

# 第一章：数据结构的基本理论

---

## 1.数据结构：

---

是相互之间存在一种或者多种特定关系的数据元素的集合。

## 2.基本的数据结构分为：

---

线性结构：线性表、栈和队列、串、多维数组和广义表

非线性结构：树、图。

## 3.数据的逻辑结构：

---

线性结构、树状结构、网状结构、集合。

## 4.数据的存储结构：

---

顺序存储结构（数组）、链式存储结构（链表）。

# 第二章：算法

---

## 1.算法的五个基本特性：

---

输入性、输出性、有穷性、确定性、可行性。

## 2.算法时间复杂度

---

## 3.算法空间复杂度

---

# 第三章：线性表

---

# 第一部分：线性表的顺序存储结构

## 1.有关线性表顺序存储结构的操作

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

//线性表顺序存储
#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，它的值是函数结果状态代码，比如ok
typedef int ElemType; //ElemType类型自己根据需求定义，这里是int

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

//打印函数
Status visit(ElemType c)
{
    printf("%d ", c);
    return OK;
}

//初始化顺序线性表
```

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

//判断空表

/\* 初始条件：顺序线性表L已经存在

操作结果：若L是空表，返回TRUE；否则返回FALSE。

\*/

```
Status ListEmpty(SqList L){
    if(L.length==0){
        return TRUE;
    }
    else{
        return FALSE;
    }
}
```

/\* 初始条件：顺序线性表L已存在。操作结果：将L重置为空表 \*/

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

//SqList \*L 和 SqList L 的区别是：\*L表示要修改内容， L表示只是访问，不修改内容

//返回元素个数

//初始条件：顺序线性表L已经存在，操作结果：返回L中元素的个数

```
int ListLength(SqList L){
    return L.length;
}
```

//返回元素的值

//初始条件：L存在

//操作结果：用e返回L中第i个元素的值，！！i指的是位置，第一个位置上的数组是从0开始的

```
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;
}
```

//返回元素的位置

//操作结果：返回L中某个元素e的位置

```
int LocateElem(SqList L,ElemType e){
    int i;
    if(L.length==0){
        return 0;
    }
    if(i>=L.length){
        return 0;
    }
    for (i = 0; i < L.length;i++){
        if(L.data[i]==e){
            break;
        }
    }
    return i + 1;
}
```

//插入数据

//操作结果：在L中第i个位置之前插入新的数据元素e；L的长度增加1

```
Status ListInsert(SqList *L, int i,ElemType e){
    int j;
    if(L->length==MAXSIZE){
        return ERROR;
    }
}
```

```

        if(i<1 || i>L->length+1){
            return ERROR;
        }
        if(i<=L->length){
            for (j = L->length - 1; j>=i-1;j--){ //将要插入位置之后的元素向后移动一位
                L->data[j + 1] = L->data[j];
            }
        }
        L->data[i - 1] = e; //将新元素插入
        L->length++;

        return OK;
    }

```

//删除元素

//操作结果：删除L的第i个元素，并用e返回它的值，L的长度减1

Status ListDelete(SqList \*L ,int i,ElemType \*e){

```

    int j;
    if(L->length==0){
        return ERROR;
    }
    if(i<1 || i>L->length){
        return ERROR;
    }
    *e = L->data[i - 1];
    if(i<L->length){
        for (j = i; j < L->length;j++){
            L->data[j - 1] = L->data[j];
        }
    }
    L->length--;
    return OK;
}

```

//输出线性表

```

Status ListTraverse(SqList L){
    int i;
    for (i = 0; i < L.length;i++){
        visit(L.data[i]);
    }
    printf("\n");
    return OK;
}

//合并
//实现两个线性表集合A和B的并及操作
//也就是把存在集合B中但是并不存在A中的数据元素 插入到A中就可以了。

void unionL(SqList *La,SqList Lb){
    int La_len, Lb_len, i;
    ElemType e; //声明La和Lb是相同的数据元素
    La_len = ListLength(*La); //求线性表的长度
    Lb_len = ListLength(Lb);
    for (i = 1; i <= Lb_len;i++){
        GetElem(Lb, i, &e); //取Lb中第i个元素赋给e
        if(!LocateElem(*La,e)){ //La中不存在和e相同的
数据元素
            ListInsert(La, ++La_len, e); //插入
        }
    }
}

int main(){
    SqList L;
    SqList Lb;

    ElemType e;
    Status i;
    int j, k;
    i = InitList(&L);
    printf("初始化L后:  L.length=%d\n", L.length);
    for (j = 1; j <= 5;j++){

```

```

        i = ListInsert(&L, 1, j);
    }
    printf("在L的表 头依次插入1-5后:  L.data=");
    ListTraverse(L);

    printf("L.length=%d \n", L.length);
    i=ListEmpty(L);
    printf("L是否空: i=%d(1:是 0:否)\n", i);

    i=ClearList(&L);
    printf("清空L后: L.length=%d\n", L.length);
    i=ListEmpty(L);
    printf("L是否空: i=%d(1:是 0:否)\n", i);

    for(j=1; j<=10; j++)
        ListInsert(&L, j, j);
    printf("在L的表尾依次插入1~10后: L.data=");
    ListTraverse(L);

    printf("L.length=%d \n", L.length);

    ListInsert(&L, 1, 0);
    printf("在L的表头插入0后: L.data=");
    ListTraverse(L);
    printf("L.length=%d \n", L.length);

    GetElem(L, 5, &e);
    printf("第5个元素的值为: %d\n", e);
    for(j=3; j<=4; j++)
    {
        k=LocateElem(L, j);
        if(k)
            printf("第%d个元素的值为%d\n", k, j);
        else
            printf("没有值为%d的元素\n", j);
    }

```

```

k=ListLength(L); /* k为表长 */
for(j=k+1;j>=k;j--)
{
    i=ListDelete(&L,j,&e); /* 删除第j个数据 */
    if(i==ERROR)
        printf("删除第%d个数据失败\n",j);
    else
        printf("删除第%d个的元素值为：
%d\n",j,e);
}
printf("依次输出L的元素：");
ListTraverse(L);

j=5;
ListDelete(&L,j,&e); /* 删除第5个数据 */
printf("删除第%d个的元素值为： %d\n",j,e);

printf("依次输出L的元素：");
ListTraverse(L);

//构造一个有10个数的Lb
i=InitList(&Lb);
for(j=6;j<=15;j++)
    i=ListInsert(&Lb,1,j);

unionL(&L,Lb);

printf("依次输出合并了Lb的L的元素：");
ListTraverse(L);

return 0;
}

```



## 2.线性表顺序存储结构的优点和缺点

优点：

- 无须为表示表中元素之间的逻辑关系而增加额外的存储空间；
- 可以快速的存取表中任一位置的元素。

缺点：

- 插入和删除操作需要移动大量元素；
- 当线性表长度变化较大的时候，难以确定存储空间的容量；
- 造成存储空间的“碎片”

## 第二部分：线性表的链式存储结构

---

### 1.有关线性表链式存储结构的操作