# 数据结构实验指导书

# （C语言版）

**2020年3月**

目 录

[1、顺序表的实现 1](#_Toc495496627)

[2、链栈的实现 3](#_Toc495496628)

[3、前序遍历二叉树 5](#_Toc495496629)

[4、图的深度优先遍历算法 7](#_Toc495496630)

[5、散列查找 9](#_Toc495496631)

# 授课平台：腾讯课堂

# 平台链接：

# <https://ke.qq.com/webcourse/4984040/105156057#from=800021724&lite=1&live=1>

# 

# 1、顺序表的实现

**1. 实验目的**

⑴ 掌握线性表的顺序存储结构；

⑵ 验证顺序表及其基本操作的实现；

⑶ 理解算法与程序的关系，能够将顺序表算法转换为对应的程序。

**2. 实验内容**

⑴ 建立含有若干个元素的顺序表；

⑵ 对已建立的顺序表实现插入、删除、查找等基本操作。

**3. 实现提示**

定义顺序表的数据类型——顺序表结构体SeqList，在SeqList基础上实现题目要求的插入、删除、查找等基本操作，为便于查看操作结果，设计一个输出函数依次输出顺序表的元素。简单起见，本实验假定线性表的数据元素为int型，要求学生：

（1）将实验程序调试通过后，用模板类改写；

（2）加入求线性表的长度等基本操作；

（3）重新给定测试数据，验证抛出异常机制。

**4. 实验程序**

**异常处理问题**

**模板类没用**

在编程环境下新建一个工程“顺序表验证实验”，并新建相应文件，文件包括顺序表结构体SeqList的定义，范例程序如下：

#define MaxSize 100 /\*假设顺序表最多存放100个元素\*/

typedef int DataType; /\*定义线性表的数据类型，假设为int型\*/

typedef struct

{

DataType data[MaxSize]; /\*存放数据元素的数组\*/

int length; /\*线性表的长度\*/

} SeqList;

文件包括建立顺序表、遍历顺序表、按值查找、插入操作、删除操作成员函数的定义，范例程序如下：

int CreatList(SeqList \*L, DataType a[ ], int n)

{

if (n > MaxSize) {

printf("顺序表的空间不够，无法建立顺序表\n");

return 0;}

for (int i = 0; i < n; i++)

L->data[i] = a[i];

L->length = n;

return 1;

}

void PrintList(SeqList \*L)

{

for (int i = 0; i < L->length; i++)

printf("%d ", L->data[i]); /\*输出线性表的元素值，假设为int型\*/

}

int Locate(SeqList \*L, DataType x)

{

for (int i = 0; i < L->length; i++)

if (L->data[i] == x)

return i+1; /\*返回序号\*/

return 0; /\*退出循环，说明查找失败\*/

}

int Insert(SeqList \*L, int i, DataType x)

{

if (L->length >= MaxSize)

{

printf("上溢错误，插入失败\n");

return 0;

}

if (i < 1 || i > L->length + 1)

{

printf("位置错误，插入失败\n");

return 0;

}

for (int j = L->length; j >= i; j--) /\*j表示元素序号\*/

L->data[j] = L->data[j - 1];

L->data[i - 1] = x;

L->length++;

return 1;

}

int Delete(SeqList \*L, int i, DataType \*ptr)

{

if (L->length == 0)

{

printf("下溢错误，删除失败\n");

return 0;

}

if (i < 1 || i > L->length)

{

printf("位置错误，删除失败\n");

return 0;

}

\*ptr = L->data[i - 1]; /\*取出位置i的元素\*/

for (int j = i; j < L->length; j++) /\* j表示元素所在数组下标\*/

L->data[j - 1] = L->data[j];

L->length--;

return 1;

}

在定义了顺序表的存储结构SeqList并实现了基本操作后，程序中就可以使用SeqList类型来定义变量，可以调用实现基本操作的函数来完成相应的功能。范例程序如下：

#include <stdio.h>

#include <stdlib.h>

/\*将顺序表的存储结构定义和各个函数定义放到这里\*/

int main(){

int r[5] = {1, 2, 3, 4, 5}, i, x;

