南京邮电大学

实 验 报 告

(2017 / 2018 学年 第 一 学期)

课程名称	数据结构A
实验名称	线性表的基本运算及多项式的算术运算
实验时间	2017年9月22日
指导单位	计算机软件与工程学院
指导老师	邹志强

学生姓名	柏超宇	班级学号	Q15010125
学院	贝尔英才学院	专业	信息安全

实验名称	线性表的基本运算及多项式的算术运算			指导教师	邹志强
实验类型	验证	实验学时	2+2	实验时间	2017.9.22

一、实验目的和要求:

目的:

1.掌握线性表的两种基本存储结构及应用场合：顺序存储和链接存储。

2.掌握顺序表和链表的各种基本操作算法。

3.理解线性表应用于多项式的实现算法。

要求:

能够正确演示线性表的查找、插入、删除运算。实现多项式的加法和乘法运算操作。

二、实验环境(实验设备)

macOS Sierra10.12.6

sublime text 3 + Sublime Clang

三、实验原理及内容

原理:

线性表（Linear List）是由n（n≥0）个数据元素（结点）a[0]，a[1]，a[2]…，a[n-1]组成的有限序列。

其中:

数据元素的个数n定义为表的长度 = "list".length() （"list".length() = 0(表里没有一个元素）时称为空表）

将非空的线性表（n>=0）记作：（a[0]，a[1]，a[2]，…，a[n-1]）

数据元素a[i]（0≤i≤n-1）只是个抽象符号，其具体含义在不同情况下可以不同

一个数据元素可以由若干个数据项组成。数据元素称为记录，含有大量记录的线性表又称为文件。这种结构具有下列特点：存在一个唯一的没有前驱的（头）数据元素；存在一个唯一的没有后继的（尾）数据元素；此外，每一个数据元素均有一个直接前驱和一个直接后继数据元素。[1]

实验内容一：

参照程序2.1~程序2.7，编写程序，完成顺序表的初始化、查找、插入、删除、输出、撤销等操作。

定义出错码、结构定义操作如下：

```
#include <stdio.h>
#include <stdlib.h>
#define ERROR 0
#define OK 1
#define Overflow 2
#define Underflow 3
#define NotPresent 4
#define Duplicate 5
typedef struct {
    int n;
    int maxLength;
    int *element;
} SeqList;
SeqList L;
```

顺序表初始化操作如下：

```
int Init (SeqList *L, int mSize) {
    L->maxLength = mSize;
    L->n = 0;
    L->element = (int *)malloc(sizeof(int) * mSize);
    if (!L->element) {
        return ERROR;
    }
    return OK;
}
```

顺序表寻找元素操作如下：

```
int Find (SeqList L , int i , int *x) {
    if (i < 0 || i > L.n - 1) {
        return ERROR;
    }
    *x = L.element[i];
    return OK;
}
```

顺序表插入操作如下：

```
int Insert(SeqList *L , int i , int x) {
    int j;
```

```
    if (i < -1 || i > L->n - 1) {
        return ERROR;
    }
    if (L->n == L->maxLength) {
        return ERROR;
    }
    for(j = L->n - 1 ; j > i ; j++) {
        L->element[j + 1] = L->element[j];
    }
    L->element[i + 1] = x;
    L->n++;
    return OK;
}
```

顺序表删除操作如下：

```
int Delete(SeqList *L , int i) {
    int j;
    if(i < 0 || i > L-> n - 1)
        return ERROR;
    if(!L->n)
        return ERROR;
    for(j = i + 1; j < L->n; j++) {
        L->element[j - 1] = L->element[j];
    }
    L->n--;
    return OK;
}
```

顺序表输出操作如下：

```
int Output(SeqList L) {
    int i;
    if (!L.n) return ERROR;
    for(i = 0; i <= L.n - 1; i++){
        printf("%d ", L.element[i]);
    }
    return OK;
}
```

