LaTeX *Data Structure & Algorithms*

Sun for morning, moon for night, and you forever.

作者：Lyshmily.Y & 木易

组织：LyshmilyY.Y

时间：July 21, 2024

版本：V.1.0

邮箱：yjlpku.outlook.com & 845307723@qq.com

在没有结束前，总要做很多没有意义的事，这样才可以在未来某一天，用这些无意义的事去堵住那些讨厌的缺口

目录

第 1 章 绪论

1.1 数据结构

定义 1.1.1 (基本术语)

1. 数据 (**Data**): 客观事物的符号, 是所有能输入到计算机中并被计算机程序处理的符号的集合.
2. 数据元素 (**Data Element**): 数据的基本单位, 也被称为元素、记录, 数据元素用于完整地描述一个对象, 比如一名学生记录、树中棋盘的一个格局和图中的一个顶点.
3. 数据项 (**Data Item**): 是组成数据元素的、有独立含义的、不可分割的最小单位. 一个数据元素可以由一个或多个数据项组成. 比如一名学生记录中包含学号、姓名、性别、年龄等数据项.
4. 数据对象 (**Data Object**): 性质相同的数据元素的集合, 是数据的一个子集. 比如学生集合、图中的顶点集合.
5. 数据结构 (**Data Structure**): 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

定义 1.1.2 (数据结构三要素)

1. 逻辑结构: 集合、线性结构 (一对一)、树形结构 (一对多)、图形结构 (多对多)
2. 物理结构: 顺序存储结构 (逻辑和物理都相邻)、链式存储结构 (指针表示)、散列存储
3. 数据运算: 初始化、插入、删除、查找、更改、遍历

定义 1.1.3 (数据类型和抽象数据类型)

1. 数据类型 (**Data Type**): 一个值的集合和定义在此集合上的一组操作的总称
2. 抽象数据类型 (**Abstract Data Type, ADT**): 是指一个数学模型以及定义在此数学模型上的一组操作, 具体包括数据对象、数据对象上关系的集合以及对数据对象的基本操作的集合

1.2 算法和算法分析

定义 1.2.1 (算法的定义及其特性)

一、算法: 是解决特定问题求解步骤的描述, 是指令的有限序列

二、算法的特性:

1. 有穷性: 算法在执行有限步之后终止
2. 确定性: 算法的每一步骤都有确切的含义, 不会出现二义性
3. 可行性: 算法的每一步都是可行的, 可以通过执行有限次完成
4. 输入: 一个算法有零个或多个输入
5. 输出: 一个算法有一个或多个输出

三、评价算法优劣的标准:

1. 正确性: 算法的输出结果符合要求
2. 可读性: 算法是容易阅读和理解的
3. 健壮性: 算法对不合理数据有较好的处理能力
4. 高效性: 包括时间和空间两个方面, 时间高效指的是算法设计合理, 执行效率高, 空间高效指的是算法执行过程中所需的存储空间最小

第 2 章 线性表

2.1 线性表的定义和基本操作

定义 2.1.1 (线性表)

$$L = (a_1, a_2, a_3, \ldots, a_i, a_{i+1}, \ldots, a_n)$$

1. 有相同数据类型的 n 个数据元素的有限序列
2. 存在唯一的一个被称作“第一个”的数据元素
3. 存在唯一的一个被称作“最后一个”的数据元素
4. 除第一个外，每个数据元素有且仅有一个直接前驱
5. 除最后一个外，每个数据元素有且仅有一个直接后继

定义 2.1.2 (线性表基本操作)

