数据结构上机实验

连接逻辑结构与存储结构

编著：李金龙

组织：中国科学技术大学

时间：1:37, October 30, 2024

版本：0.1.9

用途：本科生《数据结构》实验指导

Can machines think? — Alan Mathison Turing

目录

第一部分

基础知识

第一章 数据结构学习内容概要

内容提要

- 数据结构课程学什么
- 问题建模与逻辑结构
- 存储结构与编程求解问题
- 复杂应用问题的求解算法

给定一个问题，用计算机编程求解，如何迈出第一步，是开始学习计算机程序设计的关键，也是很多学生学计算机面临的第一个困难。如果这第一步跨不出去，可能会让学生丧失学习程序设计的信心和兴趣。数据结构这门课将帮助学生逐步认识程序设计的主要步骤和方法，了解各种程序工作的基本过程和原理，为进一步学习计算机专业的后续课程奠定基础。

1.1 数据结构课程学什么

1.2 问题建模与逻辑结构

1.3 存储结构与编程求解问题

1.4 复杂应用问题的求解算法

第二章 数据结构上机实验常见错误解析(C语言代码)

内容提要

- 指针错误
- 逻辑错误
- 其它类型

说明：每种错误有一两句话来解释说明一下原因，给出例子，同时给出修改的方法和结果，参考第一个错误的写法。相同的错误，可以有多个例子。

2.1 指针错误

1. 调用函数scanf()从键盘输入数据时，实参是保存输入数据的变量的地址，例如

```
int a;
scanf(”%d”,a);
```

编译器显示的错误信息为：”warning: format ‘%d’ expects argument of type ‘int *’, but argument 2 has type ‘int’ ”，即函数scanf的格式串%d对应的参数是保存整数数据的变量地址（int *），但是对应的第二个参数的类型是整型int。所以，修改的方法就是在整型变量a之前应该添加’&’，改成‘scanf(”%d”,&a);’。

2. 下一个指针错误

2.2 逻辑错误

2.3 其它类型错误

第二部分

实验题

第三章 线性结构

内容提要

- 实验目的
- 实验内容
- 实验说明
- 检查/提交要求

3.1 实验目的

复习C程序设计语言知识

- 掌握指针的概念和使用：指针、函数指针、数组名、函数名等
- 预编译指令的使用
- 熟悉写代码的规范
- 掌握代码调试技术
- 学会绘制流程图

3.2 实验内容

- 实验内容一：线性表的链式实现
- 实验内容二：实现顺序栈

3.3 实验说明

1. 实验内容一：线性表的链式实现
   - 完成线性表抽象数据类型ADT的链式存储的设计与实现
   - 利用函数指针调用不同的遍历访问函数
   - 绘制3个流程图：链表中查找给定元素，链表中插入一个节点，链表中删除一个节点。不限制绘制流程图的工具和方法，推荐使用plantuml： https://plantuml.com/zh/
2. 实验内容二：实现顺序栈
   - 完成顺序栈抽象数据类型
   - ADT 的链式存储的设计与实现
   - 使用宏定义实现可以同时定义字符栈、整数栈和结构体栈，并可以使用同一个宏分别对字符栈、整数栈和结构体栈进行操作
3. 可选方案:针对动手编写代码能力尚有欠缺的同学
   - 先阅读代码 linklistc.png,hstack.png（上机实验目录下查找该文件），学习、理解和掌握所列代码
   - 再依据自己的理解，重新输入一遍代码，编译、调试和运行
   - 添加注释（遵循前面的要求）

3.4 检查/提交要求

- 代码能正确编译、运行和输出结果
- 代码注释要求：三行代码至少有一行注释
- 变量、函数等命名有意义
- 提交时间：9月25日24:00时

第四章 高精度数运算

内容提要

- 实验目的
- 实验说明
- 实验内容
- 检查/提交要求

4.1 实验目的

复习线性表、串等 C 语言知识

- 熟练线性表的概念和使用
- 预编译指令的使用
- 熟悉写代码的规范
- 掌握代码调试技术

4.2 实验内容

- 高精度数运算

4.3 实验说明

- 完成线性表抽象数据类型 ADT 的链式存储设计与实现
- 实现实数x(`$1024 < x < 1024$)的加减乘运算，要求运算精确到2^{-n},n 是一个输入参数或预定义参数
- 用十进制或二进制串（线性表）表示实数x
- 构造 ADT，具有读入十进制实数的功能，实现到 N 进制的转换，N 的值在输入时指定且 N 小于 20，实现加减乘三个基本操作，并输出对应的 N 进制结果和十进制结果
- 实现一个复杂操作：单变量多项式求值。例如求函数 $f(x) = \frac{3}{7}x^3\text{-}\frac{1}{3}x^2 + 4$ 的值（其中为$x = 1.4$，精度为：$n = 200$，$\frac{3}{7}$ 和$\frac{1}{3}$等x 前的系数可以不使用高精度）。
- 提示：在运行时解析输入的多项式字符串，分析出每项的系数和幂数，调用基本操作完成计算

4.4 检查/提交要求

- 代码能正确编译、运行和输出结果（能正确完成测试样例即可）
  - 实现二进制和十进制之间的转换: 输入一个高精度十进制小数，转换为高精度二进制小数；输入高精度二进制小数，转换为高精度十进制小数
  - 实现从十进制到 N 进制的转换（其中 N 在输入时指定且 N 小于 20），并实现加减乘操作
    - 输入高精度十进制小数，转换为高精度 N 进制
    - 可以将输入的数进行加减乘计算，并输出对应的十进制和 N 进制结果

- 单变量多项式求值
  - 完成函数$f(x)=\frac{3}{7}x^3\text{-}\frac{1}{3}x^2+4$的值的计算，其中为$x=1.4$，精度为：$n=200$
  - 可以实现一个和示例类似的计算
- 代码注释要求：三行代码至少有一行注释
- 变量、函数等命名有意义
- 提交时间：10月16日24：00