顺序表释放内存操作如下：

```
void Destroy (SeqList *L){
    (*L).n = 0;
    (*L).maxLength = 0;
    free((*L).element);
}
```

函数调用方式如下：

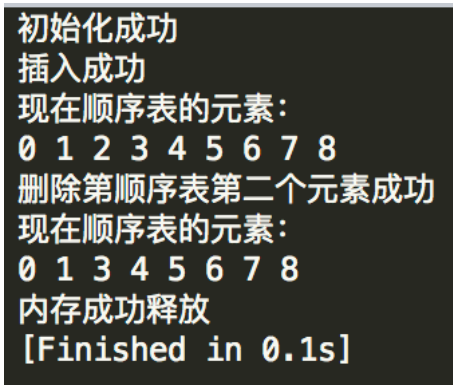
需要实现的功能	对应的语句
定义一个名称叫做list的顺序表	SeqList list
初始化list，使它最大长度为maxn	Init(&list, maxn);
在list的第i个位置插入元素x	Insert(&list, i - 1, x);
删除list的第i个元素	Delete(&list, i);
将list中的元素全部输出	Output(list);
释放list中的内存	Destroy(&list)

功能测试：

```
int main() {
    SeqList list;
    if(Init(&list, 100)){
        printf("初始化成功\n");
    };
    for(int i = 0; i < 9; i++) {
        Insert(&list, i-1, i);
    }
    printf("插入成功\n");
    printf("现在顺序表的元素：\n");
    Output(list);
    printf("\n");
    Delete(&list, 2);
    printf("删除第顺序表第二个元素成功\n");
    printf("现在顺序表的元素：\n");
    Output(list);
    printf("\n");
    Destroy(&list);
    printf("内存成功释放\n");
}
```

测试说明：首先初始化一个一个最大长度为100的list，再按顺序插入9个整数，删除其中第二个元素，最后释放顺序表占用的内存空间。并输出每一步操作后表中的元素，进行验证。

运行结果：



结果与我们预料之中的相同。说明上述算法实现了我们想要的功能。

实验内容二：

参照程序2.8~2.14，编写程序，完成带表头节点单链表的初始化、查找、插入、删除、输出、撤销、反转、排序操作

定义出错码、结构定义操作如下：

```
#include <stdio.h>
#include <stdlib.h>
#define ERROR 0
#define OK 1
#define Overflow 2
#define Underflow 3
#define NotPresent 4
#define Duplicate 5
typedef struct Node {
    int element;
    struct Node *link;
} Node;
typedef struct {
    struct Node *first;
    int n;
} SingleList;
```

带表头节点单链表初始化：

```
int Init(SingleList *L) {
    L->first = (Node *)malloc(sizeof(Node));
    if(!L->first) return ERROR;
    L->first->link = NULL;
    L->n = 0;
    L->first->element = 99;
    return OK;
```

```
}
```

单链表插入操作：

```
int Insert(SingleList *L, int i, int x) {
    Node *p, *q;
    int j;
    if(i < -1 || i > L->n - 1) return ERROR;
    p = L->first->link;
    for (j = 0; j < i ; j++) p = p->link;
    q = (Node *)malloc(sizeof(Node));
    q->element = x;
    if(i > -1) {
        q->link = p->link;
        p->link = q;
    } else {
        q->link = L->first->link;
        L->first->link = q;
    }
    L->n++;
    return OK;
}
```

单链表输出操作：

```
int Output(SingleList L) {
    Node *p;
    if(!L.n) return ERROR;
    p = L.first->link;
    while(p) {
        printf("%d ", p->element);
        p = p->link;
    }
    printf("\n");
    return OK;

}
```

单链表删除操作：

```
int Delete(SingleList *L, int i) {
    int j;
    Node *p, *q;
    if(!L->n) {
        return ERROR;
    }
    if(i < 0 || i > L->n - 1) return ERROR;
    q = L->first;
    for(j = 0; j<i;j++){
```

```
        q = q->link;
    }
    p = q->link;
    q->link = p->link;
    free(p);
    L->n--;
    return OK;
}
```

