

# Mr Bluyee's Blog

[🏠 首页](#)[📁 归档](#)[👤 关于](#)

## C实现线性表

📅 Jul 20, 2018 | 📖 学习笔记——C数据结构 | 📄 2 阅读 | 📖 1.8k 字 | ⌚ 9 分钟

线性表（Linear List）是最常用且最简单的一种数据结构。

[github源码](#)

抽象数据类型的定义如下：

ADT List {

数据对象：D = { | ∈ ElemSet, i=1,2,...,n, n≥0 }

数据关系：R1 = { | , ∈ 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。

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。

[文章目录](#)

操作结果：返回 L 中元素个数。

PriorElem( L, cur\_e, &pre\_e )

初始条件：线性表 L 已存在。

操作结果：若 cur\_e 是 L 中的数据元素，则用 pre\_e 返回它的前驱，  
否则操作失败，pre\_e 无定义。

NextElem( L, cur\_e, &next\_e )

初始条件：线性表 L 已存在。

操作结果：若 cur\_e 是 L 中的数据元素，则用 next\_e 返回它的后继，  
否则操作失败，next\_e 无定义。

GetElem( L, i, &e )

初始条件：线性表 L 已存在， $1 \leq i \leq \text{LengthList}(L)$ 。

操作结果：用 e 返回 L 中第 i 个元素的值。

LocateElem( L, e, compare( ) )

初始条件：线性表 L 已存在，compare( ) 是元素判定函数。

操作结果：返回 L 中第1个与 e 满足关系 compare( ) 的元素的位序。  
若这样的元素不存在，则返回值为0。

ListTraverse(L, visit( ))

初始条件：线性表 L 已存在，visit( ) 为元素的访问函数。

操作结果：依次对 L 的每个元素调用函数 visit( )。  
一旦 visit( ) 失败，则操作失败。

ClearList( &L )

初始条件：线性表 L 已存在。

操作结果：将 L 重置为空表。

PutElem( &L, i, &e )

初始条件：线性表 L 已存在， $1 \leq i \leq \text{LengthList}(L)$ 。

操作结果：L 中第 i 个元素赋值同 e 的值。

ListInsert( &L, i, e )

初始条件：线性表 L 已存在， $1 \leq i \leq \text{LengthList}(L)+1$ 。

操作结果：在 L 的第 i 个元素之前插入新的元素 e，L 的长度增1。

ListDelete( &L, i, &e )

初始条件：线性表 L 已存在且非空， $1 \leq i \leq \text{LengthList}(L)$ 。

操作结果：删除 L 的第 i 个元素，并用 e 返回其值，L 的长度减1。

} ADT List

对上述定义的抽象数据类型线性表，还可以进行一些更复杂的操作，例如取两个线性表的并集、交集和差集。

C实现的代码如下：

```
#include <stdio.h>
#include <malloc.h>

//线性表

#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(当存储空间不够时要用到)

typedef int ElemType;      //数据元素的类型，假设是int型的

typedef struct{
    ElemType *elem; //存储空间的基地址
    int length;      //当前线性表的长度
    int listsize;    //当前分配的存储容量
}LinearList;

int init_list(LinearList* list){
    list->elem = (ElemType *)malloc(LIST_INIT_SIZE * sizeof(ElemType));
    if (!list->elem){
        return -1; //空间分配失败
    }
    list->length = 0; //当前长度
    list->listsize = LIST_INIT_SIZE; //当前分配量
    return 0;
}

void clear_list(LinearList* list){
    list->length = 0; //当前长度
}
```

```

void destroy_list(LinearList* list){
    free(list);
}

int list_empty(LinearList* list){
    return (list->length == 0);
}

int list_length(LinearList* list){
    return list->length;
}

void print_list(LinearList* list){
    int i;
    for (i=0; i < list->length; i++){
        printf("%d ", list->elem[i]);
    }
    printf("\n");
}

int locate_elem(LinearList* list, ElemType* x){
    int pos = -1;
    for (int i = 0; i < list->length; i++){
        if (list->elem[i] == *x){
            pos = i;
        }
    }
    return pos;
}

int prior_elem(LinearList* list, ElemType* cur_elem, ElemType* pre_elem){
    int pos = -1;
    pos = locate_elem(list, cur_elem);
    if(pos <= 0) return -1;
    *pre_elem = list->elem[pos-1];
    return 0;
}

int get_elem(LinearList* list, int index, ElemType* e){
    if (index<0 || index >= list->length) return -1;
    *e = list->elem[index];
    return 0;
}

```

```

int next_elem(LinearList* list, ElemType* cur_elem, ElemType* next_elem){
    int pos = -1;
    pos = locate_elem(list, cur_elem);
    if(pos == -1 || pos == (list->length - 1)) return -1;
    *next_elem = list->elem[pos+1];
    return 0;
}

int insert_elem(LinearList* list, int index, ElemType* e){
    if (index<0 || index >= list->length) return -1;
    if (list->length >= list->listsize){ //判断存储空间是否够用
        ElemType *newbase = (ElemType *)realloc(list->elem, (list->listsize + LISTINCREMENT))
        if (!newbase) return -1;//存储空间分配失败
        list->elem = newbase;//新基址
        list->listsize += LISTINCREMENT;//增加存储容量
    }
    ElemType *q, *p;
    q = &(list->elem[index]); //q为插入位置
    for (p = &(list->elem[list->length - 1]); p >= q; --p){ //从ai到an-1依次后移，注意后移操
        *(p + 1) = *p;
    }
    *q = *e;
    ++list->length;
    return 0;
}

int delete_elem(LinearList* list, int index, ElemType* e)
{
    if (index<1 || index > list->length) return -1;
    ElemType *q, *p;
    p = &(list->elem[index]); //p为被删除元素的位置
    *e = *p; //被删除的元素赋值给e
    q = list->elem + list->length - 1; //q指向表尾最后一个元素
    for (++p; p <= q; ++p){ //从p的下一个元素开始依次前移
        *(p - 1) = *p;
    }
    --list->length;
    return 0;
}

int append_elem(LinearList* list, ElemType* e){
    if (list->length >= list->listsize){ //判断存储空间是否够用