1. **InitList(&L)** 初始化线性表，构造一个空的线性表 L
2. **DestroyList(&L)** 销毁线性表，销毁线性表 L
3. **ListInsert(&L, i, e)** 插入操作，在线性表 L 的第 i 个位置插入元素 e
4. **ListDelete(&L, i, &e)** 删除操作，删除线性表 L 的第 i 个位置的元素，并用 e 返回其值
5. **GetElem(L, i, &e)** 取值操作，返回线性表 L 的第 i 个位置的元素
6. **LocateElem(L, e)** 定位操作，在线性表 L 中查找与给定值 e 相等的元素
7. **Length(L)** 求表长，返回线性表 L 的元素个数
8. **Empty(L)** 判空操作，若 L 为空表，则返回 TRUE，否则返回 FALSE
9. **PrintList(L)** 输出操作，按顺序输出线性表 L 的所有元素

2.2 线性表的顺序表示

2.2.1 顺序表

定义 2.2.1 (顺序表)

1. 用顺序存储的方式实现线性表
2. 顺序存储: 把逻辑上相邻的元素存储在物理上相邻的存储单元中, 元素之间的关系由存储单元之间的邻接关系来体现

common.h

```c
#include <assert.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <math.h>
#define OK 1
#define ERROR 0
#define OVERFLOW -2

typedef int Status;

// 定义数据域
typedef struct elem
{
    char name[20];
    int Math;
    int English;
    int Politics;
    int Computer;
}elem;
```

SqList.h

```
#define INITSIZE 10
// 定义顺序表
typedef struct SqList
{
    int length;
    int MaxSize;
    elem* data;
}SqList;

// 函数声明

Status InitList(SqList * L); // 初始化
Status IncreaseSize(SqList * L, int len); // 增加表长
Status ListInsert(SqList * L, elem e, int i); // 插入
Status ListDelete(SqList * L, int i, elem * e); // 删除
int LocateElem(SqList L, elem e); // 按值查找
Status GetElem(SqList L, int i, elem * e); // 按位查找
int Length(SqList L); // 表长
Status Empty(SqList L);// 判空
void PrintList(SqList L); // 打印
void Printelem(elem e); // 打印单个数据元素
```

2.2.1.1 初始化顺序表

InitList

```
// 初始化
Status InitList(SqList * L)
{
    L->data = malloc(INITSIZE * sizeof(elem));
    if (!L->data) exit(OVERFLOW);
    L->length = 0;
    return OK;
}
```

2.2.1.2 增加顺序表

IncreaseSize

```
// 增加表长
Status IncreaseSize(SqList * L, int len)
{
    elem *p = L->data;
    L->data = (elem *)malloc((L->MaxSize + len) * sizeof(elem));
    if (!L->data) exit(OVERFLOW);
    for (int i = 0; i < L->length; i++)
    {
        L->data[i] = p[i];
    }
    L->MaxSize += len;
    free(p);
    return OK;
}
```

2.2.1.3 插入顺序表

ListInsert

```
// 插入
Status ListInsert(SqList * L, int i, elem e)
{
    if (i < 1 || i > L->length + 1) return ERROR;
    if (L->length >= L->MaxSize) IncreaseSize(L, INITSIZE);
    for (int j = L->length - 1; j >= i-1; j--)
    {
        L->data[j + 1] = L->data[j];
    }
    L->data[i - 1] = e;
    L->length++;
    return OK;
}
```

2.2.1.4 删除顺序表

ListDelete

```
// 删除
Status ListDelete(SqList * L, int i, elem * e)
{
    if (i < 1 || i > L->length) return ERROR;
    *e = L->data[i - 1];
    for (int j = i; j < L->length; j++)
    {
        L->data[j - 1] = L->data[j];
    }
    L->length--;
    return OK;
}
```

2.2.1.5 顺序表查找

LocateElem & GetElem

```
// 按值查找
int LocateElem(SqList L, elem e)
{
    for (int i = 0; i < L.length; i++)
    {
        if (L.data[i].name == e.name)
            return i + 1;
    }
    return ERROR;
}

// 按位查找
Status GetElem(SqList L, int i, elem * e)
{
    if (i < 1 || i > L.length) return ERROR;
    *e = L.data[i - 1];
    return OK;
}
```