单链表反转操作：

```
int ReverseList(SingleList *L) {
    if(L->first == NULL || L->first->link == NULL) return OK;
    Node *p;
    Node *q;
    Node *r;
    p = L->first->link;
    q = L->first->link->link;
    L->first->link->link = NULL;
    while(q) {
        r = q->link;
        q->link = p;
        p = q;
        q = r;
    }
    L->first->link = p;
    return OK;
}
```

单链表交换两个节点操作：

```
int Swap(SingleList *L, int x, int y) {
    Node *p, *q, *perp, *perq, *sp, *sq;
    p = L->first->link;
    perp = L->first;
    q = L->first->link;
    perq = L->first;
    if(x < -1 || x > L->n - 1) return ERROR;
    if(y < -1 || y > L->n - 1) return ERROR;
    for(int i = 0; i < x; i++) {
        perp = perp->link;
        p = p->link;
    }
    for(int i = 0; i < y; i++) {
        perq = perq->link;
        q = q->link;
    }
    sp = p->link;
```

```
    sq = q->link;
    if(abs(x - y) > 1) {
        perp->link = q;
        q->link = sp;
        perq->link = p;
        p->link = sq;
    }
    if(y - x == 1) {
        perp->link = q;
        q->link = p;
        p->link = sq;
    }
    if(y - x == -1) {
        perq->link = p;
        p->link = q;
        q->link = sp;
    }
    return OK;
}
```

单链表排序操作：

```
int SortList(SingleList *L) {
    int i, j, min, index;
    Node *p, *q;
    for(i = 0; i < L->n-1; i++) {
        min = 999;
        for(j = i; j < L->n; j++) {
            p = L->first->link;
            for(int k = 0; k < j; k++){
                p = p->link;
            }
            if (p->element < min){
                index = j;
                min = p->element;
            }
        }
        Swap(L, i, index);
    }
    return OK;
}
```

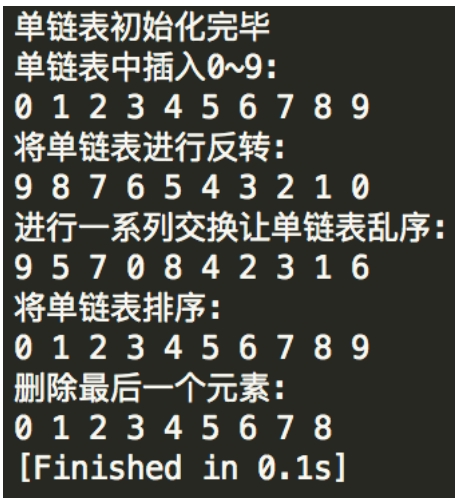
函数调用方式如下：

需要实现的功能	对应的语句
定义一个名称叫做list的单链表	SeqList list
初始化list	Init(&list);
在list的第i个位置插入节点x	Insert(&list, i - 1, x);
删除list的第i个节点	Delete(&list, i);
将list中的元素全部输出	Output(list);
将list进行反转	ReverseList(&list);
交换list中第i位和第j位节点	Swap(&list, i, j);
将list从小到大排序	SortList(&list);

功能测试：

```
int main() {
    SingleList list;
    Init(&list);
    printf("单链表初始化完毕:\n");
    for (int i = 0; i < 10; ++i) {
        Insert(&list, i - 1, i);
    }
    printf("单链表中插入0~9:\n");
    Output(list);
    ReverseList(&list);
    printf("将单链表进行反转:\n");
    Output(list);
    Swap(&list, 6, 7);
    Swap(&list, 3, 9);
    Swap(&list, 1, 4);
    printf("进行一系列交换让单链表乱序:\n");
    Output(list);
    SortList(&list);
    printf("将单链表排序:\n");
    Output(list);
    Delete(&list, 9);
    printf("删除最后一个元素:\n");
    Output(list);
}
```