第五章 线性结构的高级操作

内容提要

- 实验目的
- 实验说明
- 实验内容
- 检查/提交要求

5.1 实验目的

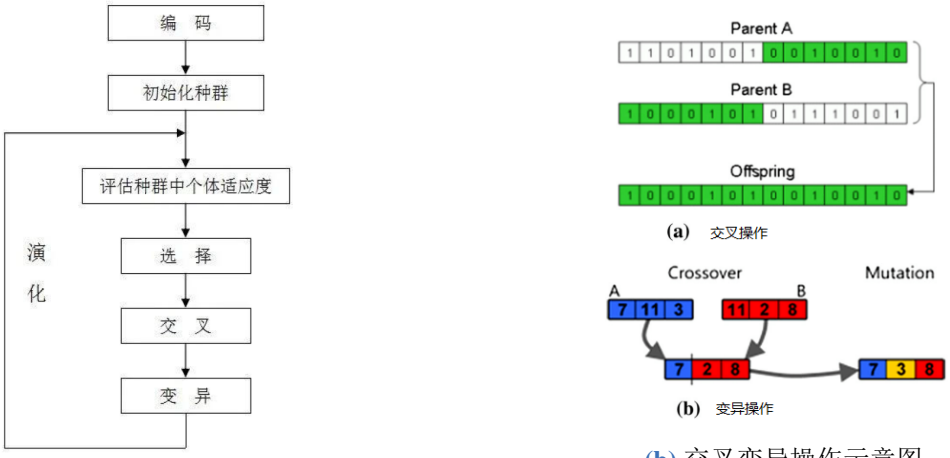
学会广义表的使用，学习编写基础算法

- 了解广义表的构成
- 学会将数据映射到广义表中
- 熟悉写代码的规范
- 掌握代码调试技术

5.2 实验内容

以下题目二选一。

- 进化（遗传）算法求解连续函数的最小值
- 用爬山法求解n-皇后的一个解



(a) 进化算法流程图

(b) 交叉变异操作示意图

图 5.1: 进化算法示意图

5.3 实验说明

1. 实验内容一：进化（遗传）算法求解连续函数的最小值 具体要求: 给定函数$f(x)$，求解其在区间$[0, 16)$ 上的最小值

   进化算法思想：

   - 进化算法/遗传算法是模拟达尔文生物进化论的自然选择和遗传学机理的生物进化过程的计算模型，是一种通过模拟自然进化过程搜索最优解的方法。直观上：优秀的父母会诞生出优秀的孩子
   - 进化算法流程图如图5.1 (a)所示
     - 编码：二进制编码
     - 初始化：初始个体数为N
     - 评估：这里的评估函数应为函数的值
     - 选择：从个体中选择n个作为父代常用方法：最佳保留法，轮盘赌选择
     - 遗传：染色体交叉
     - 变异：基因突变
     - 遗传变异后得到的子代作为新一轮个体重复评估、选择、遗传、变异
   - 进化算法两个主要的操作：交叉和变异，如图5.1 (b)所示

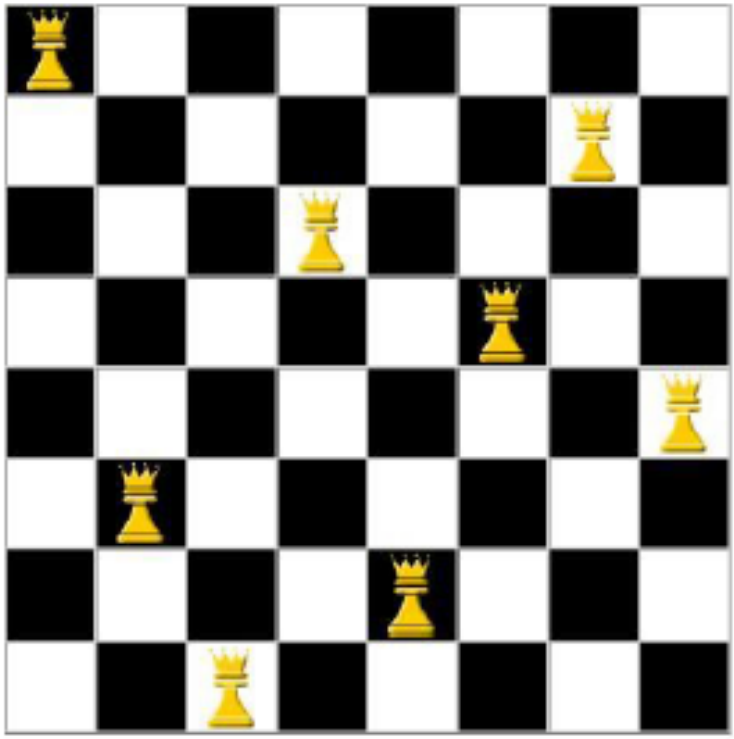


图 5.2: 交叉变异操作示意图

2. 实验内容二：用爬山法求解n-皇后的一个解
   - n-皇后问题的随机搜索算法，找到解即可
   - 问题描述：将n 个皇后放置在$n \times n$ 的棋盘上，并且使皇后彼此之间不能相互攻击，如图所示

- 我们定义一个长度为n 的一维数组solution，solution[i]=row, 表示第i 列的皇后在row 行，并且solution 是0 到$n-1$ 的一个排列，这样巧妙的避免了在水平与竖直方向上的皇后冲突
- 操作：将其中两个皇后的行数进行一次互换
- 邻居：原棋盘经过一次“操作”后的棋盘
- 目标函数：可相互攻击到的皇后对数。我们希望目标函数越小，目标函数为0 时即为我们找到的解
- 本题中我们使用爬山法求解：每次我们找出初始棋盘的所有邻居中冲突最小的作为新的初始棋盘，若所有邻居均不优于初始棋盘，则随机生成另一个初始棋盘。

