**《算法与数据结构》**

**实 验 指 导 手 册**

# 北京邮电大学软件学院

# 2020-2021学年第1学期实验报告

**课程名称： 算法与数据结构**

1. **实验名称： 实验三 字符串的模式匹配**

**实验完成人：**

**姓名：**\_\_\_王宇涵\_\_\_\_\_**学号：**\_\_\_\_2020211730\_\_\_\_**成绩：**\_\_\_\_\_\_\_\_

**指导教师：**\_\_\_\_\_\_\_**\_\_贾红娓** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**日 期： 2021 年 10 月 8 日**

1. **实验目的**

本次实验的目的是熟悉串类型的实现方法和文本模式匹配方法，熟悉串的键盘输入获取方式。

1. **实验内容**

**串的模式匹配问题**

**[问题描述]**

设有两个字符串s和t，首先将s1与t1进行比较，直到s的某一个字符si和ti相同，

再将它们之后的字符进行比较，若也相同，则如此继续往下比较，当s的某一个字符si与t的字符tj不同时，则s返回到本趟开始字符的下一个字符，即si-j+2，t返回到t1，继续开始下一趟的比较，重复上述过程。若t中的字符全部比较完，则说明本趟匹配成功，本趟的起始位置是i-j+1，否则，匹配失败。

**[基本要求]**

　　本实验要求学生掌握串的特点及顺序定长存储的方式，掌握模式匹配的基本思想及其算

法。由用户通过键盘输入建立一个主字符串和搜索串，如果主串中包含要搜索的子串，返回子串在主串中的起始位置，否则返回搜索失败。

**[测试数据]**

由学生依据软件工程的测试技术自己确定。注意测试边界数据。

1. **实验环境**

VC6.0

1. **实验要求**
   1. 编程实现必做实验内容。
   2. 填写实验报告。
2. **实验过程和实验结果**

**1.串的模式匹配问题**

**1.1问题分析**

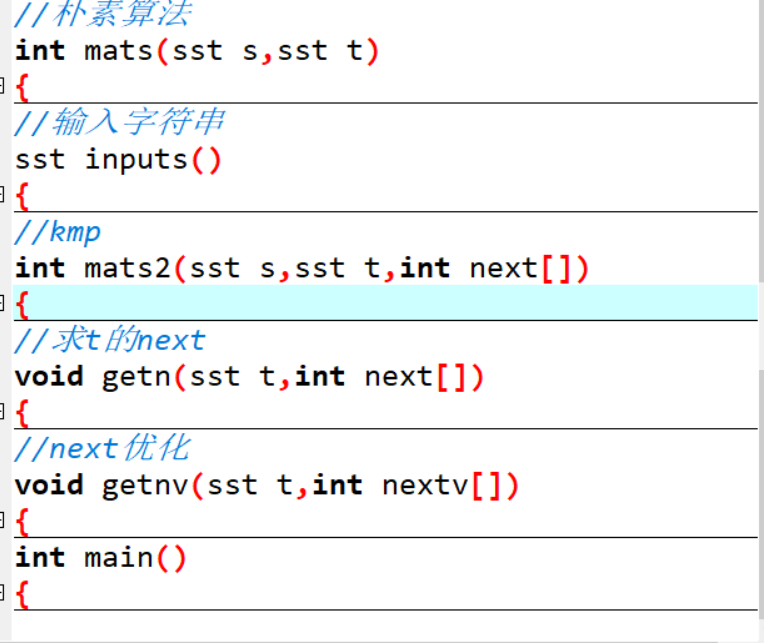
在主串S中，开始查找是否存在子串(也称做模式串)T，如果在主串S中查找到一个与模式串T相同的子串，则查找成功；如在主串S中未找到一个与模式串T相同的子串，则称查找失败。当模式匹配成功时,函数返回模式串T的第一个字符在主串S中的位置。

朴素模式：从主串的指定的起始位置字符开始和模式第一个字符比较，如果相等，则继续比较下一个字符，如果不等，则从主串的下一个字符开始和模式的第一个字符开始比较，以此类推，直到模式串所有字符都匹配完成，则匹配成功，否则，匹配不成功。

