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	串的匹配算法设计及实现			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- 27

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 3 ：串的匹配算法设计及实现

问题描述：对任意输入的一串字符，在某文档中进行匹配，并给出匹配结果。

测试数据：（1）输入的一行程序，与源代码匹配，源程序自行选择；

（2）输入的一串字符，在某文本文件中匹配，文本文件自行选择。

二、实验基本原理与设计(算法与程序设计)

需求分析：

需定义文件操作，打开和关闭文件，读取到串中进行匹配。匹配算法的具体实现，参考课本 79 页内容。

算法分析

算法 1：

分别利用计数指针 i 和 j 指示主串 S 和模式串 T 中当前正在比较的字符位置。从主串 S 的第 pos 个字符起和模式的第一个字符比较之，若相等，则继续比较后续字符；否则从主串的下一个字符起再重新和模式的字符比较。以此类推，直至模式 T 中的每个字符一次和主串的一个连续的字符序列相等，则匹配成功，函数值为和模式 T 中第一个字符相等的字符在主串 S 中的序号否则匹配不成功，函数值为 0.

```
int Index(SString S,SString T,int pos)
{
 //返回子串 T 在主串 S 中第 pos 个字符之后的位置
 //其中，T 非空 1<=pos<=Length(S)
 i = pos;  j = 1;
 while(i<=Length(S) && j<=Length(T))
 {
     if(S[i] == T[j])
     {
         ++i;++j;
     }
     else
     {
         i = i-j+2; j = 1;
     }
     if(j>Length(T)) return i-Length(T);
     else return 0;
 }
```

```
}
```

算法 2：

对算法 1 的改进，每当一趟匹配过程中出现字符比较不等时不需要回溯 i 指针，而是利用已经得到的“部分匹配”的结果将模式串向右滑动尽可能远的距离后，继续比较。若令 next[j] = k,则 next[j]表明当模式中的第 j 个字符与主串中相应的字符“失配”时，在模式中重新和主串中该字符进行比较的字符的位置。在求出模式的 next 函数后，假设子指针 i 和 j 分别指示主串和模式中比较的字符，令 i 的储值为 pos，j 的初始值为 1.若在匹配过程中 Si == Pj，则 i 和 j 分别自增 1，否则，i 不变，而 j 退回到 next[j]的位置再比较，若想等，则指针各自增 1，否则 j 退回到下一个 next 的位置，以此类推，直至下列两种可能：一种是 j 退到某个 next 的值时字符比较相等，则指针各自增 1，继续当前匹配；另一种时 j 退回到 0，则此时需要将模式继续向右滑动一个位置，即从主串的下一个字符和模式重新开始匹配。

```
int Index_KMP(SString S,SString T,int pos)
{
 //返回子串 T 在主串 S 中第 pos 个字符之后的位置
 //KMP 算法  其中，T 非空 1<=pos<=Length(S)
 i = pos;  j = 1;
 while(i<=Length(S) && j<=Length(T))
 {
     if(j == 0||S[i] == T[j])
     {
         ++i;++j;
     }
     else
         j = next[j];
     if(j>Length(T)) return i-Length(T);
     else return 0;
 }
 }
void get_next(SString T, int next[])
{
 //求模式串的 next 函数值并存入数组 next
 i = 1;  next[1]= 0;  j = 0;
 while(i < Length(T))
 {
     if(j == 0||T[i] == T[j])
     {
         ++i; ++j; next[i] = j;
     }
     else j = next[j];
  }
}
```

