数据结构
算法设计与实现

（C语言版）

王昱 著

科学出版社

数据结构算法设计与实现
（C语言版）

王 昱 著

科 学 出 版 社

北 京

内 容 简 介

本书以C语言设计与实现为数据结构的主要算法，内容包括线性表、栈和队列、串和数组、树、图、查找、排序等数据结构及相关操作，所有算法程序完全用纯C语言编写且均在VC++ 6.0下调试运行通过。

本书可作为计算机类专业或信息类专业“数据结构”课程的参考书，也可供软件编程人员和自学者参考。

图书在版编目（CIP）数据

数据结构算法设计与实现：C语言版 / 王昱著. —北京：科学出版社，2018.6

ISBN 978-7-03-057263-9

Ⅰ. ①数… Ⅱ. ①王… Ⅲ. ①数据结构 ②C语言–程序设计
Ⅳ. ①TP311.12 ②TP312.8

中国版本图书馆CIP数据核字（2018）第084368号

责任编辑：李淑丽 王迎春 / 责任校对：王 瑞
责任印制：吴兆东 / 封面设计：华路天然工作室

科学出版社出版
北京东黄城根北街16号
邮政编码：100717
http://www.sciencep.com

北京建宏印刷有限公司印刷
科学出版社发行 各地新华书店经销
*
2018年6月第 一 版 开本：787×1092 1/16
2018年6月第一次印刷 印张：15
字数：356 000

定价：89.00元

（如有印装质量问题，我社负责调换）

前　言

“数据结构”是计算机类专业和信息类专业的一门重要的专业基础课程，是介于数学、计算机硬件和计算机软件三者之间的一门核心课程，其在大学计算机课程体系中占据举足轻重的地位。在计算机科学中，数据结构不仅是一般程序设计的基础，而且是设计和实现编译程序、操作系统、数据库系统及其他系统程序和大型应用程序的重要基础。

本书以数据结构的主要算法为基础，以读者易于理解和掌握数据结构的算法设计与实现为目的，以C语言设计与实现算法为内容，主要包括线性表、栈和队列、串和数组、树、图、查找、排序等数据结构及相关操作内容。

本书完全用C语言设计与实现数据结构的主要算法，解决了算法设计与实现中C语言和C++语言混用给读者理解造成的困难，使读者理解和掌握算法设计与实现变得更加容易，书中所有算法程序均在VC++ 6.0下调试运行通过。

本书将抽象的算法变成具体的程序，使读者看到算法的执行结果，从而达到理解和掌握数据结构算法设计与实现的目的。

由于作者水平有限，书中难免存在不足之处，敬请读者批评指正。

作　者

2018年2月

目　录

第1章 线 性 表

1.1 线性表的定义及基本运算

1.1.1 线性表的定义

线性表是一种线性结构。线性结构的特点是数据元素之间是一种线性关系，数据元素“一个接一个地排列”。在一个线性表中数据元素的类型是相同的，或者说线性表是由同一类型的数据元素构成的线性结构。

线性表是具有相同数据类型的 $n(n\geq 0)$个数据元素的有限序列，通常记为

$$(a_1, a_2, \cdots, a_{i-1}, a_i, a_{i+1}, \cdots, a_n)$$

其中，n 为表长，$n=0$ 时为空表。

线性表中相邻元素之间存在顺序关系。将 a_{i-1} 称为 a_i 的直接前驱，a_{i+1} 称为 a_i 的直接后继。也就是说，对于 a_i，当 $i=2, \cdots, n$ 时，有且仅有一个直接前驱 a_{i-1}，当 $i=1, 2, \cdots, n-1$ 时，有且仅有一个直接后继 a_{i+1}，而 a_1 是表中第一个元素，没有前驱，a_n 是最后一个元素，没有后继。

需要说明的是，a_i 是序号为 $i(i=1, 2, \cdots, n)$的数据元素，通常我们将它的数据类型抽象为 DataType，DataType 根据具体问题而定，例如，在学生情况信息表中，它是用户自定义的学生类型；在字符串中，它是字符类型。