5.4 检查/提交要求

- 代码能正确编译、运行和输出结果
- 理清算法思路，需要说明算法是如何实现的
- 代码注释要求：三行代码至少有一行注释
- 变量、函数等命名有意义
- 提交时间：10月30日24:00时

第六章 串的实现

内容提要

- 实验目的
- 实验说明
- 实验内容
- 检查/提交要求

6.1 实验目的

熟练掌握并实现串的 ADT

- 掌握串的定义与基本操作，并利用不同存储方式实现串的 ADT
- 预编译指令的使用
- 熟悉写代码的规范
- 掌握代码调试技术

6.2 实验内容

- 串的ADT的实现

6.3 实验说明

- 采用两种不同的存储方式 (定长顺序结构存储、堆分配存储和块链存储三选二) 实现串的 ADT
- 实现串的基本操作，包括但不限于：
  - 初始化: 初始化串
  - 销毁: 销毁串，释放空间
  - 清空: 清为空串
  - 求长度: 返回串中的元素个数，称为串的长度
  - 模式匹配: 定位子串的位置，要求使用 KMP 算法实现
  - 求子串: 返回某个起始位置的某长度的子串
  - 替换:Replace(S, T, V), S 是主串, 用 V 替换主串 S 中出现的所有与 T 相等的不重叠的子串
  - 拼接: 拼接两个串
  - 遍历: 依次输出串中所有字符

6.4 检查/提交要求

- 代码能正确编译、运行和输出结果，即两种存储方式、基本操作准确无误
- 代码注释要求：三行代码至少有一行注释
- 变量、函数等命名有意义
- 提交时间：11月6日24:00时

第七章 二叉树的实现及应用

内容提要

- 实验目的
- 实验说明
- 实验内容
- 检查/提交要求

7.1 实验目的

掌握二叉树的实现与应用

- 掌握二叉树这一数据结构的代码实现
- 能够应用二叉树结构来实现复杂操作
- 掌握递归的算法思想
- 掌握代码调试技术

7.2 实验内容

- 二叉树的实现及应用

7.3 实验说明

- 读懂老师给出的代码框架中的代码（二叉链表实现的二叉树），几种构造二叉树的方法的代码细节都要读懂，代码下载：
  https://rec.ustc.edu.cn/share/dd8337e0-9241-11ed-b64a-eb675d79527e
- 实现操作：删除节点value=x 的节点及其子树
- 实现操作：给定id1 和id2，输出其最近共同祖先id，节点的id 具有唯一性
- 实现操作：给定id 值，输出从根节点到id 值节点的路径，用“左右右...”的左右孩子指针标记从根到节点的路径
- 编写递归算法，输出二叉树节点中最大的value 和最小的value 之差

7.4 检查/提交要求

- 代码能正确编译、运行和输出结果
- 代码注释要求：三行代码至少有一行注释
- 能回答出助教提问的关于代码框架中代码的问题
- 变量、函数等命名有意义
- 提交时间：11月13日24:00时

第八章 哈夫曼树的实现及应用

内容提要

- 实验目的
- 实验说明
- 实验内容
- 检查/提交要求

8.1 实验目的

掌握哈夫曼树的实现与应用

- 掌握哈夫曼树这一数据结构的代码实现
- 能够应用二叉树结构来实现压缩和解压缩操作
- 掌握哈夫曼编码的算法思想
- 掌握代码调试技术

8.2 实验内容

- 哈夫曼树的实现及应用

8.3 实验说明

- 读懂老师给出的代码框架中的代码（结构体数组实现的哈夫曼树），代码细节也要读懂，然后自己使用三叉链表(” 三叉” 指*parent, *lchild 和*rchild) 实现哈夫曼树，并实现以下操作
- 实现操作：压缩并解压一个文件(如.txt 文件)
- 代码下载：
  https://rec.ustc.edu.cn/share/f3b11a00-916e-11ed-8453-356902ba045b,哈夫曼树的显示web显示部分需要依据实验五的参考代码修改。

8.4 检查/提交要求

- 代码能正确编译、运行和输出结果
- 代码注释要求：三行代码至少有一行注释
- 能回答出助教提问的关于代码框架中代码的问题
- 变量、函数等命名有意义
- 提交时间：11月20日24:00时

第九章 图的 ADT 的实现

内容提要

- ❑ 实验目的
- ❑ 实验内容
- ❑ 实验说明
- ❑ 检查/提交要求

9.1 实验目的

掌握图相关的知识

- 掌握图的概念和使用
- 预编译指令的使用
- 熟悉写代码的规范
- 掌握代码调试技术

9.2 实验内容

- 图的 ADT 的实现

9.3 实验说明

- 完成图的 ADT 的设计与实现
- 读懂给出代码框架
- 实现图的基本操作包括，但是不限于：图的创建、删除顶点、增加顶点、增加边、删除边、查找顶点、修改边或顶点
- 图的存储结构自行设计，同时设计实现一个复杂操作，验证上述基本操作，并说明采该存储结构实现这个复杂操作是否合适（用代码注释说明，且向助教说明结构和操作是否适配
- 代码框架下载：
  `https://rec.ustc.edu.cn/share/3f8c3a50-9244-11ed-94f4-67e66ab69b5a`

9.4 检查/提交要求

- 代码能正确编译、运行和实现并验证图的基本操作和自己设计的复杂操作，可以通过使用的存储结构输出整个图以验证操作的实现是否正确
- 代码注释要求：三行代码至少有一行注释
- 能回答出助教提问的关于代码框架中代码的问题
- 变量、函数等命名有意义
- 提交时间：12月4日24:00时