运行结果：



根据运行结果我们可以发现结果和我们预料的意义，可以认为上述函数实现了我们想要的功能。

实验内容三：

编写程序实现一元多项式的创建、输出、撤销、以及两个一元多项式的相加和相乘操作[2]。

头文件及相关结构定义：

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <cmath>
#define OK 1
#define ERROR 0
#define Underflow 3
#define NotPresent 4
#define Duplicate 5
typedef struct PNode {
    double coef;
    int exp;
    struct PNode *link;
} PNode, *SingleList;
typedef   SingleList   polynomial;
```

输出多项式：

```
void PrintPoly(polynomial L) {
    polynomial p;
    int nCount = 0;
    p = L;
    while(1) {
        if((fabs(p->coef)) > 1e-6) {
            printf("%.2lf", p->coef);
            if(p->exp == 1) {
                printf("X");
            } else if(p->exp != 0) {
                printf("X^%d", p->exp);
            }
            nCount++;
        }

        if(p->link != NULL) {
            if(p->link->coef > 0) {
                printf("+");
            }
            p = p->link;
        } else {
            break;
        }
    }
    if(nCount == 0) {
        printf("0");
    }
    printf("\n");
}
```

将多项式排序：

```
polynomial SortPoly(polynomial &ha) {
    polynomial hb;
    polynomial t, r, s;
    int  exp = -256;
    int nCount = 0;

    t = ha;
    while(t != NULL) {
        if(t->exp > exp) {
            exp = t->exp;
        }
        t = t->link;
    }
```

```
    hb = (polynomial)malloc(sizeof(PNode));
    r = hb;

    while(exp >= -256) {
        t = ha;
        nCount = 0;
        while(t != NULL) {
            if(exp == t->exp) {
                nCount += t->coef;
            }
            t = t->link;
        }
        if(nCount != 0) {
            s = r;
            r ->coef = nCount;
            r ->exp = exp;
            r ->link = (polynomial)malloc(sizeof(PNode));
            r = r->link;
        }
        exp--;
    }
    s ->link = NULL;
    free(r);
    return (hb);
}
```

创建一个多项式：

```
void Creat(SingleList &L) {
    polynomial r, s;
    double m;
    int n = 0;
    char c = '0', flag = 0, array[1000];
    L = (polynomial)malloc(sizeof(PNode));
    printf("coef,exp:");
    scanf("%lf,%d", &L->coef, &L->exp);
    printf("\n");
    r = L;
    while(1) {
        printf("coef,exp:");
        scanf("%lf,%d", &m, &n);
        printf("\n");
        if((fabs(m)) > 1e-6) {
            s = (polynomial)malloc(sizeof(PNode));
            s->coef = m;
            s->exp = n;
            r->link = s;
```

```
            r = r->link;
        } else {
            break;
        }
    }
    r->link = NULL;
}
```

两个多项式相加：

```
polynomial Add(polynomial &ha, polynomial &hb) {
    polynomial p, q, r, hc;

    p = ha;
    q = hb;

    hc = (polynomial)malloc(sizeof(PNode));
    r = hc;

    while(p != NULL) {
        r->coef = p->coef;
        r->exp = p->exp;
        r->link = (polynomial)malloc(sizeof(PNode));
        r = r->link;
        p = p->link;
    }

    while(q != NULL) {
        r->coef = q->coef;
        r->exp = q->exp;
        r->link = (polynomial)malloc(sizeof(PNode));
        r = r->link;
        q = q->link;
    }
    r->link = NULL;
    return (SortPoly(hc));
}
```

两个多项式相乘：

```
polynomial Multiplied1(polynomial p1, polynomial p2) {
    polynomial hd;
    polynomial t, s, r;

    hd = (polynomial)malloc(sizeof(PNode));
    t = hd;
    while(p2 != NULL) {
        s = t;
```