KMP：建立next数组，为朴素算法优化。利用已经部分匹配这个有效信息，保持i指针不回溯，通过修改j指针，让模式串尽量地移动到有效的位置

**1.2设计方案**

1. 用结构体先定义串，包括数据和长度
2. 输入字符串函数
3. 朴素算法函数
4. 求next的函数
5. Next优化函数
6. KMP函数
7. 主函数

****

**1.3算法**

1. **输入**

**//输入字符串**

**sst inputs()**

**{**

**//如果用cin输入，字符串会从0开始**

**sst s1;**

**char c;**

**int i=0;**

**while(1)**

**{**

**scanf("%c",&c);**

**if(c==' '||c=='\n')**

**{**

**break;**

**}**

**i++;**

**s1.ch[i]=c;**

**//cout<<i<<endl;**

**//cout<<s1.ch[i]<<endl;**

**}**

**s1.length=i;**

**//s1.ch[i+1]='\0';**

**return s1;**

**}**

1. **朴素**

**//朴素算法**

**int mats(sst s,sst t)**

**{**

**//int k=1;**

**int i=1,j=1;**

**while(i<=s.length&&j<=t.length)**

**{**

**if(s.ch[i]==t.ch[j])**

**{**

**//cout<<s.ch[i]<<t.ch[i]<<endl;**

**i++;**

**j++;//继续比较**

**}**

**else**

**{**

**/\*k++;**

**i=k;**

**j=1;**

**or\*/**

**i=i-j+2;//检查下一个子串**

**j=1;**

**}**

**//cout<<i<<" "<<j<<endl;**

**}**

**if(j>t.length)**

**{**

**//cout<<i-t.length<<endl;**

**return i-t.length;**

**}**

**else**

**{**

**return 0;**

**}**

**}**

1. **Kmp**

**//kmp**

**int mats2(sst s,sst t,int next[])**

**{**

**//int k=1;**

**int i=1,j=1;**

**while(i<=s.length&&j<=t.length)**

**{**

**if(s.ch[i]==t.ch[j]||j==0)**

**{**

**//cout<<s.ch[i]<<t.ch[i]<<endl;**

**i++;**

**j++;//继续比较**

**}**

**else**

**{**

**j=next[j];//模式串向右**

**}**

**//cout<<i<<" "<<j<<endl;**

**}**

**if(j>t.length)**

**{**

**//cout<<i-t.length<<endl;**

**return i-t.length;**

**}**

**else**

**{**

**return 0;**

**}**

**}**

1. **Next**

**//求t的next**

**void getn(sst t,int next[])**

**{**

**int i=1,j=0;**

**next[1]=0;**

**while(i<t.length)**

**{**

**if(j==0||t.ch[i]==t.ch[j])**

**{//相等的情况**

**i++;**

**j++;**

**//pi=pj,next[j+1]=next[j]+1**

**next[i]=j;**

**}**

**else**

**{**

**j=next[j];**

**}**

**}**

**}**

1. **Next优化**