第三部分

参考解答

第十章 实验参考代码

10.1 实验一参考代码

10.1.1 熟悉链表操作

代码下载链接：https://rec.ustc.edu.cn/share/3e386a60-9169-11ed-ba57-4dfe7b98e8ee

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

//需要定义什么样的链表，就将数据域的数据类型定义为ElemType
#define ElemType int

//用于调试，把下面这句宏定义注释掉，那么宏定义#ifdef ...# endif 包括的语句在编译时全部忽略
#define _DEBUG_ 1


//静态部分的定义
typedef struct Lnode {          //链表结点
    ElemType data;              //结点数据域
    struct Lnode * next;        //结点链域

} LinkNode, *LinkList;

//
LinkNode *first ;          //为型的变量，为单链表的头指针 firstLinkList

//以下先给出基本操作的实现

/*** 初始化链表     initList ()
 *** 在内存中将头指针赋予正确的值，考虑下面三种初始化的方法
 ***/
LinkNode *initList (){        //初始化链表，在内存中生成头结点
    LinkNode *p = (LinkNode *)malloc(sizeof(LinkNode));
    if (!p) {
        printf ("初始化分配头结点失败！\n");
        exit (0);
    }
    p->next = NULL;       //p->指针赋值为空，通常用于判断链表的结束；域不用，故不用赋值nextdata
    return p;
}

#ifdef _DEBUG_

LinkNode head;          //定一个头结点，结构体变量，全局变量
```

```
/*** 考虑下面这两种初始化 ***/
void  initList1 (LinkNode *p){
    p = (LinkNode *)malloc(sizeof(LinkNode));
    if (!p) {
        printf ("初始化分配头结点失败！\n");
        exit (0);
    }
    p->next = NULL;       //p->指针赋值为空，通常用于判断链表的结束；域不用，故不用赋值nextdata
}

void  initList2 (){
    first  = &head;
    first ->next = NULL;
}
#endif

/*** 求链表的长度    ListLength()
 *** 需要将链表遍历一次时间复杂度为 ,O(n链表长度=)
 ***/
int ListLength(LinkList p){
    int count = 0;
    while (p->next!=NULL){ //为什么直接用指针循环，不用辅助变量也可以？为什么不会丢失链表？p
        count++;
        p = p->next;
    }
    // for( int  count=0;p->next!=NULL;p=p->next,count++);
    return count;
}

/*** 判断链表是否为空    ListEmpty()
 *** 如果为空，返回，否则返回非零，时间复杂度为 0O(1)
 ***/
int ListEmpty(LinkList p){
    if (p->next!=NULL)
        return 1;
    return 0;
}

/*** 清空链表    ClearEmpty()
 *** 循环删除链表第一个节点，时间复杂度为 O(n)
 ***/
void ClearList(LinkList p){
    LinkNode *q;          //额外辅助变量
    while(p->next!=NULL){
        q = p->next;   //辅助变量指向第一个存放数据的节点数据元素/
        p-> next = q->next;  //头指针的指向第二个元素next
        free(q);          //释放第一个节点占据的存储空间
    }   //循环停止时，头指针的指向，表示没有真正的数据元素存在nextNULL
}
```

```
/*** 销毁链表   DestroyEmpty()
 *** 先清空链表，然后去释放头结点分配的空间，时间复杂度为 O(n)
 ***/
void  DestroyList (LinkList  p){
    ClearList (p);
    free (p);   //用 initList ()初始化，用 initList2 ()初始化时，不用任何操作。  initList1 ()呢？
}

/*** 取链表第个节点的数据 i   GetElem()
 *** 遍历链表，注意边界条件的检查，时间复杂度为 O(n)
 *** 返回表示取值失败，非零表示成功 0
 ***/
int  GetElem(LinkList p,  int  i, ElemType *e){
    int  k = 0;
    while(p=p->next){ //语言中，赋值表达式的值就是等号左边的值；该句完成赋值的同时，循环判断是否为空Cp
        k++;
        if  (k==i){
            *e = p->data;
            return  k;
        }
    }
    return  0;
} //为什么不判断输入是否正确？i

/*** 查找链表数据元素为的节点 e   LocateElem()
 *** 遍历链表，时间复杂度为 O(n)
 *** 返回指向数据元素 data=的第一个节点e 适用于数据域是整数或者字符，否则不能用）('='
 ***/
LinkNode *LocateElem(LinkList p,  ElemType e){
    while(p=p->next){
        if  (p->data == e)
            return  p;
    }
    return  NULL;
}

/*** 查找链表某个节点的前驱节点   PriorElem()
 *** 遍历链表，注意边界条件的检查，时间复杂度为 O(n)
 *** 返回时，表示没找到  NULLcur_e
 ***/
// void  PriorElem(LinkList  p,  LinkNode *cur_e,  LinkNode *pre_e){ 错误为什么？  //,  无法得到想要的返回指针
LinkNode *PriorElem(LinkList p,  LinkNode *cur_e){
    for (;p->next;p=p->next)
        if  (p->next == cur_e)
            return  p;
    return  NULL;
}
```

```
/*** 查找链表某个节点的后继节点    NextElem()
 *** 直接返回的指针，时间复杂度为 cur_enextO(1)
 *** 返回时，表示没后继   NULLcur_e; 可以写一个循环来检查是否在链表中cur_e
 ***/
LinkNode *NextElem(LinkList p, LinkNode *cur_e){
    return cur_e->next;
}

/*** 向链表中插入一个节点     ListInsert ()
 *** 遍历链表，查找插入位置，时间复杂度为 O(n)
 *** 插入成功返回插入节点指针，返回时，表示插入失败   NULL
 ***/
LinkNode *ListInsert (LinkList p, int i, ElemType e){
    if (i<1) return NULL;    //位置，太小i
    for (;p;p=p->next)
        if (--i<1){ //找到插入位置
            LinkNode *q = (LinkNode *)malloc(sizeof (LinkNode));
            if (!q) {
                printf ("插入节点时，分配空间失败！\n");
                exit (0) ;
            }
            q->next = p->next; //新节点的指向循环到当前位置的nextpnext
            p->next = q;          // 当前位置的指针指向新节点，这两行代码不能交换次序next
            q->data = e;
            return q;
        }
    return NULL;    //位置，太大，超过链表长度i
}

