学生学号	0121510880209	实验课成绩	

武汉理工大学

学 生 实 验 报 告 书

实验课程名称 数据结构

开 课 学 院 计算机科学与技术学院

指导老师姓名 钟忺

学 生 姓 名 彭玉全

学生专业班级 软件 1503

2016 — 2017 学年 第 1 学期

实验课程名称： 数据结构

实验项目名称	线性表算法设计及应用			实验成绩	
实验者	彭玉全	专业班级	软件 1503	组别	
同组者				实验日期	2016-10-13

第一部分：实验分析与设计（可加页）

一、实验内容描述（问题域描述）

实验题 1-1 多项式运算

问题描述：有两个指数递减的一元多项式，写一程序先求这两个多项式的和，再求它们的积。

实验题 1-2 约瑟夫环问题

问题描述：编号为 1，2，・・・，n 的 n 个人围坐在一圆桌旁，每人持有一个正整数的密码。从第一个人开始报数，报到一个预先约定的正整数 m 时，停止报数，报 m 的人退席，下一个人又重新从 1 开始报数，依此重复，直至所有的人都退席。编一程序输出他们退席的编号序列。

二、实验基本原理与设计(算法与程序设计)

1-1 需求分析：用带表头结点的单链表作为多项式的存储表示；要建立两个单链表；多项式相加就是要把一个单链表中的结点插入到另一个单链表中去，要注意插入、删除操作中指针的正确修改。

算法分析和核心代码：

1. 主函数选项卡菜单栏实现一元多项式的多种操作

```
cout<<"*********选项卡*********"<<endl;
cout<<"1 -> 创建一元多项式 "<<endl;
cout<<"2 -> 销毁一元多项式"<<endl;
cout<<"3 -> 打印一元多项式"<<endl;
cout<<"4 -> 一元多项式相加"<<endl;
cout<<"5 -> 一元多项式相减"<<endl;
cout<<"6 -> 一元多项式相乘"<<endl;
cout<<"0 -> 退出程序" <<endl;
cout<<"请选择编号:";
cin>>select;
switch (select)
```

2.创建多项式项的表示结构体

```
typedef struct // 项的表示,
{
    float coef; // 系数
    int expn; // 指数
}term, ElemType;
```

创建节点类型和链表类型

```
typedef struct LNode // 结点类型
{
    ElemType data;
    struct LNode *next;
}LNode,*Link,*Position;
typedef struct LinkList // 链表类型
{
    Link head,tail; // 分别指向线性链表中的头结点和最后一个结点
    int len; // 指示当前线性链表中数据元素的个数
}LinkList;
typedef LinkList polynomial;
```

3. 定义链表操作函数

```
void InitList(LinkList &P);//初始化链表
void CreatePolyn(polynomial &P,int m);//建立表示一元多项式的有序链表 P
void DestroyPolyn(polynomial &P);//销毁一元多项式 P
void PrintPolyn(polynomial P);//打印
int PolyLength(polynomial P);//返回项数
void AddPolyn(polynomial &Pa,polynomial &Pb);//相加运算
void MultiplyPolyn(polynomial &P,polynomial &Pa,polynomial &Pb);//相乘
void SubtractPolyn(polynomial &Pa,polynomial &Pb);//相减运算
```

4. 一元多项式相加算法分析

```
void AddPolyn(polynomial &Pa, polynomial &Pb)//相加运算
{
    LNode *pa = Pa.head->next;//pa指向Pa的当前结点
    LNode *pb = Pb.head->next;//pb指向Pb的当前结点
    while (pa != NULL && pb != NULL)//pa和pb均为非空
    {
        if (pa->data.expn < pb->data.expn)//多项式Pa当前结点的指数值小
        {   pa = pa->next;}
        else if (pa->data.expn == pb->data.expn) //指数相等
        {
            //修改多项式Pa中当前结点的系数值
            pa->data.coef = pa->data.coef + pb->data.coef;
            pa = pa->next; pb = pb->next;
        }
        else if (pa->data.expn > pb->data.expn)//多项式Pb当前结点的指数值小
        {
            LNode *q = Pa.head;
            while (q->next != pa)//找到当前pa结点
            {   q = q->next;}
            q->next = pb;//pb接入当前pa结点的下一个节点
            pb = pb->next;
            Pa.len++;
        }
    }
```