1.1.2 线性表的基本运算

1. 线性表初始化运算

初始条件：线性表 L 不存在。
操作结果：构造一个空的线性表 L。

2. 求线性表的长度运算

初始条件：线性表 L 存在。
操作结果：返回线性表 L 中所含元素的个数。

3. 取表元运算

初始条件：线性表 L 存在。

操作结果：返回线性表 L 中的第 i 个元素的值或地址，i 符合要求。

4. 按值查找运算

初始条件：线性表 L 存在。

操作结果：在线性表 L 中查找首次出现的值为 x 的元素，若查找成功，则返回该元素的序号或地址，否则返回一个特殊值，表示查找失败。

5. 插入运算

初始条件：线性表 L 存在。

操作结果：在线性表 L 的第 i 个位置上插入一个值为 x 的新数据元素，插入成功后使原序号为 $i, i+1, \cdots, n$ 的数据元素的序号变为 $i+1, i+2, \cdots, n+1$，表长变为原表长+1，i 符合要求。

6. 删除运算

初始条件：线性表 L 存在。

操作结果：在线性表 L 中删除序号为 i 的数据元素，删除成功后使序号为 $i+1, i+2, \cdots, n$ 的元素变为序号为 $i, i+1, \cdots, n-1$，表长变为原表长–1，i 符合要求。

7. 输出运算

初始条件：线性表 L 存在。

操作结果：输出线性表 L 中所有的数据元素。

需要说明的是，某数据结构上的基本运算并不是它的全部运算，而是一些常用的基本运算，而每一个基本运算在实现时也可能根据不同的存储结构派生出一系列相关的运算。

在上面各操作中定义的线性表 L 仅仅是一个抽象在逻辑结构层次的线性表，尚未涉及它的存储结构。

1.2　线性表的顺序存储结构

1.2.1　顺序表定义

```
#define MAXSIZE 100                /*顺序表空间的存储量*/
typedef int DataType;              /*顺序表元素类型*/
typedef  struct{
    DataType *data;                /*顺序表元素存储空间基址*/
    int length;                    /*顺序表当前长度*/
}SeqList;                          /*顺序表类型*/
```

1.2.2 顺序表基本运算

1. 初始化运算

```
/*初始化,构造一个空顺序表*/
void Init_SeqList(SeqList *L,int n)
```

2. 插入元素运算

```
/*插入元素,在顺序表第 i 个位置之前插入元素 x,即插入在第 i 个位置*/
int Insert_SeqList(SeqList *L,int i,DataType x)
```

3. 删除元素运算

```
/*删除元素,将顺序表第 i 个位置的元素删除*/
int Delete_SeqList(SeqList *L,int i)
```

4. 按值查找元素运算

```
/*按值查找元素,在顺序表中查找元素 x,若查找成功,则返回 x 的位置序号,若查找失败,则返回-1*/
int Locate_SeqList(SeqList *L, DataType x)
```

5. 输出顺序表运算

```
/*输出顺序表*/
void Display_SeqList(SeqList *L)
```

1.2.3 顺序表算法设计与实现

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#define MAXSIZE 100                    /*顺序表空间的存储量*/
typedef int DataType;                  /*顺序表元素类型*/
typedef  struct{
    DataType *data;                    /*顺序表元素存储空间基址*/
    int  length;                       /*顺序表当前长度*/
}SeqList;                              /*顺序表类型*/
void wait()
{
```