/*** 从链表中删除一个节点     ListDelete ()
 *** 遍历链表，查找删除位置，时间复杂度为 O(n)
 *** 删除成功返回非零，返回时，表示删除失败 0
 ***/
int ListDelete (LinkList p, int i, ElemType *e){
    if (i<1) return 0;    //位置，太小i
    LinkNode *q = p;          //将视为循环变量，指向的前驱pqp
    for(p=p->next;p;p=p->next){
        if (--i<1){ //找到删除位置
            q->next = p->next; //的指向当的，前驱越过当前节点qnextpnext
            *e = p->data;
            free (p);          //释放空间
            return 1;
        }
        q = p;
    }
    return 0;    //位置，太大，超过链表长度i
}

/*** 定义节点访问函数  visit ()
```

```
 *** 采用函数指针来实现，供 ListTranverse ()函数调用
 ***/
void  PrintLinkNode(LinkNode *p){
      printf ("%d,",p->data);  //适用于是整数data
}

void  Add2(LinkNode *p){
     p->data += 2;     // visit ()函数的实现，每个元素+2
      printf ("+2,");
}

//用于调试代码
void  DebugLinkNode(LinkNode *p){
      printf ("结点-(*addr)=value :  ");
      printf ("(*%lx)=%d\n",p,p->data);  //仅适用于为整数data
}

/*** 遍历链表     ListTraverse ()
 *** 遍历链表，查找删除位置，时间复杂度为 O(n)
 *** 删除成功返回非零，返回时，表示删除失败 0
 ***/
void  ListTraverse (LinkList  p,void  (*ptrFunc)(LinkNode *ptr)){
      printf ("链表表长(=%d)：", ListLength(p));
     while(p=p->next)
          (*ptrFunc)(p);
      printf ("\n");
     //ListLength(p); 错误，为什么？  //
}

int  main(){  //用于测试基本操作

     // void  (*ptrFunc)(LinkNode *p) = PrintLinkNode; 声明函数指针   //
     void  (*ptrFunc)(LinkNode *p) = DebugLinkNode;  //声明函数指针

     //初始化
      first  =  initList ();
     //  initList1 ( first ); 这种初始化怎么样？    //
     //  initList2 (); 如何？          //


     //在不同位置插入数据
     ListTraverse ( first ,ptrFunc);
      ListInsert ( first ,1,2) ;
      ListInsert ( first ,1,3) ;
      ListInsert ( first ,1,4) ;
      ListInsert ( first ,1,5) ;
     ListTraverse ( first ,ptrFunc);
      ListInsert ( first ,1,6) ;
      ListInsert ( first ,1,7) ;
```

```
ListInsert ( first ,1,8) ;
ListInsert ( first ,1,9) ;
ListTraverse ( first ,ptrFunc);
ListInsert ( first ,3,666) ;
ListInsert ( first ,5,777) ;
ListInsert ( first ,7,888) ;
ListInsert ( first ,9,999) ;
ListTraverse ( first ,ptrFunc);

//查找第个数据或者值为的数据ix
ElemType ei;
printf (”取数据之前 %d ——”, ei);
GetElem( first ,10,&ei);
printf (”读取的数据为 %d \n”,ei);

LinkNode *q = LocateElem( first , 888);
if (!q)
    printf (”没找到值所对应的结点\n”);
else{
    q = PriorElem( first ,q);
    printf (”找到结点的前驱为 %d —— ”, q->data);
    printf (”找到结点为 %d —— ”, q->next->data);
    if (q->next->next)
        printf (”找到结点的后继为 %d ”, NextElem( first ,NextElem( first ,q))->data);
    printf (”\n”);
}

//删除数据
printf (”删除前的值 %d —— ”, ei);
if ( ListDelete ( first ,15,&ei)>0)
    printf (”删除的值为 %d\n”, ei);
else
    printf (”删除失败 %d \n”, ei);

ListTraverse ( first ,ptrFunc);
printf (”删除前的值 %d —— ”, ei);
if ( ListDelete ( first ,10,&ei)>0)
    printf (”删除的值为 %d\n”, ei);
else
    printf (”删除失败 %d \n”, ei);

printf (”删除前的值 %d —— ”, ei);
if ( ListDelete ( first ,6,&ei)>0)
    printf (”删除的值为 %d\n”, ei);
else
    printf (”删除失败 %d \n”, ei);
ListTraverse ( first ,ptrFunc);

ptrFunc = Add2;                    // visit ()函数定义为Add2()
```

```
    printf ("每个数据元素准备+2\n");
    ListTraverse ( first ,ptrFunc);          //将链表每个数据都加2
    printf ("完成后，新的链表：+2");
    ListTraverse ( first ,PrintLinkNode);   //直接用函数名当参数，进行调用
    ListTraverse ( first ,PrintLinkNode);
    ListTraverse ( first ,Add2);