5. 一元多项式相乘算法分析

```
void MultiplyPolyn(polynomial &Pd, polynomial &Pa, polynomial &Pb)//
相乘运算
{
    polynomial Pc;
    InitList(Pd);
    LNode *pa = Pa.head->next;//pa 指向 Pa 的当前结点
    while (pa != NULL)
    {
```

```
        for (int i = 1; i <= Pa ·.len; i++)
        {
            InitList(Pc);
            LNode *pc = Pc.head;//pc 指向 Pc 的头结点
            LNode *pb = Pb.head->next;//pd 指向 Pd 的当前结点
            for (int j = 1; j <= Pb.len; j++)
            {
                LNode *s = (LNode *)malloc(sizeof(LNode));
                if (s == NULL)//内存分配失败报错处理
                  {
                      cout << "出错了 212 程序终止" << endl;
                      exit(-1);
                  }
                  //系数相乘 指数相加
                   s->data.coef = pa->data.coef * pb->data.coef;
                  s->data.expn = pa->data.expn + pb->data.expn;
                  pc->next = s;//将 s 结点的数据链接到 pc
                  s->next = NULL;//next 置空
                  pc = s;
                  pb = pb->next;//pb 下移
                  Pc.len++;//pc 长度增加
              }
        Pc.tail = pb;//Pc 尾指针指向 pb 的当前结点
        AddPolyn(Pd, Pc);
        free(Pc.head);//释放无关内存
        free(Pc.tail);
        pa = pa->next;//一次循环结束 pa 下移
          }
    }
}
```

6. 一元多项式创建函数算法分析

```
void CreatePolyn(polynomial &P, int m)
 {//建立表示一元多项式的有序链表 P
    Position q, s;
    term e;//保存项
    InitList(P);
    cout << "请依次输入" << m << "对系数、指数：" << endl;
    for (int i = 1; i <= m; i++)
    {
        cin >> e.coef >> e.expn;//依次接受输入的系数和指数
        q = P.tail;
        s = (LNode*)malloc(sizeof(LNode));//内存分配与错误处理
        if (s == NULL)
        {
          cout << "出错了 68 程序终止" << endl;
          exit(-1);
        }
```

```
        s->data.coef = e.coef;//将数据转移至链表
        s->data.expn = e.expn;
        P.tail->next = s;//s 结点连接至 P
        s->next = NULL;//置空
        P.tail = s;//链接
        P.len++;//长度增加
    }
    cout<<"新建多项式 Y= ";
    PrintPolyn(P);
}
```

1-2 **需求分析**：用不带表头结点的循环单链表表示围成圆圈的 n 个人；建立此循环单链表；某人离席相当于删除一个结点要正确设置程序中循环终止的条件和删除结点时指针的修改变化。

算法分析：约瑟夫环

```
void josephus_2 (LinkList   r, int m, int n) {
       //r：（不带头结点的）单向循环链表的尾指针
       LinkList   p=r;  //p：指向尾指针
       for (i=1;i<=n; ++i) {
            for (j=1;j<=m-1;++j)   p=p->next;
                // p 后移 m-1 次，定位于 m-1 处（要退席的人之前）
       q=p->next; //q 定位于 m 人（要退席的人）
       printf("%d",q->data); //第 m 人报数
       p->next=q->next;    //第 m 人退席，删除第 m 人
      }
}
```