```
            t->coef = p2->coef * p1->coef;
            t->exp = p2->exp + p1->exp;
            t->link = (polynomial)malloc(sizeof(PNode));
            t = t->link;
            p2 = p2->link;
        }
        s->link = NULL;
        free(t);
        return (SortPoly(hd));
}

polynomial Multiplied(polynomial &p1, polynomial &p2) {
        polynomial hd;
        polynomial t, q, s, r;
        hd = (polynomial)malloc(sizeof(PNode));
        hd->coef = 0;
        hd->exp = 0;
        hd->link = NULL;

        while(p1 != NULL) {
            t = Multiplied1(p1, p2);
            hd = Add(t, hd);
            p1 = p1->link;
        }
        return (SortPoly(hd));
}
```

函数调用方法：

需要实现的功能	对应的语句
创建一个多项式a	Creat(a);
输出一个多项式a	PrintPoly(a)
将两个多项式相加	Add(a,b)
将两个多项式相乘	Multiplied(a, b)

功能测试：

```
int main() {
    polynomial ha, hb, hc, hd;
    printf("第一个多项式:\n");
    Creat(ha);
    printf("第一个多项式: ");
    PrintPoly(ha);
    printf("第二个多项式:\n");
    Creat(hb);
    printf("第二个多项式: B=");
    PrintPoly(hb);
    printf("相加所得的多项式:");
    hc = Add(ha, hb);
    PrintPoly(hc);
    printf("相乘所得的多项式: ");
    hd = Multiplied(ha, hb);
    PrintPoly(hd);
    return 0;
}
```

由于sublime对scanf的支持不是很好，修改运行设置，使程序在terminal中运行[3]。

主函数：

```
int main() {
    polynomial ha, hb, hc, hd;
    printf("第一个多项式:\n");
    Creat(ha);
    printf("第一个多项式: ");
    PrintPoly(ha);
    printf("第二个多项式:\n");
    Creat(hb);
    printf("第二个多项式: B=");
    PrintPoly(hb);
    printf("相加所得的多项式:");
    hc = Add(ha, hb);
    PrintPoly(hc);
    printf("相乘所得的多项式: ");
    hd = Multiplied(ha, hb);
    PrintPoly(hd);
    return 0;
}
```

运行结果：

```
第一个多项式：
coef,exp:1,3

coef,exp:2,4

coef,exp:3,5

coef,exp:0,0

第一个多项式： 1.00X^3+2.00X^4+3.00X^5
第二个多项式：
coef,exp:-1,3

coef,exp:2,4

coef,exp:3,6

coef,exp:0,0

第二个多项式： B=-1.00X^3+2.00X^4+3.00X^6
相加所得的多项式:3.00X^6+3.00X^5+4.00X^4
相乘所得的多项式： 9.00X^11+6.00X^10+9.00X^9+1.00X^8-1.00X^6
```

该程序输出与我们预料之中的相符，说明程序能基本实现我们想要的功能。

四、实验小结：

通过本次实验，我自己编写了相关代码，提高了自己的代码能力，也让我认识到了自己的不足之处，对于简单的线性表的掌握并没有我想象之中的那么熟练，第三个多项式的加法和乘法参考了GitHub上面的相关代码。我也希望能够通过一次次的数据结构的实验，改变自己面向api编程的坏毛病，多看源码，多关注和思考别人是怎么去实现这种功能的，而不是仅仅满足学会使用框架。

参考资料

[1]https://zh.wikipedia.org

[2]https://liuyanzhao.com/6279.html

[3]https://github.com/oxlm/GitHub/blob/master/%E5%A4%9A%E9%A1%B9%E5%BC%8F%E7%9B%B8%E5%8A%A0%E5%87%8F%E4%B9%98C%E8%AF%AD%E8%A8%80%E7%A8%8B%E5%BA%8F.cpp

五、指导教师评语

成绩		批阅人		日期	