核心代码：

算法 1：

```
#include<iostream>
#include<string>
#include<fstream>
using namespace std;
#define PATH "D:/string1.txt"
#define MAXLEN 10240
```

```
typedef char SString[MAXLEN + 1];
int Length(SString S)
{
return strlen(S);
}
//第一种算法 暴力匹配
int Index(SString S, SString T, int pos)
{
    int m = 0;
    int n = 0;
    int i = pos;
    int j = 0;
    while (i <= Length(S) && j <= Length(T))
    {
        if (S[i] == T[j])
        {
            ++i;
            ++j;
            if (j == Length(T))//保存相关量
            {
                m = 1;
                n = i;
            }
        }
        else
        {
            i = i - j + 1; j = 0;
        }
    }
    if (m == 1)
    return n - Length(T)+1;
    else
    return 0;
}
void StrAssign(SString S, char* s)// 生成一个其值等于s的串T
{
    int i;
    for (i = 0; i<strlen(s); ++i)
    {
        S[i] = s[i];
    }
    S[i] = '\0';
}
int main()
{
```

```
    SString S, T;//主串和模式串
    char s[MAXLEN]="\0", t[100];
    int next[100] = {0};
    char ch;
    int i = 0;
    ifstream input(PATH);
    if (!input)
    {
        cout << "文件打开失败" << endl;
        exit(1);
    }
    while (!input.eof())
    {
        input.get(ch);
        s[i] = ch;
        i++;
    }
    input.close();
    StrAssign(S, s);//主串
    cout << "请输入你要匹配的模式串：";
    cin.get(t,100);
    StrAssign(T, t);
    if (Index(S, T, 0) == 0)
    cout << "在主串中未匹配到此字符串" << endl;
    else
    cout << "在第"<<Index(S, T, 0) <<"位"<< endl;

    return 0;
}
    return 0;
```

算法 2：

```
#include<string>
#include<fstream>
using namespace std;
#define PATH "D:/string1.txt"
#define MAXLEN 10240
typedef char SString[MAXLEN + 1];
int Length(SString S)
{
    return strlen(S);
}
//第二种算法  next
int Index_KMP(SString S, SString T, int pos,int next[])
{
    int i = pos;
```

```
    int j = 1;
    while (i <= Length(S) && j <= Length(T))
    {
        if (j == 0 || S[i] == T[j])
        {
            ++i; ++j;
        }
        else
            j = next[j];
        if (j == Length(T))
            break;
    }
    if (j >= Length(T))
        return i - Length(T);
    else
        return 0;
}
void get_next(SString S,int next[])
{
    int i = 1;
    next[1] = 0;
    int j = 0;
    while (i < Length(S))
    {
        if (j == 0 || S[i] == S[j])
        {
            ++i; ++j; next[i] = j;
        }
        else
            j = next[j];
    }
}
void StrAssign(SString S, char* s)// 生成一个其值等于s的串T
{
    int i = 0;
    S[0] = '0';
    for (i = 0; i<=strlen(s); ++i)
    {
        S[i+1] = s[i];
    }
    S[i] = '\0';
}
int main()
{
    SString S, T;//主串和模式串
```

```
    char s[MAXLEN]="\0", t[100];
    int next[100] = {0};
    char ch;
    int i = 0;
    ifstream input(PATH);
    if (!input)
    {
        cout << "文件打开失败" << endl;
        exit(1);
    }
    while (!input.eof())
    {
        input.get(ch);
        s[i] = ch;
        i++;
    }
    input.close();
    StrAssign(S, s);//主串
    cout << "请输入你要匹配的模式串：";
    cin.get(t,100);
    StrAssign(T, t);
    get_next(T, next);
    if (Index_KMP(S, T, 1, next) == 0)
        cout << "在主串中未匹配到此字符串" << endl;
    else
        cout << "在第" << Index_KMP(S, T, 1, next)+1 << "位" << 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）

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

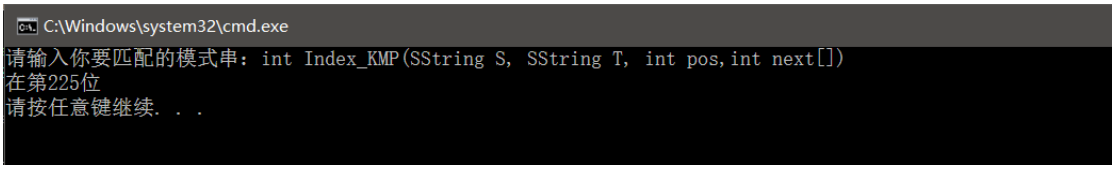
Strng1.txt 文件内容截图

string1.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
#include<iostream>
#include<string>
#include<fstream>
using namespace std;
#define PATH "D:/string1.txt"
#define MAXLEN 10240
typedef char SString[MAXLEN + 1];
int Length(SString S)
{
        return strlen(S);
}
//第二种算法  next
int Index_KMP(SString S, SString T, int pos,int next[])
```