核心代码：

```
#include<iostream>
using namespace std;
typedef struct LNode {
    int data;
    struct LNode *next;
}LNode, *LinkList;
void josephus(LinkList r, int m, int n)
{
    //r：（不带头结点的）单向循环链表的尾指针
    LinkList q, p = r;  //p：指向尾指针
    for (int i = 1; i <= n; ++i)
    {
      for (int j = 1; j <= m - 1; ++j)
           p = p->next;
      // p 后移 m-1 次，定位于 m-1 处（要退席的人之前）
      q = p->next; //q 定位于 m 人（要退席的人）
      cout << q->data << " ";//第 m 人报数
      p->next = q->next;    //第 m 人退席，删除第 m 人
    }
```

```
}
int main()
{
    int m = 0;
    int n = 0;
    cout << "请输入围成圈的人数 n=";
    cin >> n;
    cout << "请输入退下的人报出的数字 m=";
    cin >> m;
    if (n < 0 || m < 0)
    {
      cout << "数组输入错误，程序退出";
      exit(-1);
    }
    LinkList p, r, cur;//p 为当前节点  r 为 p 的前驱节点
    p = new LNode;
    if (!p)
    {
      cout << "链表创建错误，请检查 39";
      exit(-1);
    }
    p->data = 1;
    p->next = p;//循环链表
    cur = p;
    for (int i = 2; i<=n; i++)
    {
        LinkList t = new LNode;
        if (!t)
        {cout << "链表创建错误，请检查 50" << endl;}
        t->data = i;
        t->next = cur->next;
        cur->next = t;
        cur = t;
    }
    josephus(cur, m, n);
    return 0;
}
```

三、主要仪器设备及耗材

1. 实验设备

   PC 机

2. 开发环境

   VS 2015 Pro

第二部分：实验调试与结果分析（可加页）

一、调试过程（包括调试方法描述、实验数据记录，实验现象记录，实验过程发现的问题等）

1．调试方法描述

完成代码输入后，运行程序，出现内存溢出 bug。使用断点依照主函数代码运行流程逐代码块，逐句调试。直至解决所有 bug，程序运行通过。

2．实验输入/输出数据记录

1-1 开始程序

```
*********选项卡*********
1 -> 创建一元多项式
2 -> 销毁一元多项式
3 -> 打印一元多项式
4 -> 一元多项式相加
5 -> 一元多项式相减
6 -> 一元多项式相乘
0 -> 退出程序
请选择编号:
```

输入 1 开始创建

```
请选择编号:1
请输入第一个一元多项式的项数:3
请依次输入3对系数、指数:
3 3
2 2
1 1
新建多项式Y= 3X^3+2X^2+1X^1
```

输入 4 开始多项式相加

```
请选择编号:4
请输入第一个一元多项式的项数:3
请依次输入3对系数、指数:
3 3
2 2
1 1
新建多项式Y= 3X^3+2X^2+1X^1
请输入第二个一元多项式的项数:3
请依次输入3对系数、指数:
4 4
3 3
2 2
新建多项式Y= 4X^4+3X^3+2X^2
多项式 Y1= 3X^3+2X^2+1X^1
多项式 Y2= 4X^4+3X^3+2X^2
相加后 Y= 3X^3+2X^2+1X^1+4X^4+3X^3+2X^2
```

输入 6 开始多项式相乘

```
请选择编号:6
请输入第一个一元多项式的项数:2
请依次输入2对系数、指数:
2 2
3 3
新建多项式Y= 2X^2+3X^3
请输入第二个一元多项式的项数:2
请依次输入2对系数、指数:
2 2
3 3
新建多项式Y= 2X^2+3X^3
多项式 Y1= 2X^2+3X^3
多项式 Y2= 2X^2+3X^3
相乘后 Y= 4X^4+12X^5+9X^6
```

1-2 程序测试

```
D:\编程练习2016\shiyan1_2.exe
请输入围成圈的人数n=7
请输入退下的人报出的数字m=20
6 1 7 5 3 2 4
--------------------------------
Process exited after 6.923 seconds with return value 0
请按任意键继续. . .
```

二、实验结果

经过算法设计，代码编辑，测试，断点调试，程序最终正常通过测试。实验题 1-1 中，新增一元多项式的减法，具体算法和代码参考一元多项式加法，代码实现较为简单；主函数中创建选项卡便于一元多项式的具体操作，可根据不同选项实现不同的功能；选项 3 中增加销毁一元多项式的功能，通过释放结点内存来实现手动删除多项式释放内存；目标功能全部实现。