2.2.1.6 顺序表辅助函数

Length & Empty & PrintList

```
// 表长
int Length(SqList L)
{
    return L.length;
}
// 判空
Status Empty(SqList L)
{
    if (L.length == 0) return OK;
    else return ERROR;
}
// 打印
void PrintList(SqList L)
{
    printf("%5s %15s %15s %15s %20s\n",
    "Name", "Score:Math", "Score:English",
    "Score:Politics", "Score:Computer");
    for (int i = 0; i < L.length; i++)
    {
        printf("%5s %10d %15d %15d %20d\n",
        L.data[i].name,L.data[i].Math,L.data[i].English,
        L.data[i].Politics,L.data[i].Computer);
    }
    printf("\n");
}

void Printelem(elem e)
{
    printf("%5s %15s %15s %15s %20s\n",
    "Name", "Score:Math", "Score:English",
    "Score:Politics", "Score:Computer");
    printf("%5s %10d %15d %15d %20d\n",
    e.name,e.Math,e.English,e.Politics,e.Computer);
    printf("\n");
}
```

2.2.1.7 顺序表实例化

EgSqList.c

```c
int main()
{
    SqList L;
    elem student;
    InitList(&L);
    for (int i = 0; i < 5; i++)
    {
        scanf("%s %d %d %d %d",student.name, &(student.Math),
        &(student.English),&(student.Politics),&(student.Computer));
        ListInsert(&L,i+1,student);
    }
    printf(" 初始顺序表:\n");
    PrintList(L);
    strcpy(student.name,"LY");
    student.Math = 150;
    student.English = 85;
    student.Politics = 75;
    student.Computer = 145;
    ListInsert(&L,2,student);
    printf(" 在第二个位置插入新元素:\n");
    PrintList(L);
    ListDelete(&L,3,&student);
    printf(" 删除第三个元素:\n");
    Printelem(student);
    printf(" 删除后的顺序表:\n");
    PrintList(L);
    return 0;
}
```

data.txt

```
Amy 123 86 74 143
Bob 135 75 81 108
Dav 118 74 80 134
Fab 98  56 67 120
Joy 108 64 70 118
```

output

```
./EgSqList < data.txt
初始顺序表:
Name       Score:Math    Score:English  Score:Politics       Score:Computer
Amy          123             86             74                143
Bob          135             75             81                108
Dav          118             74             80                134
Fab           98             56             67                120
Joy          108             64             70                118

在第二个位置插入新元素:
Name       Score:Math    Score:English  Score:Politics       Score:Computer
Amy          123             86             74                143
LY           150             85             75                145
Bob          135             75             81                108
Dav          118             74             80                134
Fab           98             56             67                120
Joy          108             64             70                118
```

output

```
删除第三个元素:
Name       Score:Math    Score:English  Score:Politics       Score:Computer
Bob         135             75             81                  108

删除后的顺序表:
Name       Score:Math    Score:English  Score:Politics       Score:Computer
Amy         123             86             74                  143
LY          150             85             75                  145
Dav         118             74             80                  134
Fab          98             56             67                  120
Joy         108             64             70                  118
```

2.3 线性表的链式表示

2.3.1 单链表

定义 2.3.1 (单链表)

1. 单链表第一个元素在删除和增加需要特殊处理
2. 头插法和尾插法的区别, 头插法实现反置
3. 在增删、初始化操作传入引用, 在判空、打印以及求表长传入值

LNode

```
typedef struct LNode
{
    elem data;
    struct LNode *next;
}LNode,*LinkList;
```

