要
    L->listsize = 0; //可以不要

    return OK;
}
```

```
/* 删除线性表 */
Status DestroyList(sqlist *L)
{
    int i;
    /* 首先释放二次申请空间 */
    for(i=0; i<L->length; i++)
        free(L->elem[i]);

    /* 未执行 InitList，直接执行本函数，
       则可能出错，因为指针初值未定 */
    if (L->elem)
            free(L->elem);
    L->length   = 0; //可以不要
    L->listsize = 0; //可以不要

    return OK;
}
```

问：当类型是
double
char[]
char *
struct student
struct student *
时，需要做什么改变？

答：当类型是
double
char[]
struct student时，
不需要任何变化!!!

答：当类型是
char *
struct student *时，
要首先释放二次申请空间



/* linear_list_sq.c 的实现 */

```
/* 删除线性表 */
Status DestroyList(sqlist *L)
{




    /* 未执行 InitList，直接执行本函数，
       则可能出错，因为指针初值未定 */
    if (L->elem)
            free(L->elem);
    L->length   = 0; //可以不要
    L->listsize = 0; //可以不要

    return OK;
}
```

```
/* 删除线性表 */
Status DestroyList(sqlist *L)
{
    int i;
    /* 首先释放二次申请空间 */
    for(i=0; i<L->length; i++)
        free(L->elem[i]);

    /* 未执行 InitList，直接执行本函数，
       则可能出错，因为指针初值未定 */
    if (L->elem)
            free(L->elem);
    L->length   = 0; //可以不要
    L->listsize = 0; //可以不要

    return OK;
}
```

类型为 char * 和 struct student *时，此处打开，其它类型时注释掉

问：当类型是
double
char[]
char *
struct student
struct student *
时，需要做什么改变？

问：如何处理具体类型不同时的代码差异？
答：按需注释/非注释

续问：有没有更好的方法？
续答：编译预处理 - 条件编译

请先去查看"条件编译"部分的课件

答：当类型是
double
char[]
struct student时，
不需要任何变化!!!

答：当类型是
char *
struct student *时，
要首先释放二次申请空间