实验题 1-2 中，约瑟夫环的算法分析参考数据结构 ppt，具体实现使用了循环链表，在主函数中完成链表创建，数据输入，调用约瑟夫环函数，输出结果。

三、实验小结、建议及体会

经过此次试验，我熟悉了链表的结构特点，熟悉了链表的创建，销毁，插入等相关操作，加深了自己对链表知识的熟悉程度。不管是单链表，带头节点的单链表，以及循环链表，在创建使用的过程中都要注意内存泄露的问题，指针的正确使用在程序中也很重要。在结构体创建时要注意区分，结点类型和链表类型，注意结点和指针在使用方面的不同。

实验课程名称： 数据结构

实验项目名称	栈和队列的算法设计及应用			实验成绩	
实验者	彭玉全	专业班级	软件 1503	组别	
同组者				实验日期	2016-10-13

第一部分：实验分析与设计（可加页）

三、实验内容描述（问题域描述）

实验题 2-1 算术运算器设计

问题描述：设计一个模拟计算器功能的程序，它读入一个表达式，如果是一个正确的表达式（即它由操作数、圆括号和+、-、*、/四种运算符组成），则求出该表达式的值；否则给出某种错误信息。

四、实验基本原理与设计(算法与程序设计)

2-1 需求分析：读入一个以字符序列形式给出的以等号（=）结尾的表达式；程序中应考虑运算符的优先级、运算的合法性。

算法分析：

为实现运算符优先算法，使用两个工作栈，一个 SqStackR 用于寄存运算符，一个 SqStackD 用于寄存操作数和结果。

首先置操作数栈为空栈，表达式起始符为“#”为运算符栈的栈底元素。

依次读入表达式中每个字符，若是操作数则进 SqStackD 栈，若是运算符和 SqStackR 栈的栈顶元素比较优先级后做出相应的操作，直至整个表达式求值完毕。代码实现如下：

```
else
    switch (OpPriority(GetTop(S), c[i]))
    {
    case '<'://栈顶元素优先级低
        Push(S, c[i]);
        i++;
        break;
    case '='://脱掉括号 并接受下一个字符
        Pop(S);
        i++;
        break;
    case '>'://退栈并保存结果
        r = Pop(S);
        a = Pop(D);
        b = Pop(D);
        Push(D, Operate(a, r, b));
        break;
    }
```

具体算法实现：创建两个栈 各自实现创建，初始化，判空，销毁，进栈，出栈，获取栈顶元素的操作，代码参考源码。char OpPriority(char top, char c)函数用于比较当前运算符与栈顶运算符的优先级，返回标志符号。float Expression(char c[], SqStackR &S, SqStackD &D)用于解析表达式，并计算结果。主函数通过数组来接收保存表达式，调用 Expression()函数开始解析表达式。