SlList.h

```
// 函数声明
Status InitList(LinkList *L); // 初始化
Status Empty(LinkList L);// 判空
LNode * LocateElem(LinkList L, elem e); // 按值查找
LNode * GetElem(LinkList *L, int i); // 按位查找
Status InsertNextNode(LNode *p,elem e); //指定元素后插入
Status InsertPreNode(LNode *p,elem e); //指定元素前插入
Status ListInsert(LinkList *L, int i, elem e); // 按位置插入
Status ListDelete(LinkList *L, int i, elem * e); // 按位置删除
Status DeleteNode(LNode *p); // 删除指定节点
int Length(LinkList L); // 表长
void PrintList(LinkList L); // 打印链表
void PrintNode(LNode p); // 打印节点
LinkList TailInsert(LinkList *L,int n); // 尾插法
LinkList HeadInsert(LinkList *L,int n); // 头插法
void ListReverse(LinkList *L); //翻转链表
```

2.3.1.1 初始化单链表

InitList

```
// 初始化
Status InitList(LinkList *L)
{
    *L = (LNode *) malloc (sizeof(LNode));
    if (*L == NULL)
        return ERROR;
    (*L) ->next = NULL;
    return OK;
}
// 判空
Status Empty(LinkList L)
{
    if (L->next == NULL)
        return OK;
    return ERROR;
}
```

2.3.1.2 单链表查找

Find

```
// 按值查找
LNode * LocateElem(LinkList L, elem e)
{
    LNode *p = L->next;
    while(p != NULL && p->data.name != e.name)
        p = p->next;
    return p;
}
// 按位查找
LNode * GetElem(LinkList *L, int i)
{
    if (i < 1)
        return NULL;
    LNode *p = *L;
    // 第 j 个节点 p
    int j = 0;
    // 找到第 i 个元素的位置
    while (p !=NULL && j < i)
    {
        p = p->next;
        j++;
    }
    return p;
}
```

2.3.1.3 单链表插值

Insert 辅助函数

```
//指定元素后插入
Status InsertNextNode(LNode *p,elem e)
{
    LNode *s = (LNode *)malloc(sizeof(LNode));
    if (s == NULL || p == NULL)
        return ERROR;
    s->data = e;
    s->next = p->next;
    p->next = s;
    return OK;
}
//指定元素前插入
Status InsertPreNode(LNode *p,elem e)
{
    LNode *s = (LNode *)malloc(sizeof(LNode));
    if (s == NULL || p == NULL)
        return ERROR;
    s->next = p->next;
    p->next = s;
    s->data = p->data;
    p->data = e;
    return OK;
}
```

ListInsert

```
// 按位置插入
Status ListInsert(LinkList *L, int i, elem e)
{
    if (i < 1)
        return ERROR;
    LNode *p = (LNode *)malloc(sizeof(LNode));
    if (p == NULL)
        return ERROR;
    if (i ==1)
        {
            p->data = e;
            p->next = (*L)->next;
            (*L)->next = p;
            return OK;
        }
    p = GetElem(L,i-1);
    if (p == NULL)
        return ERROR;
    return InsertNextNode(p,e);
}
```

2.3.1.4 单链表删除

ListDelete

```
// 删除
Status ListDelete(LinkList *L, int i, elem * e)
{
    if (i < 1)
        return ERROR;
    if (i == 1)
    {
        if ((*L)->next == NULL)
            return ERROR;
        *e = (*L)->next->data;
        (*L)->next = (*L)->next->next;
        return OK;
    }
    LNode *p = GetElem(L,i-1);
    if (p == NULL || p->next == NULL)
        return ERROR;
    LNode * q = p->next;
    *e = q->data;
    p->next =q->next;
    //释放删除节点空间
    free(q);
    return OK;
}
```