    //销毁链表，销毁过程中调用了清空链表
    DestroyList ( first );

    return 0;
}
```

10.1.2 顺序栈的实现

代码下载链接：https://rec.ustc.edu.cn/share/5bdda3b0-9169-11ed-a4ad-85472e2d3ddf

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

#define STACK_INIT_SIZE 100
#define STACKINCREMENT 10

//顺序整数栈的静态结构
unsigned    int_stacksize ;   //预分配的栈空间大小
int * int_stacktop_ptr ;      //栈顶指针
int * int_stackbase_ptr ;     //栈底指针

/***
 *** 下面用宏定义来实现 8 个栈的基本操作
 ***  ## 表示把前后两个参数连接起来
 ***  gcc -E hstack.cpp 只做预编译，输出宏定义展开后的结果，用于粗略检查宏定义是否存在问题
 ***/
#define  initStack ( stack ) stack ## _stackbase_ptr = ( stack *)malloc( sizeof ( stack )*STACK_INIT_SIZE);\
                             if ( stack ## _stackbase_ptr ){\
                                 stack ## _stacktop_ptr = stack ## _stackbase_ptr ;\
                                 stack ## _stacksize = STACK_INIT_SIZE;\
                             } else exit (0)

#define stackEmpty(stack) stack ## _stackbase_ptr == stack ## _stacktop_ptr ? 1:0

#define getTop(stack,e) stack ## _stackbase_ptr == stack ## _stacktop_ptr ? 0:(e = *(stack ## _stacktop_ptr -1),1)

#define  clearStack ( stack ) stack ## _stacktop_ptr = stack ## _stackbase_ptr

#define  destroyStack ( stack ) free ( stack ## _stackbase_ptr )
```

```
#define stackLength(stack) stack ## _stacktop_ptr - stack ## _stackbase_ptr

#define pop(stack,e) (stack ## _stackbase_ptr == stack ## _stacktop_ptr )? 0:(e =*(--stack ## _stacktop_ptr ),1)

#define push(stack,e) if (stack ## _stacktop_ptr - stack ## _stackbase_ptr >= stack ## _stacksize ){\
                        stack ## _stackbase_ptr = (stack *) realloc (stack ## _stackbase_ptr , \
                          (stack ## _stacksize + STACKINCREMENT)*sizeof(stack));\
                        if (! stack ## _stackbase_ptr ) exit (0);\
                        stack ## _stacktop_ptr = stack ## _stackbase_ptr + stack ## _stacksize ;\
                        stack ## _stacksize += STACKINCREMENT;}\
                      *(stack ## _stacktop_ptr ++) = e


//栈的遍历操作 stackTraverse () 忽略

/*** 定义其它基本类型的栈，比如字符栈或者结构体栈，只需要定义静态结构，基本操作重复利用上面的宏定义
 *** 下面的例子分别定义了字符栈和结构体栈
 ***/
unsigned  char_stacksize ;      //预分配的栈空间大小
char * char_stacktop_ptr ;      //栈顶指针
char * char_stackbase_ptr ;     //栈底指针

//结构体栈的静态部分，未测试
typedef struct node {           //用给定义的栈取名字typedef
    int data [10];
    float x,y;
} tnode;

unsigned tnode_stacksize ;      //预分配的栈空间大小
tnode * tnode_stacktop_ptr ;    //栈顶指针
tnode * tnode_stackbase_ptr ;   //栈底指针

//定义栈的，目的是为了只用基本操作实现复杂功能，防止误操作ADT
int main(){

    initStack (int);
    initStack (char);
    initStack (tnode);

    //测试整数栈
    int x;
    if (pop(int,x))
        printf ("出栈成功 %d\n",x);
    else
        printf ("栈空，不能出栈\n");

    printf ("栈中有 %d 个元素\n",stackLength(int));

    if (stackEmpty(int))
```

```
        printf ("栈空，无法取栈顶\n");
    else
        if (getTop(int,x))
            printf ("栈顶元素是 %d \n", x);

    push(int,3);
    printf ("栈中有 %d 个元素\n",stackLength(int));

    push(int,4);
    push(int,5);

    printf ("栈中有 %d 个元素\n",stackLength(int));

    if (pop(int,x))
        printf ("出栈成功 %d\n",x);
    else
        printf ("栈空，不能出栈\n");

    printf ("栈中有 %d 个元素\n",stackLength(int));

    if (stackEmpty(int))
        printf ("栈空，无法取栈顶\n");
    else
        if (getTop(int,x))
            printf ("栈顶元素是 %d \n", x);

    clearStack (int);
}
```

10.2 实验二参考代码

10.2.1 高精度运算

代码下载链接：https://rec.ustc.edu.cn/share/51242370-916b-11ed-9aea-07abf44cdf11，提供代码的同学未具名(需要认领的同学联系我)。

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

// 可以使用宏定义来规定精度
#define INTE_MAX 1000
#define DECI_MAX 1000
// 输出精度
#define PRINT_MAX 50
```

```
typedef struct num{ // 高精度数的存储结构
    int negative; // 正负，正号为，负号为01
    int data[INTE_MAX + DECI_MAX]; // 存储数据
    int *number; // 令number = data + DECI_MAX 则num = ... + number[-1] * base^-1 + number[0] * base^0 + number[1]
        * base^1 +...
    int base;
}num;

num* initNUM();
num* ReadFromString(char * str, int base); // 从一个数字字符串读取生成高精度数
void printnum(num* N); // 输出高精度数
num* baseConversion(num* N, int base); // return 转化为A 进制下的数base
num * add(num * A, num* B); // return A + B
num * minus(num * A, num* B); // return A - B
num * multi(num* A,num* B); // return A * B
num * calculate (char * fx,char * x, int inbase, int outbase); // 仅含有数字字母加减乘的多项式计算

int main(){
    num * tmp,* t1 , * t2;
    printf ("测试读取字符串：\n");
    tmp = ReadFromString("-204.2", 10);
    printnum(tmp);

    printf ("测试进制转化：\n");
    t1 = baseConversion(tmp, 2);
    printnum(t1);
    t2 = baseConversion(tmp, 10);
    tmp = ReadFromString("-11001100.00110011001100110011001100110011001100110011001100110011001100", 2);
    t2 = baseConversion(tmp, 10);
    printnum(t2);