核心代码：

```
#include<iostream>
using namespace std;
#define MAX 16
#define AGAIN 10
typedef struct//运算符
{
char *base;
char *top;
int stacksize;
}SqStackR;
typedef struct {//数据
float *base;
float *top;
int stacksize;
}SqStackD;
void InitStack(SqStackR &S) //运算符堆栈的初始化
{
S.base = (char*)malloc(MAX * sizeof(char));
if (!S.base)
{    cout << "运算符队列创建失败" << endl;exit(0);     }
S.top = S.base;
S.stacksize = MAX;
}

void InitStack(SqStackD &S) //数据堆栈的初始化
{
S.base = (float *)malloc(MAX * sizeof(float));
if (!S.base)
{   cout << "数据队列创建失败" << endl;exit(0);    }
S.top = S.base;
S.stacksize = MAX;
}

int StackEmpty(SqStackR &S) //判断是否空栈
{
if (S.top == S.base)
     return 1;
else
```

```
    return 0;
}

int StackEmpty(SqStackD &S) //判断是否空栈
{
if (S.top == S.base)      return 1;
else    return 0;
}

void DestroyStack(SqStackR &S) //销毁运算符队列
{
S.top = S.base;
S.stacksize = 0;
}

void DestroyStack(SqStackD *S) //销毁数据队列
{
S->top = S->base;
S->stacksize = 0;
}

void Push(SqStackR &S, char e) //运算符入栈
{
int Top = (int)S.top, Base = (int)S.base;
if ((Top - Base) == S.stacksize)//判断栈满否
{
    S.base = (char *)realloc(S.base, (S.stacksize +AGAIN)*sizeof(char));
    if (!S.base)
    {    cout << "运算符队列新增初始化失败" << endl;exit(0);}
    S.top = S.base + S.stacksize;//新的站定
    S.stacksize = S.stacksize + AGAIN;//新的站的长度
}
*(S.top) = e;
S.top++;
}

void Push(SqStackD &S, float e) //数据的入栈
{
int Top = (int)S.top, Base = (int)S.base;
```

```
if ((Top - Base) == S.stacksize)
{
    S.base = (float *)realloc(S.base, (S.stacksize +AGAIN)*sizeof(float));
    if (!S.base)
    {   cout << "数据队列新增初始化失败" << endl;    exit(0);}
    S.top = S.base + S.stacksize;
    S.stacksize = S.stacksize + AGAIN;
}
*(S.top) = e;
S.top++;
}

char Pop(SqStackR &S) //出栈返回栈顶元素后删除栈顶元素
{
char e;
if (StackEmpty(S))
{   cout << "运算符栈是空的 111" << endl;    exit(0); }
S.top--;
e = *(S.top);
return e;
}

float Pop(SqStackD &S) //出栈返回栈顶元素后删除栈顶元素
{
float e;
if (StackEmpty(S))
{   cout << "数据栈是空的 124" << endl; exit(0);}
S.top--;
e = *(S.top);
return e;
}

char GetTop(SqStackR &S)//取运算符栈顶元素
{
char e;
if (StackEmpty(S))
{       cout << "运算符栈是空的 137" << endl;        exit(0); }
e = *(S.top - 1);
return e;
```

```
}

float GetTop(SqStackD &S)
{//取数据栈顶元素
float e;
if (StackEmpty(S))
{    cout << "数据栈是空的 148" << endl; exit(0);}
e = *(S.top - 1);
return e;
}

int OpJudge(char c)  //判断是否为运算符
{
char ch[7] = { '+','-','*','/','(',')','#' };
for (int i = 0; i < 7; i++)
{
    if (c == ch[i])
        return 1;
}
return 0;
}

char OpPriority(char top, char c)
{
char ch;
if (top == '(' && c == ')' || top == '#' && c == '#')
    ch = '=';
else if (top == '+' || top == '-')      //栈顶元素为‘+’或‘-’的时候
        switch (c)
        {
        case '+':
        case '-':
        case ')':
        case '#': ch = '>'; break;
        case '*':
        case '/':
        case '(': ch = '<';
        }
else if (top == '*' || top == '/')     //栈顶元素为‘*’或‘/’的时候
```

```
        switch (c)
            {
            case '+':
            case '-':
            case '*':
            case '/':
            case ')':
            case '#': ch = '>'; break;
            case '(': ch = '<';
            }
    else if (top == '(')                //栈顶元素为‘(’的时候
        switch (c)
            {
            case '+':
            case '-':
            case '*':
            case '/':
            case '(': ch = '<'; break;
            case '#':    cout<<"表达式解析错误 201"<<endl;
                exit(0);
            }
    else if (top == ')')      /*栈顶元素为‘)’的时候*/
        switch (c)
            {
            case '+':
            case '-':
            case '*':
            case '/':
            case '#': ch = '>'; break;
            case '(': cout << "表达式解析错误 212" << endl;
                exit(0);
            }
    else if (top == '#')                /*栈顶元素为‘#’的时候*/
        switch (c)
            {
            case '+':
            case '-':
            case '*':
            case '/':
```

```
            case '(': ch = '<'; break;
            case ')':    cout << "表达式解析错误 224" << endl;
                exit(0);
            }
return ch;
}

float Operate(float a, char r, float b)
{
float s, d1 = a, d2 = b;
switch (r)
{
    case '+':s = d1 + d2; break;
    case '-':s = d2 - d1; break;
    case '*':s = d1 * d2; break;
    case '/':
        if (d1 == 0)
            { s = 0; break; }
        s = d2 / d1;
        break;
}
return s;
}

float Expression(char c[], SqStackR &S, SqStackD &D) {
float a, b, s = 0, e = 1.0;
int i = 0;
char r;
while (c[i] != '#' || GetTop(S) != '#')
{
    if (!OpJudge(c[i]))
    {
        if (c[i] >= '0' && c[i] <= '9')
        {
            s += c[i] - '0';
            while (!OpJudge(c[++i])) //对小数位的判断
            {
                if (c[i] != '.')
                {
```

```
                    s *= 10;
                    s += c[i] - '0';
                }
                else
                {
                    while (!OpJudge(c[++i]))
                    {
                        e *= 0.1;
                        s += (c[i] - '0') * e;
                    }
                    break;
                }
            }
            Push(D, s);
            s = 0;
            e = 1.0;
        }
        else
        {
            cout << "表达式解析错误 282" << endl;
            return 0;
        }
    }
    else
        switch (OpPriority(GetTop(S), c[i]))
        {
        case '<':
            Push(S, c[i]);
            i++;
            break;
        case '=':
            Pop(S);
            i++;
            break;
        case '>':
            r = Pop(S);
            a = Pop(D);
            b = Pop(D);
            Push(D, Operate(a, r, b));
```

```
                break;
            }
    }
    return (GetTop(D));
    }


    int main()
    {
    char c[50], x;
    SqStackR S;
    SqStackD D;
    InitStack(S);
    Push(S, '#');
    InitStack(D);
    float r = 0;//结果
    for (int i = 0; i < 50; i++)
    {    c[i] = '#';}
    cout<<"输入一个标准的表达式=：";
    for (int i = 0; i < 50; i++)
    {
        if ((x = getchar()) != 10)
            c[i] = x;
        else
            break;
    }
    r = Expression(c, S, D);
    cout << " = "<<r<<endl;
    return 0;
```