SeqList L; /\*定义变量L为顺序表类型\*/

CreatList(&L, r, 5); /\*建立具有5个元素的顺序表\*/

printf("当前线性表的数据为：");

PrintList(&L);

Insert(&L, 2, 8); /\*在第2个位置插入值为8的元素\*/

printf("\n执行插入操作后数据为：");

PrintList(&L); /\*输出插入后的线性表1 8 2 3 4 5\*/

printf("\n请输入查找的元素值：");

scanf("%d", &x);

i = Locate(&L, x);

if (0 == i) printf("查找失败\n");

else printf("元素%d的位置为：%d\n", x, i);

printf("\n请输入要删除第几个元素：");

scanf("%d", &i);

if (Delete(&L, i, &x) == 1) { /\*删除第i个元素\*/

printf("\n删除第%d个元素是%d，删除后数据为：", i, x);

PrintList(&L); /\*输出删除后的线性表\*/

}

else printf("\n删除操作失败\n");

return 0;

}

# 2、链栈的实现

**1. 实验目的**

⑴ 掌握栈的链接存储结构；

⑵ 验证链栈及其基本操作的实现；

⑶ 验证栈的操作特性。

**2. 实验内容**

⑴ 建立一个空栈；

⑵ 对已建立的栈进行插入、删除、取栈顶元素等基本操作。

**3. 实现提示**

定义链栈中的结点结构（链栈中结点结构基于单链表相同），定义链栈的数据类型——链栈结构体，包括入栈、出栈、取栈顶元素等基本操作。本节的实验采用模板实现，要求学生：

（1）假设栈元素为字符型，修改主函数；

（2）重新设计测试数据，考查栈的上溢、下溢等情况，修改主函数。

**4. 实验程序**

在编程环境下新建一个工程“链栈验证实验”，并新建相应文件，文件包括链栈结构体的定义，范例程序如下：

typedef int DataType; /\*栈元素的数据类型，假设为int型\*/

typedef struct Node

{

DataType data; /\*存放栈元素的数据域\*/

struct Node \*next; **/\***存放下一个结点的地址**\*/**

} Node;

**Node \*top;** **/\***栈顶指针**\*/**

文件包括链栈初始化、入栈、出栈、获取栈顶元素、判空操作成员函数的定义，范例程序如下：

void InitStack(Node \*top)

{

top = NULL;

}

void Push(Node \*top, DataType x)

{

Node \*s = (Node \*)malloc(sizeof(Node)); /\*申请一个结点s\*/

s->data = x;

s->next = top; top = s; /\*将结点s插在栈顶\*/

}

int Pop(Node \*top, DataType \*ptr)

{

Node \*p = top;

if (top == NULL) {printf("下溢错误，删除失败\n"); return 0; }

\*ptr = top->data; /\*存储栈顶元素\*/

top = top->next; /\*将栈顶结点摘链\*/

free(p);

return 1;

}

int GetTop(**Node \*top,** DataType \*ptr)

{

if (top == NULL) {printf("下溢错误，取栈顶失败\n"); return 0; }

\*ptr = top->data; return 1;

}

int Empty(Node \*top)

{

if (top == NULL) return 1; /\*栈空则返回1\*/

else return 0;

}

在定义了链栈的存储结构并实现了基本操作后，可以调用实现基本操作的函数来完成相应的功能。范例程序如下：

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

/\*将单链表的结点结构定义和链栈的各个函数定义放到这里\*/

int main( )

{

DataType x;

Node \*top = NULL; /\*定义链栈的栈顶指针并初始化\*/

InitStack(top); /\*初始化链栈\*/

printf("对15和10执行入栈操作，");

Push(top, 15);