匹配成功：



匹配失败：

3. 实验过程中遇到的问题

在算法 1 和 2 中同时遇到了算法匹配越界的问题，在本应匹配结束时，程序又继续进行匹配，改进算法如下，

```
int Index_KMP(SString S, SString T, int pos,int next[])
{
    int i = pos;
    int j = 1;
    while (i <= Length(S) && j <= Length(T))
    {
        if (j == 0 || S[i] == T[j])
        {++i; ++j;}
        else
            j = next[j];
        if (j == Length(T))//比较全部符合匹配则退出循环
            break;
```

```
    }
    if (j >= Length(T))
        return i - Length(T);
    else
        return 0;
}
```

二、实验结果

经过需求分析，算法分析，代码编写，测试，调试，程序最终实现所需要的功能，及在一个文件中匹配模式串并输出匹配结果。

三、实验小结、建议及体会

经过此次实验，加深了我对串的认识和了解，对串的基本操作有了详细的认识，能够自由使用串这一数据存储样式。能够在串中查找某个子串，求一个子串，在串的某个位置差入一个子串以及删除一个子串。了解了关于串的定长顺序存储表示，用一组地址连续的存储单元存储串值得字符序列。在串的定长顺序存储结构中，按照预定义的大小，为每个定义的串变量分配一个固定长度的存储区。熟悉运用串的模式匹配算法求子串位置，以及改进算法 KMP 算法的使用，next 数组值 的求解。

实验课程名称： 数据结构

实验项目名称	二维数组连乘算法设计及实现			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	2016-10-27

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 4 二维数组连乘问题编制哈夫曼编码

问题描述： 写一个程序完成多个二维数组的 连乘，并要求找出运算量最小的乘积顺序。

测试数据： $A_{10*30}*B_{30*70}*C_{70*1}*D_{1*200}$

提示：A10*30*B30*70*C70*1*D1*200 共有五中完全加括号的方式:

(A((BC)D)),(A(B(CD))4),((AB)(CD)),(((AB)C)D),((A(BC))D)

其运算量分别为：

24500，494000，175000，23700，4400

所以，运算量最小的乘积顺序是：((A(BC))D)

推广到一般情形：

给定 n 个矩阵{A1,A2,…,An}，其中 Ai 与 Ai+1 是可乘的，i=1，2…，n-1。如何确定计算矩阵连乘积的计算次序，使得依此次序计算矩阵连乘积需要的数乘次数最少。

计算步骤：

（1）m11=m22=m33=m44=0

（2）m12=m11*m22=10*30*70=21000

m23=m22*m33=30*70*1=2100

m34=m33*m44=70*1*200=14000

（3）m13=min{m12*m33, m11*m23}

=min{21000+0+10*70*1,0+2100+10*30*1}=2400

m24=min{m23*m44, m22*m34}

=min{2100+0+30*1*200, 0+14000+30*70*200}=8100

（4）m14=min{m11*m24, m12*m34 ,m13*m44}

=min{0+8100+10*30*200,21000+14000+10*70*200, 2400+0+10*1*200}

=4400

其中，mij 表示{Ai…Aj}连乘。

二、实验基本原理与设计(算法与程序设计)