2.3.1.5 单链表辅助函数

Length & PrintList

```
// 表长
int Length(LinkList L)
{
    int len =0;
    LNode *p = L;
    while(p->next != NULL)
    {   p = p->next;
        len++;}
    return len;
}
// 打印
void PrintList(LinkList L)
{
    printf("%5s %15s %15s %15s %20s\n",
    "Name", "Score:Math", "Score:English",
    "Score:Politics", "Score:Computer");
    LNode *p = L->next;
    while(p != NULL)
    {   printf("%5s %10d %15d %15d %20d\n",
        p->data.name,p->data.Math,p->data.English,
        p->data.Politics,p->data.Computer);
        p = p->next;}
    printf("\n");
}
// 打印单个数据元素
void PrintNode(LNode p)
{
    printf("%5s %15s %15s %15s %20s\n","Name",
    "Score:Math", "Score:English",
     "Score:Politics", "Score:Computer");
    printf("%5s %10d %15d %15d %20d\n",
    p.data.name,p.data.Math,p.data.English,
    p.data.Politics,p.data.Computer);
    printf("\n");
}
```

2.3.1.6 头插法 & 尾插法

TailInsert & HeadInsert

```
// 尾插法实现单链表
LinkList TailInsert(LinkList *L,int n)
{
    LNode *r = *L;
    for (int i = 0; i < n; i++)
    {
        LNode *p = (LNode*)malloc(sizeof(LNode));
        scanf("%s %d %d %d %d\n",
        p->data.name,&(p->data.Math),&(p->data.English),
        &(p->data.Politics),&(p->data.Computer));
        p->next = r->next;
        r->next = p;
        r = p;
    }
    return *L;
}
// 头插法实现单链表
LinkList HeadInsert(LinkList *L,int n)
{
    for (int i = 0; i < n; i++)
    {
        LNode *p = (LNode*)malloc(sizeof(LNode));
        scanf("%s %d %d %d %d\n",
        p->data.name,&(p->data.Math),&(p->data.English),
        &(p->data.Politics),&(p->data.Computer));
        p->next = (*L)->next;
        (*L)->next = p;
    }
    return *L;
}
```

ListReverse

```
// 反转单链表
void ListReverse(LinkList *L)
{
    LNode *p,*q;
    p = (*L)->next;
    (*L)->next = NULL;
    while(p != NULL)
    {
        q = p->next;
        p->next = (*L)->next;
        (*L)->next = p;
        p = q;
    }
}
```

2.3.1.7 单链表实例化

EgSlList

```
int main()
{
    LinkList L;
    LNode p;
    elem student;
    InitList(&L);
    printf(" 尾插法建立单链表:\n");
    TailInsert(&L,5);
    PrintList(L);
    printf(" 反转单链表:\n");
    ListReverse(&L);
    PrintList(L);
    printf(" 删除第三个元素后的单链表:\n");
    ListDelete(&L,3,&student);
    PrintList(L);
    printf(" 被删除的元素:\n");
    p.data = student;
    PrintNode(p);
    printf(" 单链表第一个位置插入:\n");
    ListInsert(&L,1,student);
    PrintList(L);
    return 0;
}
```

output

```
./SlList < data.txt
尾插法建立单链表:
Name    Score:Math    Score:English    Score:Politics    Score:Computer
Amy         123             86               74               143
Bob         135             75               81               108
Dav         118             74               80               134
Fab          98             56               67               120
Joy         108             64               70               118

反转单链表
Name    Score:Math    Score:English    Score:Politics    Score:Computer
Joy         108             64               70               118
Fab          98             56               67               120
Dav         118             74               80               134
Bob         135             75               81               108
Amy         123             86               74               143
```

output

```
删除第三个元素后的单链表:
Name    Score:Math    Score:English    Score:Politics    Score:Computer
Joy         108             64               70               118
Fab          98             56               67               120
Bob         135             75               81               108
Amy         123             86               74               143

被删除的元素:
Name    Score:Math    Score:English    Score:Politics    Score:Computer
Dav         118             74               80               134

单链表第一个位置插入:
Name    Score:Math    Score:English    Score:Politics    Score:Computer
Dav         118             74               80               134
Joy         108             64               70               118
Fab          98             56               67               120
Bob         135             75               81               108
Amy         123             86               74               143
```

2.3.2 双链表

定义 2.3.2 (双链表)