三、主要仪器设备及耗材

1. 实验设备

PC 机

2. 开发环境

VS 2015 Pro

第二部分：实验调试与结果分析（可加页）

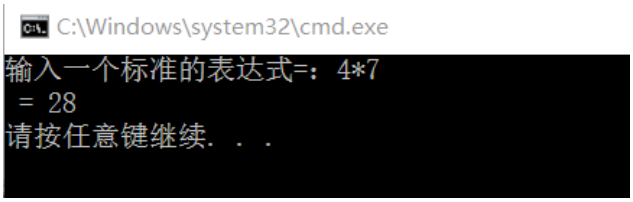
四、调试过程（包括调试方法描述、实验数据记录，实验现象记录，实验过程发现的问题等）

1．调试方法描述

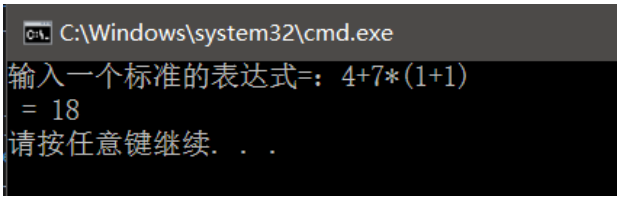
完成代码输入后，运行程序，出现内存溢出 bug。使用断点依照主函数代码运行流程逐代码块，逐句调试。直至解决所有 bug，程序运行通过。

2．实验输入/输出数据记录

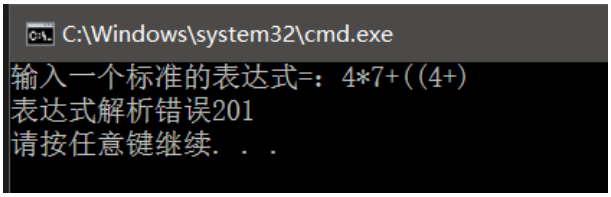
按照提示输入一个表达式 1：



按照提示输入一个表达式 2：



错误处理机制：



五、实验结果

经过算法设计，代码编辑，测试，断点调试，程序最终正常通过测试

程序实现基本的算法功能，能够处理标准的表达式的求解，按照标准格式输入的表达式能够正确给出结果，但此程序目前容错功能较差，还不能据出错情况给出详细的出错点解释。

三、实验小结建议和体会

经过此次试验，我熟悉了队列和堆栈的结构特点，熟悉了队列和堆栈的创建，销毁，插入等相关操作，加深了自己对队列和堆栈知识的熟悉程度。再次强调了指针的正确使用在程序中很重要。