算法分析：

动态规划法：将矩阵连乘积 AA…A 简记为 A[i:j],这里 i<=j，考察计算 A[i:j]的最优计算次序。设计这个计算次序在矩阵 A 和 A 之间将矩阵链断开，i<=k<j,则其相应完全加括号方式为$(A_iA_{i+1}\cdots A_k)(A_{k+1}A_{k+2}\cdots A_j)$。计算量：A[i:k]的计算量加上 A[k+1:j]的就量，再加上 A[i:k]和 A[k+1:j]相乘的计算量。

计算 A[i:j]的最优次序所包含的计算矩阵子链 A[i:k]和 A[k+1:j]的次序也是最优的

建立递归关系，设计 A[i:j]，1<=i<j<n,所需要的最少数乘积

次数 m[i:j]，则原问题的最优值为 m[1,n]。当 i=j 时，A[i:j] = A_i，当 i<j 时，m[i,j] = m[I,k] + m[k+1,j] + p_kp_j.

用动态规划算法解决此问题，可依据其递归式以自底向上的方式进行计算。在计算过程中，保存已解决的子问题答案。每个子问题只计算一次，而在后面需要时只简单查一下，从而避免大量的重复计算。

核心代码：

```
int MatrixChain()
{
    for (int i = 0; i <= n; i++)
        m[i][i] = 0;
    for (int r = 2; r <= n; r++)// 对角线循环
        for (int i = 0; i <= n - r; i++)// 行循环
        {
            int j = r + i - 1;// 列的控制
                              //找 m[i][j] 的最小值，先初始化一下，令 k=i
            m[i][j] = m[i + 1][j] + p[i + 1] * p[i] * p[j + 1];
            s[i][j] = i;
            //k 从 i+1 到 j-1 循环找 m[i][j] 的最小值
            for (int k = i + 1; k<j; k++)
            {
                int temp = m[i][k] + m[k + 1][j] + p[i] * p[k + 1] * p[j + 1];
                if (temp<m[i][j])
                {
                    m[i][j] = temp;
                  //s[][] 用来记录在子序列 i-j 段中，在 k位置处 //断开能得到最优解
                    s[i][j] = k;
                }
            }
        }
    return m[0][n - 1];
   }  //根据 s[][] 记录的各个子段的最优解，将其输出
```