Push(top, 10);

if (GetTop(**top,** &x) == 1)

printf("当前栈顶元素为：%d\n", x); /\*输出当前栈顶元素10\*/

if (Pop(top, &x) == 1)

printf("执行一次出栈操作，删除元素：%d\n ", x); /\*输出出栈元素10\*/

if (GetTop(top, &x) == 1)

printf("当前栈顶元素为：%d\n", x); /\*输出当前栈顶元素15\*/

printf("请输入待插入元素：");

scanf("%d", &x);

Push(&S, x);

if (Empty(top) == 1)

printf("栈为空\n");

else

printf("栈非空\n"); /\*栈有2个元素，输出栈非空\*/

DestroyStack(top);

return 0;

}

# 3、前序遍历二叉树

**1. 实验目的**

⑴ 掌握二叉树的逻辑结构；

⑵ 掌握二叉树的二叉链表存储结构；

⑶ 验证二叉树的二叉链表存储及遍历操作。

**2. 实验内容**

⑴ 建立一棵含有*n*个结点的二叉树，采用二叉链表存储；

⑵ 输出前序遍历该二叉树的遍历结果。

**3. 实现提示**

定义二叉树的数据类型——二叉树结点结构体BiNode，在BiNode基础上实现题目要求的建立二叉链表、前序遍历等基本操作。建立二叉链表可以采用扩展二叉树的一个遍历序列，例如前序序列，将扩展二叉树的前序序列由键盘输入，建立该二叉树的二叉链表存储。

简单起见，本实验假定二叉树的数据元素为char型，要求学生：

（1）将实验程序调试通过后，用模板类改写；

（2）加入层序遍历二叉树等基本操作。

**4. 实验程序**

在编程环境下新建一个工程“二叉链表验证实验”，并新建相应文件，文件包括二叉树结构体的定义，范例程序如下：

typedef char DataType;

typedef struct BiNode

{

DataType data;

struct BiNode \*lchild, \*rchild;

} BiNode;

BiNode \*root;

文件包括建立二叉链表、前序遍历操作成员函数的定义，范例程序如下：

BiNode \*CreatBiTree(BiNode \*root)

{

char ch;

cin >> ch; /\*输入结点的数据信息\*/

if (ch == '#')

root = NULL; /\*递归结束，建立一棵空树\*/

else

{

root = (BiNode \*)malloc(sizeof(BiNode)); /\*生成新结点\*/

root->data = ch; /\*新结点的数据域为ch\*/

root->lchild = CreatBiTree(root->lchild); /\*递归建立左子树\*/

root->rchild = CreatBiTree(root->rchild); /\*递归建立右子树\*/

}

return root;

}

void PreOrder(BiNode \*root) //前序遍历

{

if (root == NULL)

return; /\*递归调用的结束条件\*/

else

{ //根左右

printf("%c ", root->data); /\*访问根结点的数据域，为char型\*/

PreOrder(root->lchild); /\*前序递归遍历root的左子树\*/

PreOrder(root->rchild); /\*前序递归遍历root的右子树\*/

}

}

在定义了二叉树的存储结构并实现了基本操作后，可以调用实现基本操作的函数来完成相应的功能。范例程序如下：

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

/\*将二叉链表的结点结构定义和各个函数定义放到这里\*/

int main( )

{

BiNode \*root = NULL; /\*定义二叉树的根指针变量\*/

root = CreatBiTree(root); /\*建立一棵二叉树\*/

printf("该二叉树的根结点是：%c\n", root->data);

printf("\n该二叉树的前序遍历序列是：");

PreOrder(root);

return 0;

}

# 4、图的深度优先遍历算法

**1. 实验目的**

⑴ 掌握图的逻辑结构；

⑵ 掌握图的邻接矩阵存储结构；

⑶ 验证图的邻接矩阵存储及其深度优先遍历操作的实现。

**2. 实验内容**

⑴ 建立无向图的邻接矩阵存储；

⑵ 对建立的无向图，进行深度优先遍历；

**3. 实现提示**

定义邻接矩阵存储的无向图结构体MGraph，在其基础上实现题目要求的图建立、深度优先遍历等基本操作。

**4. 实验程序**

在编程环境下新建一个工程“图的深度优先遍历验证实验”，并新建相应文件，文件包括图的邻接矩阵结构体MGraph的定义，范例程序如下：

typedef char DataType; /\*图中顶点的数据类型，假设为char型\*/

typedef struct /\*定义邻接矩阵存储结构\*/

{

DataType vertex[MaxSize]; /\*存放顶点的一维数组\*/

int edge[MaxSize][MaxSize]; /\*存放边的二维数组\*/

int vertexNum, edgeNum; /\*图的顶点数和边数\*/

} MGraph;