```
    printf("\n 请按任意键...\n");
    getch();
}
int go_on()
{
    int flag=1;
    char choice;
    while(1){
        printf("\n 继续吗? [Y/N]");
        choice=getche();
        if(choice=='Y' || choice=='y')
            break;
        else if(choice=='N' || choice=='n'){
            flag=0;
            break;
        }
    }
    return(flag);
}
/*初始化,构造一个空顺序表*/
void Init_SeqList(SeqList *L,int n)
{
    L->data=(DataType *)malloc(n*sizeof(DataType));
    if(!L->data){
        printf("\n 内存分配失败.\n");
        exit(-1);
    }
    L->length=0;
}
/*插入元素,在顺序表第 i 个位置之前插入元素 x,即插入在第 i 个位置*/
int  Insert_SeqList(SeqList *L,int i,DataType x)
{
    DataType *p,*q;
    if(L->length == MAXSIZE){          /*表满,不能插入*/
        printf("\n 表满,不能插入.\n");
        return(-1);                    /*不能插入,返回-1*/
    }
    if(i<1 || i>L->length+1){          /*插入位置错,不能插入*/
```

```
        printf("\n 插入位置错,不能插入.\n");
        return(0);                    /*插入失败,返回 0*/
    }
    q=&(L->data[i-1]);                /*q 指向插入位置*/
    for(p=&(L->data[L->length-1]);p>=q;p--) /*元素向后移动*/
        *(p+1)=*p;
    *q=x;                             /*插入元素 x*/
    L->length++;                      /*表当前长度增加 1*/
    return(1);                        /*插入成功,返回 1*/
}
void Insert(SeqList *L)
{
    DataType x;
    int i,flag=1,insert_flag;
    while(flag){
        printf("\n 请输入要插入元素的位置: ");
        scanf("%d",&i);
        printf("请输入要插入元素:");
        scanf("%d",&x);
        insert_flag=Insert_SeqList(L,i,x);
        if(insert_flag==1)
            printf("\n 插入成功.\n");
        else
            printf("\n 插入失败.\n");
        flag=go_on();
    }
}
/*删除元素,将顺序表第 i 个位置元素删除*/
int Delete_SeqList(SeqList *L,int i)
{
    DataType *p,*q;
    if(L->length==0){                 /*表空,不能删除*/
        printf("\n 表空,不能删除.\n");
        return(-1);                   /*删除失败,返回-1*/
    }
    if(i<1 || i>L->length){           /*删除位置错,不能删除*/
        printf ("\n 删除位置错,不能删除.\n");
        return(0);                    /*删除失败,返回 0*/
```

```
    }
    q=&(L->data[L->length-1]);          /*q指向最后一个元素位置*/
    for(p=&(L->data[i]);p<=q;p++)       /*元素向前移动*/
        *(p-1)=*p;
    L->length--;                        /*表当前长度减1*/
    return(1);                          /*删除成功,返回1*/
}
void Delete(SeqList *L)
{
    int i,flag=1,delete_flag;
    while(flag){
        printf("\n请输入要删除元素的位置: ");
        scanf("%d",&i);
        delete_flag=Delete_SeqList(L,i);
        if(delete_flag==1)
            printf("\n删除成功.\n");
        else
            printf("\n删除失败.\n");
        flag=go_on();
    }
}
/*按值查找元素,在顺序表中查找元素x,若查找成功,则返回x的位置序号,若查找
失败,返回-1*/
int Locate_SeqList(SeqList *L, DataType x)
{
    DataType *p;
    int i;
    i=0;
    p=L->data;
    while(i<=L->length-1 && *p!= x){
        i++;
        p++;
    }
    if(i>L->length-1)
        return(-1);                     /*查找失败,返回-1*/
    else
        return(i+1);                    /*查找成功,返回x的位置序号*/
}
```

```
void Locate(SeqList *L)
{
     DataType x;
     int flag=1,locate_flag;
     while(flag){
          printf("\n请输入要查找元素: ");
          scanf("%d",&x);
          locate_flag=Locate_SeqList(L,x);
          if(locate_flag>0)
               printf("\n查找成功,%d 是第%d个元素.\n",x,locate_
               flag);
          else
               printf("\n查找失败,没有元素%d.\n",x);
         flag=go_on();
     }
}
/*输出顺序表*/
void Display_SeqList(SeqList *L)
{
     int i;
     printf("\n顺序表全部元素\n");
     for(i=0;i<=L->length-1;i++)
          printf("%4d",L->data[i]);
     printf("\n");
}
main()
{
     SeqList L;
     char choice;
     int flag=1;
     Init_SeqList(&L,MAXSIZE);
     do{
        printf("\n");
        printf("----顺序表(动态数组实现)----\n");
        printf("    1.......插入元素\n");
        printf("    2.......删除元素\n");
        printf("    3.......查找元素\n");
        printf("    4.......输出元素\n");
```