    printf ("测试加减乘：\n");
    t1 = ReadFromString("1.14514",10);
    t2 = ReadFromString("-1.919810",10);
    printnum(add(t1,t2));
    printnum(minus(t1,t2));
    printnum(multi(t1,t2));
    printf ("上述转化为进制：7\n");
    printnum(baseConversion(add(t1,t2),7));
    printnum(baseConversion(minus(t1,t2),7));
    printnum(baseConversion(multi(t1,t2),7));

    printf ("多项式测试：\n");
    printf ("测试进制转化：\n");
    tmp = calculate ("-204.2","0",10,2);printnum(tmp); // 将进制的转进制10204.22
    tmp = calculate ("x","-11001100.00110011001100110011001100110011001100110011001100110011001100",2,10);printnum(tmp); // 将进
        制的转进制可以看到，常数多项式可以考虑这两种输入方式2-11001100.0011001110.
    printf ("测试加减乘：\n");
    tmp = calculate ("1.14514+-1.919810","8008208820",10,10);printnum(tmp); // 符号优先级同一般机器语
```

```
        言 http ://home.ustc.edu.cn/~ziheng/pic/test.png
    tmp = calculate ("1.14514--1.919810","8008208820",10,10);printnum(tmp);
    tmp = calculate ("1.14514*-1.919810","8008208820",10,10);printnum(tmp);
    printf ("上述转化为进制：7\n");
    tmp = calculate ("1.14514+-1.919810","0",10,7);printnum(tmp);
    tmp = calculate ("1.14514--1.919810","0",10,7);printnum(tmp);
    tmp = calculate ("1.14514*-1.919810","0",10,7);printnum(tmp);
    printf ("含有变量的多项式测试：\n");
    tmp = calculate ("12.31379131*x*x+x+8.2137619836821388","1.612368921371923122414",10,10);printnum(tmp);
    tmp = calculate ("-12.31379131*x*x+x+-8.2137619836821388","1.612368921371923122414",10,10);printnum(tmp); // 测
        试负号
    tmp = calculate ("x*-12.31379131*x-x-8.2137619836821388","1.612368921371923122414",10,10);printnum(tmp); // 测
        试负号
    return 0;
}

num* initNUM(){
    num* tmp = (num*)malloc(sizeof(num));
    tmp->base = 0;
    tmp->negative = 0;
    tmp->number = tmp->data + DECI_MAX;
    for(int i = -DECI_MAX; i < INTE_MAX; ++i)
        tmp->number[i] = 0;
    return tmp;
}

num* baseConversion(num* N, int base){  // return 转化为A 进制下的数base
    num* tmp = initNUM();
    if(N->base == base){
        *tmp = *N;
        tmp->number = tmp->data + DECI_MAX;
        return tmp;
    }
    tmp->base = base;                   // 进制
    tmp->negative = N->negative;  // 正负与原来一致
    // 整数部分
    for(int i = 0; i < INTE_MAX; ++i){
        for(int j = 0; j < INTE_MAX; ++j)
            tmp->number[j] *= N->base;
        tmp->number[0] += N->number[INTE_MAX - 1 - i];
        for(int j = 0; j < INTE_MAX - 1; ++j)
            if(tmp->number[j] >= base){
                tmp->number[j + 1] += tmp->number[j] / base;
                tmp->number[j] = tmp->number[j] % base;
            }
    }
    // 小数部分
    int deci[DECI_MAX];
    for(int i = 1; i < DECI_MAX; ++i) deci[i] = N->number[-i];  // 保存的小数部分N
    for(int i = 1; i < DECI_MAX; ++i) {
```

```
        deci[0] = 0;
        for (int j = 1; j < DECI_MAX; ++j)
            deci[j] *= base;
        for (int j = DECI_MAX - 1;j > 0; --j) {
            deci[j - 1] += deci[j] / N->base;
            deci[j] = deci[j] % N->base;
        }
        tmp->number[--i] = deci[0];
    }
    return tmp;
}

int abscompare(num * A, num* B){    // 比较 abs(A) 与 abs(B) 的大小
    A = baseConversion(A, B->base);
    for(int i = INTE_MAX - 1; i >= -DECI_MAX; --i) // 从高位比较
        if (A->number[i] != B->number[i]){
            int tmp = A->number[i] - B->number[i];
            free (A);
            return tmp > 0 ? 1 : -1;
        }
    free (A);
    return 0;
}

num * minus(num * A, num* B){   // return A - B
    if (A->negative != B->negative){    // 符号不同AB
        B->negative = 1 - B->negative; // 将符号改为一致AB
        num* tmp = add(A,B);
        B->negative = 1 - B->negative;
        return tmp;
    }
    if (abscompare(A,B) < 0){    // |A| > |B，| 则 A - B 变号，return -(B - A)
        num* tmp = minus(B,A);
        tmp->negative = 1 - tmp->negative;
        return tmp;
    }
    // 同号，且AB|A| > |B| , A - B
    num* tmp = baseConversion(A,B->base);  // tmp = A ，且与进制一样B
    // 借位
    for(int i = -DECI_MAX; i < INTE_MAX; ++i){ // 从低位向高位遍历
        tmp->number[i] -= B->number[i];
        if (tmp->number[i] < 0){
            tmp->number[i + 1] --;
            tmp->number[i] += tmp->base;
        }
    }
    return tmp;
}
```