1. 双链表有两个指针域, 分别指向直接前驱和直接后继
2. 头插法和尾插法的区别, 头插法实现反置
3. 在插入和删除时, 第一个和最后一个节点有特殊情况

DNode

```
typedef struct DNode
{
    elem data;
    struct DNode *prior;
    struct DNode *next;
}DNode,*DLinkList;
```

DlList.h

```
// 函数声明
Status InitList(DLinkList *L); // 初始化
Status Empty(DLinkList L);// 判空
DNode * LocateElem(DLinkList L, elem e); // 按值查找
DNode * GetElem(DLinkList *L, int i); // 按位查找
Status InsertNextNode(DNode *p,elem e); //指定元素后插入
Status InsertPreNode(DNode *p,elem e); //指定元素前插入
Status ListInsert(DLinkList *L, int i, elem e); // 按位置插入
Status ListDelete(DLinkList *L, int i, elem * e); // 删除
Status DeleteNode(DNode *p); // 删除指定节点
int Length(DLinkList L); // 表长
void PrintList(DLinkList L); // 打印链表
void PrintNode(DNode p); // 打印节点
DLinkList TailInsert(DLinkList *L,int n); // 尾插法
DLinkList HeadInsert(DLinkList *L,int n); // 头插法
void ListReverse(DLinkList *L); //翻转链表
```

2.3.2.1 初始化双链表

InitList

```
// 初始化
Status InitList(DLinkList *L)
{
    (*L) = (DNode *)malloc(sizeof(DNode));
    if ((*L) == NULL)
        return ERROR;
    (*L)->prior = NULL;
    (*L)->next = NULL;
    return OK;
}
// 判空
Status Empty(DLinkList L)
{
    if (L->next == NULL)
        return OK;
    return ERROR;
}
```

2.3.2.2 双链表查找

Find

```
// 按值查找
DNode * LocateElem(DLinkList L, elem e)
{
    DNode *p = L->next;
    while (p != NULL)
    {
        if (p->data.name == e.name)
            return p;
        p = p->next;
    }
    return p;

}
// 按位查找
DNode * GetElem(DLinkList *L, int i)
{
    DNode * p = (*L);
    int j = 0;
    while (p->next != NULL && j<i)
    {
        p = p->next;
        j++;
    }
    return p;
}
```

2.3.2.3 双链表插值

Insert 辅助函数

```
// 指定元素后插入
Status InsertNextNode(DNode *p,elem e)
{
    DNode *q = (DNode*)malloc(sizeof(DNode));
    if (q == NULL || p == NULL)
        return ERROR;
    q->data = e;
    p->next->prior = q;
    q->next = p->next;
    q->prior = p;
    p->next = q;
    return OK;
}
//指定元素前插入
Status InsertPreNode(DNode *p,elem e)
{
    DNode *q = (DNode*)malloc(sizeof(DNode));
    if (q == NULL || p ==NULL)
        return ERROR;
    q->data = e;
    p->prior->next = q;
    q->prior = p->prior;
    q->next = p;
    p->prior = q;
    return OK;
}
```

ListInsert

```
// 按位置插入
Status ListInsert(DLinkList *L, int i, elem e)
{
    if (i < 1)
        return ERROR;
    DNode *p = (DNode*)malloc(sizeof(DNode));
    if (p == NULL)
        return ERROR;
    if (i == 1)
    {
        p->data = e;
        p->next = (*L)->next;
        (*L)->next->prior = p;
        (*L)->next = p;
        p->prior = (*L);
        return OK;
    }
    p = GetElem(L,i-1);
    return InsertNextNode(p,e);

}
```

2.3.2.4 双链表删除

ListDelete

```
// 删除
Status ListDelete(DLinkList *L, int i, elem * e)
{
    if (i < 1)
        return ERROR;
    DNode *p = (*L)->next;
    if (i == 1)
    {
        if (p == NULL)
            return ERROR;
        (*L)->next = p->next;
        *e = p->data;
        if (p->next != NULL)
            p->next->prior = (*L);
        free(p);
        return OK;
    }
    p = GetElem(L,i);
    if (p == NULL)
        return ERROR;
    *e = p->data;
    return DeleteNode(p);
}
```