源代码：

```
#include<iostream>
using namespace std;
```

```
const int MAX = 100;
int l[MAX + 1];//存放矩阵的行列值
int m[MAX][MAX];//用来记录第i至j个矩阵的最优解
int t[MAX][MAX];//用来记录最优解的断点
int n;//矩阵个数
int MaxtrixChain()
{
    for (int i = 0; i <= n; i++)
        m[i][i] = 0;//初始化矩阵
    for (int k = 2; k <= n; k++)
    {
        for (int i = 0; i <= n - k; i++)
        {
            int j = k + i - 1;
            m[i][j] = m[i + 1][j] + l[i + 1] * l[i] * l[j + 1];
            t[i][j] = i;
            for (int q = i+1; q < j; q++)
            {
                int temp = m[i][q] + m[q + 1][j] + l[i] * l[q + 1] * l[j + 1];
                if (temp < m[i][j])
                {
                    m[i][j] = temp;
                    t[i][j] = q;
                }
            }
        }
    }
    return m[0][n-1];
}
void print(int i, int j)
{
    if (i == j)
    {
        cout << (char)('A' + i);
        return;
    }

    if (i < t[i][j])
        cout << '(';
    print(i, t[i][j]);

    if (i < t[i][j])
        cout << ')';
    if (t[i][j] + 1<j)
        cout << '(';
```

```
    print(t[i][j] + 1, j);

    if (t[i][j] + 1<j)
        cout << ')';
}
void print()
{
    cout << '(';
    print(0, n - 1);
    cout << ')';
    cout << endl;
}
int main()
{
    cout << "请输入矩阵的个数： ";
    cin >> n;
    cout << "输入矩阵（A10*30*B30*70*C70*1*D1*200 -> 10 30 70 1 200） :" << endl;
    for (int i = 0; i <= n; i++)
        cin >> l[i];
    cout << "输出结果： " ;
    MaxtrixChain();
    print(0, n - 1); //最终解值为 m[0][n-1];
    cout << 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）

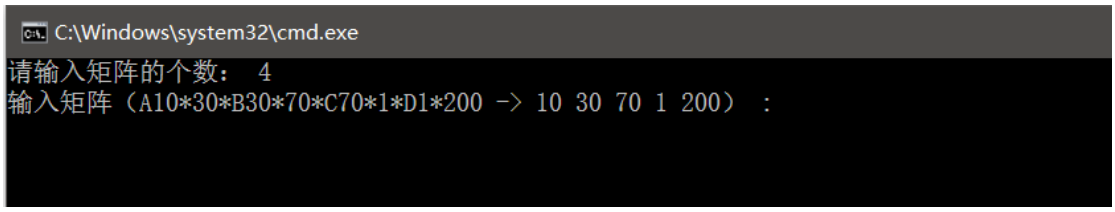
三、调试过程（包括调试方法描述、实验数据记录，实验现象记录，实验过程发现的问题等）

1. 调试方法描述

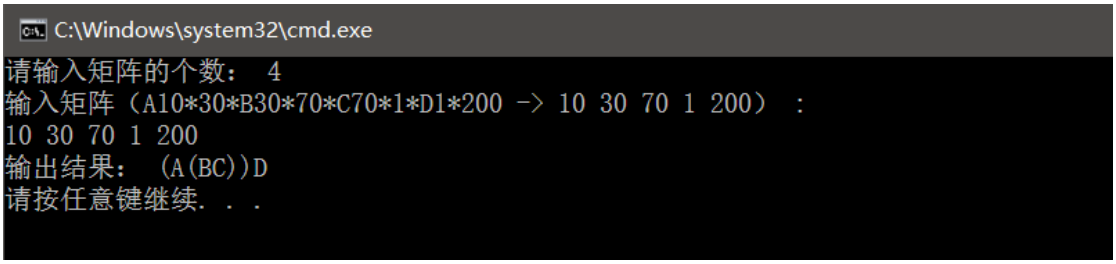
完成代码输入后，运行程序，出现内存溢出 bug。使用断点依照主函数代码运行流程逐代码块，逐句调试。直至解决所有 bug，程序运行通过。

2. 实验输入/输出数据记录

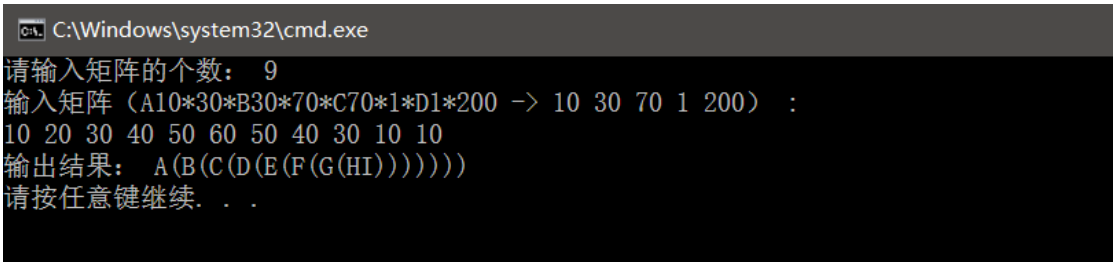
程序开始 输入矩阵个数



输入矩阵的 i 和 j 仿照例子输入



测试输出数据



3. 实验中遇到的问题及解决办法

算法分析部分一开始并不知道采用此中算法比较简单，第一次尝试采用穷举法过于复杂，在 CSDN 上发现了关于此类方法的介绍，所以参考了一下。载输出递归调用时要理清思绪不然很容易找不到头绪。

四、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

程序实现算法设计的基本功能，能够对多个数组连乘作出处理，输出最优运算次序。

五、实验小结建议和体会

经过此次试验，我熟悉了二维数组的具体操作，同时加深了自己对广义表的认识和理解。加深了自己对课本知识的理解和掌握。特别在动态规划算法上有了更为清醒的认识。