文件包括建立图、图的深度优先遍历操作成员函数的定义，范例程序如下：

void CreatGraph(MGraph \*G, DataType a[ ], int n, int e)

{

int i, j, k;

G->vertexNum = n; G->edgeNum = e;

for (i = 0; i < G->vertexNum; i++) /\*存储顶点信息\*/

G->vertex[i] = a[i];

for (i = 0; i < G->vertexNum; i++) /\*初始化邻接矩阵\*/

for (j = 0; j < G->vertexNum; j++)

G->edge[i][j] = 0;

for (k = 0; k < G->edgeNum; k++) /\*依次输入每一条边\*/

{

scanf("%d%d", &i, &j); /\*输入边依附的顶点编号\*/

G->edge[i][j] = 1; G->edge[j][i] = 1; /\*置有边标志\*/

}

}

void DFSTraverse(MGraph \*G, int v) /\*全局数组变量visited[n]已初始化为0\*/

{

printf("%c ", G->vertex[v]); visited[v] = 1;

for (int j = 0; j < G->vertexNum; j++)

if (G->edge[v][j] == 1 && visited[j] == 0)

DFSTraverse(G, j);

}

在定义了图的邻接矩阵存储结构并实现了基本操作后，可以调用实现基本操作的函数来完成相应的功能。范例程序如下：

#include <stdio.h>

#include <stdlib.h>

#define MaxSize 10 /\*假设图中最多顶点个数\*/

int visited[MaxSize]={0}; /\*全局数组变量visited初始化\*/

/\*把邻接矩阵的存储结构定义和各个函数定义放到这里\*/

int main( )

{

int i;

char ch[ ]={'A','B','C','D','E'};

MGraph MG;

CreatGraph(&MG, ch, 5, 6); /\*建立具有5个顶点6条边的无向图\*/

for (i = 0; i < MaxSize; i++)

visited[i] = 0;

printf("深度优先遍历序列是：");

DFSTraverse(&MG, 0); /\*从顶点0出发进行深度优先遍历\*/

return 0;

}

# 5、散列查找

**1. 实验目的**

⑴ 掌握散列查找的基本思想；

⑵ 掌握闭散列表的构造方法；

⑶ 掌握线性探测处理冲突的方法；

⑷ 验证散列技术的查找性能。

**2. 实验内容**

⑴ 对于给定的一组整数和散列函数，采用线性探测法处理冲突构造散列表；

⑵ 设计查找算法，验证查找性能。

**3. 实现提示**

首先将待查找集合存储到闭散列表ht中，然后随机生成待查元素的下标，考查在查找成功情况下的比较次数。

**4. 实验程序**

由于程序比较简单，使用单文件结构即可。新建文件“散列查找”，注意从下标0开始存放待查找元素，范例程序如下：

int HashSearch1(int ht[ ], int m, int k, int \*p) /\*形参p传指针，返回位置\*/

{

int i, j, flag = 0; /\*flag=0表示散列表未满\*/

j = H(k); /\*计算散列地址\*/

i = j; /\*记载比较的起始位置\*/

while (ht[i] != 0 && flag == 0)

{

if (ht[i] == k) { /\*比较若干次查找成功\*/

\*p = i; return 1;

}

else i = (i + 1) % m; /\*向后探测一个位置\*/

if (i == j) flag = 1; /\*表已满\*/

}

if (flag == 1) {printf("溢出"); exit(-1);} /\*表满，产生溢出\*/

else { /\*比较若干次查找不成功，插入\*/

ht[i] = k; \*p = i; return 0;

}

}