2.3.2.5 双链表辅助函数

Length

```
// 表长
int Length(DLinkList L)
{
    int len = 0;
    while (L->next != NULL)
    {
        len++;
        L = L->next;
    }
    return len;
}
```

PrintList

```
// 打印链表
void PrintList(DLinkList L)
{
    printf("%5s %15s %15s %15s %20s\n",
    "Name", "Score:Math", "Score:English",
    "Score:Politics", "Score:Computer");
    DNode *p = L->next;
    while(p != NULL)
    {
        printf("%5s %10d %15d %15d %20d\n",
        p->data.name,p->data.Math,p->data.English,
        p->data.Politics,p->data.Computer);
        p = p->next;
    }
    printf("\n");
}
// 打印节点
void PrintNode(DNode p)
{
    printf("%5s %15s %15s %15s %20s\n",
    "Name", "Score:Math", "Score:English",
    "Score:Politics", "Score:Computer");
    printf("%5s %10d %15d %15d %20d\n",
    p.data.name,p.data.Math,p.data.English,
    p.data.Politics,p.data.Computer);
    printf("\n");
}
```

2.3.2.6 头插法 & 尾插法

TailInsert

```
// 尾插法
DLinkList TailInsert(DLinkList *L,int n)
{
    DNode *r = *L;
    for (int i = 0; i < n; i++)
    {
        elem e;
        scanf("%s %d %d %d %d\n",
        e.name,&(e.Math),&(e.English),&(e.Politics),&(e.Computer));
        DNode *p = (DNode*)malloc(sizeof(DNode));
        if (p == NULL)
            return NULL;
        p->data = e;
        p->next = r->next;
        r->next = p;
        p->prior = r;
        r = p;
    }
    return *L;
}
```

HeadInsert

```
// 头插法
DLinkList HeadInsert(DLinkList *L,int n)
{
    for (int i = 0; i < n; i++)
    {
        elem e;
        scanf("%s %d %d %d %d\n",
        e.name,&(e.Math),&(e.English),&(e.Politics),&(e.Computer));
        DNode *p = (DNode*)malloc(sizeof(DNode));
        if (p == NULL)
            return NULL;
        p->data = e;
        p->next = (*L)->next;
        p->prior = (*L);
        if ((*L)->next != NULL)
            (*L)->next->prior = p;
    }
    return *L;
}
```

ListReverse

```
//翻转链表
void ListReverse(DLinkList *L)
{
    DNode *p,*q;
    p = (*L)->next;
    (*L)->next = NULL;
    while(p != NULL)
    {
        q = p->next;
        p->next = (*L)->next;
        p->prior = (*L);
        if ((*L)->next != NULL)
            (*L)->next->prior = p;
        (*L)->next = p;
        p = q;
    }
}
```

2.3.2.7 双链表实例化

EgDlList

```
int main()
{
    DLinkList L;
    DNode p;
    elem student;
    InitList(&L);
    printf(" 尾插法建立双链表:\n");
    TailInsert(&L,5);
    PrintList(L);
    printf(" 反转双链表:\n");
    ListReverse(&L);
    PrintList(L);
    printf(" 删除第三个元素后的双链表:\n");
    ListDelete(&L,3,&student);
    PrintList(L);
    printf(" 被删除的元素:\n");
    p.data = student;
    PrintNode(p);
    printf(" 双链表第一个位置插入:\n");
    ListInsert(&L,1,student);
    PrintList(L);
    return 0;
}
```