```
num * add(num * A, num* B){ // return A + B
    if (A->negative != B->negative){    // 符号不同AB
        B->negative = 1 - B->negative; // 将符号改为一致AB
        num* tmp = minus(A,B);
        B->negative = 1 - B->negative;
        return tmp;
    }
    // 同号AB
    num* tmp = baseConversion(A,B->base);  // tmp = A，且与进制一样B
    // 进位
    for(int i = -DECI_MAX; i < INTE_MAX; ++i){ // 从低位向高位遍历
        tmp->number[i] += B->number[i];
        if (tmp->number[i] >= tmp->base){
            tmp->number[i + 1] ++;
            tmp->number[i] -= tmp->base;
        }
    }
    return tmp;
}

num * multi(num* A,num* B){    // A * B
    num* tmp = initNUM();
    A = baseConversion(A,B->base); // 令转化到与一个进制AB
    tmp->negative = A->negative == B->negative ? 0 : 1; // 同号得正，异号为负
    tmp->base = B->base;
    // 我们知道 (a0 + a1*p^1 + a2*p^2 ...) * (b0 + b1*p^1 + .... ) = sum_i (sum_k+j=i ak + bj )*p^i
    for(int i = -DECI_MAX; i < INTE_MAX; ++i){
        for(int j = -DECI_MAX; j < INTE_MAX; ++j)
            if (i - j >= -DECI_MAX && i - j < INTE_MAX) // i - j 也需要满足在精度范围内
                tmp->number[i] += A->number[j] * B->number[i - j]; // sum_i (sum_k+j=i ak + bj ) * p^i 其中是 ki-j
        // 进位
        if (tmp->number[i] >= tmp->base){
            tmp->number[i + 1] = tmp->number[i] / tmp->base;
            tmp->number[i] = tmp->number[i] % tmp->base;
        }
    }
    return tmp;
}

void printnum(num* N){
    if (N->negative) printf("-");
    int i;
    for(i = INTE_MAX - 1;i >= 1 && !N->number[i]; --i); // 从高位到低位找到非位0
    for( ; i >= -PRINT_MAX; --i){
        if (N->number[i] < 10)    printf ("%d",N->number[i]);
        else  if (N->number[i] < 36) printf("%c",'a' + N->number[i]-10);
        if (i == 0)    printf (".");
    }
    printf ("\n");
```

```
}

num* ReadFromString(char * str, int base){  // 从一个数字字符串读取生成高精度数
    // 如 ReadFromString("123546.2321", 10) 返回一个进制下的高精度数10
    num* tmp = initNUM();
    tmp->base = base;
    if( str [0] == '-'){  // 处理负号
        tmp->negative = 1;
        str ++;
    }
    int dot;
    for(dot = 0; str [dot] && str[dot]!= '.'; ++dot);      // 找到小数点所在位置
    for(int i = 0; i < dot; ++i)                  // 整数
        tmp->number[i] = (str[dot - i - 1] >= '0' && str[dot - i - 1] <= '9') ?  str [dot - i - 1] - '0' : str [dot - i
            - 1] - 'a' + 10;  // 数字 or 字
            母
    for(int i = -1; str [dot - i ]; --i ) // 小数
        tmp->number[i] = (str[dot - i] >= '0' && str[dot - i] <= '9') ?  str [dot - i] - '0' : str [dot - i] - 'a' + 10;
              // 数字 or 字
            母
    return tmp;
}

// 提供了一种大一统的输入，可处理所有样例，基本上实现单变量多项式就可以实现所有样例（毕竟常数也是多项式
// 当然也可以自行设计其他类型的输入
/***
 * 是多项式字符串fx
 * 是多项式自变量的值x
 * 是输入多项式与自变量的进制inbase
 * 是最终结果的进制outbase
***/
num *calculate(char * fx,char * x, int inbase, int outbase){     // 仅存在fx 数字字母 +-*
    int i , j;
    char fx1[999] = {0};
    for(i = 0; fx[i] && fx[i] != '+'; ++i); // 检测第一个加法
    if(fx[i]){       // 形如 A + B
        strncpy(fx1 ,fx , i);
        num * t1 = calculate (fx1, x, inbase, outbase);
        num * t2 = calculate (fx + i + 1, x, inbase, outbase);
        num * tmp = add(t1 ,t2); free (t1 ); free (t2);
        t1 = baseConversion(tmp,outbase); free (tmp);
        return t1;
    }

    for(i = j = 0; fx[j ]; ++j)             // 检测最后一个减法
        if(j > 0 && fx[j] == '-' && fx[j - 1] != '*' && fx[j - 1] != '-')    // 需要排除负数的 -
            i = j;
    if(i > 0){  // 形如 A - B 且中全是乘法 B
        strncpy(fx1 ,fx , i);
        num * t1 = calculate (fx1, x, inbase, outbase);
```

```
        num * t2 = calculate(fx + i + 1, x, inbase, outbase);
        num * tmp = minus(t1,t2); free(t1); free(t2);
        t1 = baseConversion(tmp,outbase); free(tmp);
        return t1;
    }

    for(i = 0; fx[i] && fx[i] != '*'; ++i); // 检测第一个乘法
    if(fx[i]){
        strncpy(fx1,fx, i);
        num * t1 = calculate(fx1, x, inbase, outbase);
        num * t2 = calculate(fx + i + 1, x, inbase, outbase);
        num * tmp = multi(t1,t2); free(t1); free(t2);
        t1 = baseConversion(tmp,outbase); free(tmp);
        return t1;
    }
    // 单独一个数
    int negative = 0;
    if(*fx == '-'){
        fx++;
        negative = 1;
    }
    num * tmp = ReadFromString(*fx == 'x' ? x : fx,inbase);
    tmp->negative = negative;
    num * t1 = baseConversion(tmp,outbase); free(tmp);
    return t1;
}
```

致谢

1. 本书部分内容由授课班级的助教梁永濠、丁家和、陈新宇、唐晨铨和刘阳协助整理、收集和编写。
2. 本书基于ElegantLaTeX 项目组提供的书籍模板完成撰写。ElegantLaTeX 项目组致力于打造一系列美观、优雅、简便的模板方便用户使用。目前由 ElegantNote，ElegantBook，ElegantPaper 组成，分别用于排版笔记，书籍和工作论文。
3. 本书内容用于教学，部分引用图表来自互联网，不一一指出原始出处，如有侵犯了原作者知识产权的情形，请告知并联系jlli@ustc.edu.cn.