```
        printf("    0.......退出\n");
        printf("------------------------------\n");
        printf("请选择[1/2/3/4/0]:");
        choice=getche();
        switch(choice){
            case '1':Insert(&L);break;
            case '2':Delete(&L);break;
            case '3':Locate(&L);break;
            case '4':Display_SeqList(&L);break;
            case '0':flag=0;break;
        }
        wait();
    }while(flag==1);
}
```

1.3 线性表的链式存储结构

1.3.1 单链表

1. 单链表定义

```
typedef int DataType;          /*单链表元素类型*/
typedef struct node{
   DataType data;              /*单链表元素*/
   struct node *next;          /*单链表元素后继指针*/
}LNode,*LinkList;              /*单链表结点类型、单链表类型*/
```

2. 单链表基本运算

1）初始化运算

```
/*初始化,构造一个空的带头结点的单链表*/
void Init_LinkList(LinkList *L)
```

2）判断单链表空运算

```
/*判断带头结点单链表是否为空*/
int Empty_LinkList(LinkList L)
```

3）求单链表长度运算

```
/*求带头结点单链表长度*/
int Length_LinkList(LinkList L)
```

4）按序号查找元素运算

```
/*按序号查找元素,在带头结点的单链表中查找第 i 个元素,若查找成功,则返回指
```

向第 i 个元素结点的指针,否则返回空*/

```
LNode *Locatei_LinkList(LinkList L,int i)
```

5）按值查找元素运算

```
/*按值查找元素,在带头结点的单链表中查找元素值为 x 的第 1 个元素,若查找成功,
则返回指向 x 所在结点的指针,*k 返回其位置序号,否则返回空,*k 无意义*/
LNode *Locatex_LinkList(LinkList L,int x,int *k)
```

6）插入元素运算

```
/*插入元素,在带头结点单链表中第 i 个位置之前插入元素 x,即插入在第 i 个位置*/
int Insert_LinkList(LinkList L,int i,DataType x)
```

7）删除元素运算

```
/*删除元素,在带头结点单链表中删除第 i 个位置的元素*/
int Delete_LinkList(LinkList L,int i)
```

8）输出单链表运算

```
/*输出带头结点单链表中的所有数据元素*/
void Display_LinkList(LinkList L)
```

3. 单链表算法设计与实现

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
typedef int DataType;                    /*单链表元素类型*/
typedef struct node{
    DataType data;                       /*单链表元素*/
    struct node *next;                   /*单链表元素后继指针*/
}LNode,*LinkList;                        /*单链表结点类型、单链表类型*/
void wait()
{
     printf("\n 请按任意键...\n");
     getch();
}
int go_on()
{
    int flag=1;
    char choice;
    while(1){
         printf("\n 继续吗? [Y/N]");
         choice=getche();
         if(choice=='Y' || choice=='y')
```

```
                break;
        else if(choice=='N' || choice=='n'){
                flag=0;
                break;
        }
    }
    return(flag);
}
/*初始化,构造一个空的带头结点的单链表*/
void Init_LinkList(LinkList *L)
{
     *L=(LNode *)malloc(sizeof(LNode));
     if(*L==NULL){
          printf("\n 内存分配失败.\n");
          exit(-1);
     }
     (*L)->next=NULL;
}
/*判断带头结点的单链表是否为空*/
int Empty_LinkList(LinkList L)
{
     if(L->next==NULL)
          return(1);
     else
          return(0);
}
/*求带头结点单链表长度*/
int Length_LinkList(LinkList L)
{
    LNode *p=L->next;
    int k;
    k=0;
    while(p!=NULL){
         k++;
         p=p->next;
    }
    return(k);
}
```