output

```
./DlList < data.txt
尾插法建立双链表:
Name    Score:Math    Score:English    Score:Politics    Score:Computer
Amy         123             86               74               143
Bob         135             75               81               108
Dav         118             74               80               134
Fab          98             56               67               120
Joy         108             64               70               118

反转单链表
Name    Score:Math    Score:English    Score:Politics    Score:Computer
Joy         108             64               70               118
Fab          98             56               67               120
Dav         118             74               80               134
Bob         135             75               81               108
Amy         123             86               74               143
```

output

```
删除第三个元素后的单链表:
Name    Score:Math    Score:English    Score:Politics    Score:Computer
Joy        108             64              70               118
Fab         98             56              67               120
Bob        135             75              81               108
Amy        123             86              74               143

被删除的元素:
Name    Score:Math    Score:English    Score:Politics    Score:Computer
Dav        118             74              80               134

单链表第一个位置插入:
Name    Score:Math    Score:English    Score:Politics    Score:Computer
Dav        118             74              80               134
Joy        108             64              70               118
Fab         98             56              67               120
Bob        135             75              81               108
Amy        123             86              74               143
```

2.3.3 循环链表

定义 2.3.3 (循环链表)

1. 分为循环单链表和循环双链表, 与单链表和双链表的区别在于尾节点指向头节点
2. 判空操作与单链表、双链表存在差异
3. 插入、删除、以及判断表头、表尾节点有一定变化

2.3.4 静态链表

定义 2.3.4 (静态链表)

1. 使用数组的链表, 数组第一个元素当做头节点, 每个位置除了保存数据元素外, 还保存下一个元素的位置
2. 初始化时将所有位置的下一个元素位置清空
3. 插入、删除操作需要找到上一个节点的位置, 然后将下一个元素位置清空

2.4 Summary

2.4.1 顺序表与链表比较

表 2.1: 顺序表与链表比较

比较项目 \ 存储结构		顺序表
空间	存储空间	① 预先分配 ② 会导致空间闲置或溢出
	存储密度	① 存储密度 = 1 ② 不需要额外空间来表示节点的逻辑关系
时间	存取元素	① 随机存取 ② 按位置访问元素时间复杂度 $O(1)$
	插入、删除	平均移动一半的元素, 时间复杂度 $O(n)$
适用情况		① 表长变化不大, 确定变化范围 ② 很少进行插入或删除, 经常按照元素位置序号访问数据

比较项目 \ 存储结构		链表
空间	存储空间	① 动态分配 ② 不会出现空间闲置或溢出
	存储密度	① 存储密度 < 1 ② 需要借助指针来表示节点的逻辑关系
时间	存取元素	① 顺序存取 ② 按位置访问元素时间复杂度 $O(n)$
	插入、删除	不需要移动元素, 确定插入或删除位置后, 时间复杂度 $O(n)$
适用情况		① 表长变化很大 ② 频繁进行插入或删除操作

2.4.2 单链表、双链表、循环链表比较

表 2.2: 单链表、双链表、循环链表比较

操作名称 / 链表名称	查找 表头节点	查找 表尾节点	查找节点 *p 前驱节点
带头节点 单链表 L	**L−>next** 时间复杂度 $O(1)$	从 **L−>next** 依次向后遍历 时间复杂度 $O(n)$	无法找到前驱节点
带头节点头指针 L 循环单链表	**L−>next** 时间复杂度 $O(1)$	从 **L−>next** 依次向后遍历 时间复杂度 $O(n)$	**p−>next** 可以找到前驱节点 时间复杂度 $O(n)$
带头节点尾指针 R 循环单链表	**R−>next** 时间复杂度 $O(1)$	**R** 时间复杂度 $O(1)$	**p−>next** 可以找到前驱节点 时间复杂度 $O(n)$
带头节点 双向循环链表 L	**L−>next** 时间复杂度 $O(1)$	**L−>prior** 时间复杂度 $O(1)$	**p−>prior** 可以找到前驱节点 时间复杂度 $O(1)$

第 3 章 栈和队列