```

```

        ElemType *newbase = (ElemType *)realloc(list->elem, (list->listsize + LISTINCREMENT))
        if (!newbase) return -1;//存储空间分配失败
        list->elem = newbase;//新基址
        list->listsize += LISTINCREMENT;//增加存储容量
    }
    list->elem[list->length] = *e;
    ++list->length;
    return 0;
}

int pop_elem(LinearList* list, ElemType* e){
    if (list_empty(list)) return -1;
    *e = list->elem[list->length - 1];
    --list->length;
    return 0;
}

void union_list(LinearList* list_a, LinearList* list_b, LinearList* list_c){ //并集,C=A∪B
    int i,a_length,b_length;
    ElemType elem;
    a_length = list_length(list_a);
    b_length = list_length(list_b);
    for(i=0;i<a_length;i++){
        get_elem(list_a, i, &elem);
        append_elem(list_c,&elem);
    }
    for(i=0;i<b_length;i++){
        get_elem(list_b, i, &elem);
        if(locate_elem(list_a, &elem) == -1){
            append_elem(list_c,&elem);
        }
    }
}

void intersect_list(LinearList* list_a, LinearList* list_b, LinearList* list_c){ //交集,C=A∩B
    int i,a_length;
    ElemType elem;
    a_length = list_length(list_a);
    for(i=0;i<a_length;i++){
        get_elem(list_a, i, &elem);
        if(locate_elem(list_b, &elem) != -1){
            append_elem(list_c,&elem);
        }
    }
}


```

```

    }
}

void except_list(LinearList* list_a, LinearList* list_b, LinearList* list_c){ //差集,C=A-B(属
    int i,a_length;
    ElemType elem;
    a_length = list_length(list_a);
    for(i=0;i<a_length;i++){
        get_elem(list_a, i, &elem);
        if(locate_elem(list_b, &elem) == -1){
            append_elem(list_c,&elem);
        }
    }
}
}

```



测试用例：

```

int main(void)
{
    int i;
    ElemType elem;
    LinearList *list_a = (LinearList *)malloc(sizeof(LinearList));
    LinearList *list_b = (LinearList *)malloc(sizeof(LinearList));
    LinearList *list_c = (LinearList *)malloc(sizeof(LinearList));
    init_list(list_a);
    init_list(list_b);
    init_list(list_c);

    for (i = 0; i < 10; i++){
        append_elem(list_a,&i);
    }

    for (i = 0; i < 20; i+=2){
        append_elem(list_b,&i);
    }
    print_list(list_a);
    print_list(list_b);

    pop_elem(list_a,&elem);
}

```

```

print_list(list_a);
printf("pop: %d \n",elem);

delete_elem(list_a, 2, &elem);
print_list(list_a);
printf("delete: %d \n",elem);

insert_elem(list_a, 2, &elem);
printf("insert: %d \n",elem);
print_list(list_a);

get_elem(list_a, 5, &elem);
printf("get elem at 5: %d \n",elem);

printf("locate : elem %d at %d \n",elem,locate_elem(list_a,&elem));

printf("list_a length : %d \n",list_length(list_a));

print_list(list_a);
print_list(list_b);

union_list(list_a,list_b,list_c);
print_list(list_c);
clear_list(list_c);

intersect_list(list_a,list_b,list_c);
print_list(list_c);
clear_list(list_c);

except_list(list_a,list_b,list_c);
print_list(list_c);

destroy_list(list_a);
destroy_list(list_b);
destroy_list(list_c);

return 0;
}

```

运行结果显示如下：



```
0 1 2 3 4 5 6 7 8 9
0 2 4 6 8 10 12 14 16 18
0 1 2 3 4 5 6 7 8
pop: 9
0 1 3 4 5 6 7 8
delete: 2
insert: 2
0 1 2 3 4 5 6 7 8
get elem at 5: 5
locate : elem 5 at 5
list_a length : 9
0 1 2 3 4 5 6 7 8
0 2 4 6 8 10 12 14 16 18
0 1 2 3 4 5 6 7 8 10 12 14 16 18
0 2 4 6 8
1 3 5 7
```

Donate

--	--

**本文作者：**Mr Bluyee

**本文链接：**<https://www.mrbluyee.com/2018/07/20/C实现线性表/>

**版权声明：**The author owns the copyright, please indicate the source reproduced.

撰写评论

发布

账号（邮件地址）

还没有评论，快来抢沙发吧！

Search

📁 分类

- 学习笔记——C 算法
- 学习笔记——C数据结构
- 学习笔记——Python
- 学习笔记——android
- 学习笔记——expert c programming
- 学习笔记——linux

学习笔记——opencv

学习笔记——嵌入式开发

学习笔记——机器学习

学习笔记——网络协议

## ☆ 标签

---

[android](#) [C](#) [网络协议](#) [linux](#) [嵌入式开发](#) [Python](#) [opencv](#) [机器学习](#)

## 📄 最近文章

---

[linux解压缩命令](#)

[linux查找命令](#)

[Little Kernel 04](#)

[Little Kernel 03](#)

[Little Kernel 02](#)

[Little Kernel 01](#)

[消息摘要算法](#)

[C按位操作实现CRC计算算法](#)

[CRC循环冗余校验算法](#)

[链表的反转](#)

## 🔗 友情链接

---

[人生的小站](#)