**//next优化**

**void getnv(sst t,int nextv[])**

**{**

**int i=1,j=0;**

**nextv[1]=0;**

**while(i<t.length)**

**{**

**if(j==0||t.ch[i]==t.ch[j])**

**{**

**i++;**

**j++;**

**if(t.ch[i]!=t.ch[j])**

**{**

**nextv[i]=j;**

**}**

**else**

**{**

**nextv[i]=nextv[j];**

**}**

**}**

**else**

**{**

**j=nextv[j];**

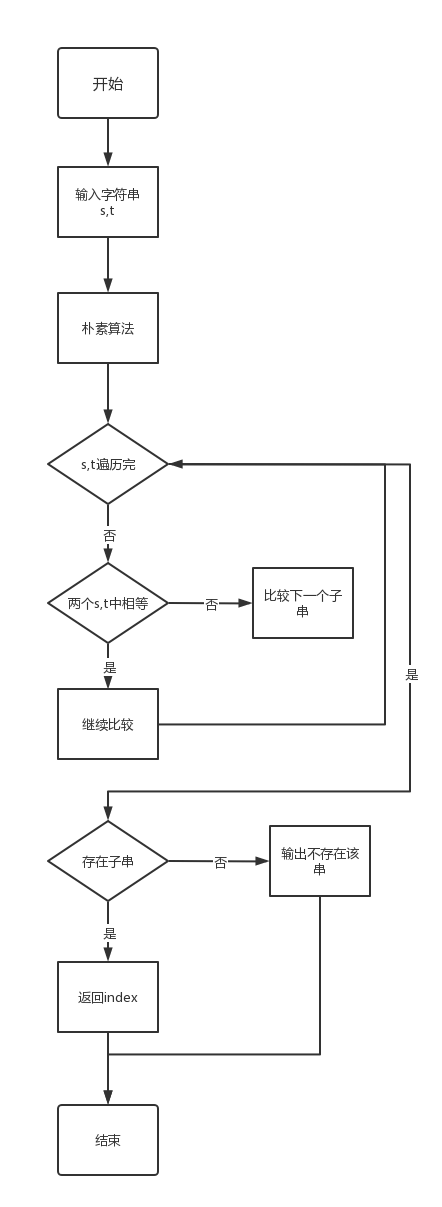
**}**

**}**

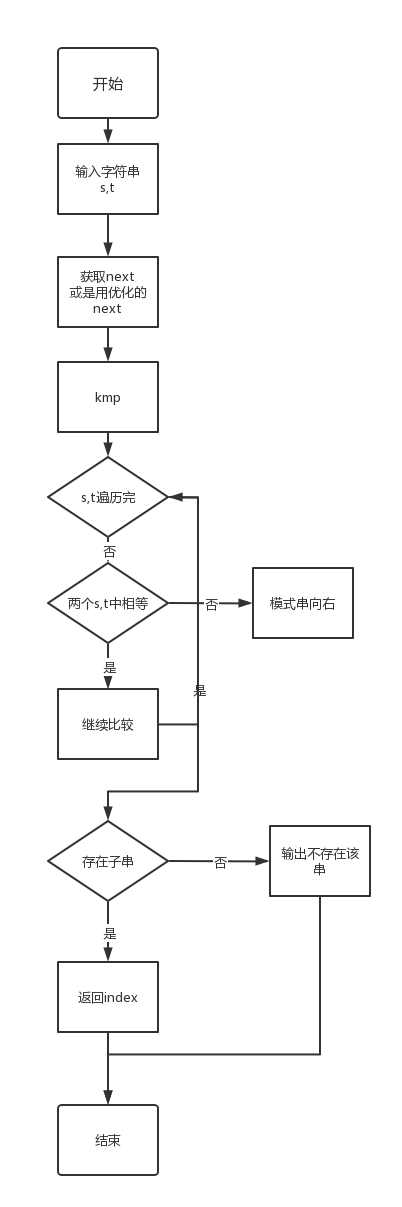
**}**

**1.4设计图**

**1）朴素模式**

****

**2）Kmp**

****

**1.5程序**

**#include<cstdio>**

**#include<cstdlib>**

**#include<cstring>**

**#include<iostream>**

**#include<cmath>**

**#define MAXSIZE 100086**

**using namespace std;**

**typedef struct lnode**

**{**

**char ch[MAXSIZE];//每个存储一个字符**

**int length;//实际长度**

**}sst;**

**int next[10086],nextv[10086];**

**//朴素算法**

**int mats(sst s,sst t)**

**{**

**//int k=1;**

**int i=1,j=1;**

**while(i<=s.length&&j<=t.length)**

**{**

**if(s.ch[i]==t.ch[j])**

**{**

**//cout<<s.ch[i]<<t.ch[i]<<endl;**

**i++;**

**j++;//继续比较**

**}**

**else**

**{**

**/\*k++;**

**i=k;**

**j=1;**

**or\*/**

**i=i-j+2;//检查下一个子串**

**j=1;**

**}**

**//cout<<i<<" "<<j<<endl;**

**}**

**if(j>t.length)**

**{**

**//cout<<i-t.length<<endl;**

**return i-t.length;**

**}**

**else**

**{**

**return 0;**

**}**

**}**

**//输入字符串**

**sst inputs()**

**{**

**//如果用cin输入，字符串会从0开始**

**sst s1;**

**char c;**

**int i=0;**

**while(1)**

**{**

**scanf("%c",&c);**

**if(c==' '||c=='\n')**

**{**

**break;**

**}**

**i++;**

**s1.ch[i]=c;**

**//cout<<i<<endl;**

**//cout<<s1.ch[i]<<endl;**

**}**

**s1.length=i;**

**//s1.ch[i+1]='\0';**

**return s1;**

**}**

**//kmp**

**int mats2(sst s,sst t,int next[])**

**{**

**//int k=1;**

**int i=1,j=1;**

**while(i<=s.length&&j<=t.length)**

**{**

**if(s.ch[i]==t.ch[j]||j==0)**

**{**

**//cout<<s.ch[i]<<t.ch[i]<<endl;**

**i++;**

**j++;//继续比较**

**}**

**else**

**{**

**j=next[j];//模式串向右**

**}**

**//cout<<i<<" "<<j<<endl;**

**}**

**if(j>t.length)**

**{**

**//cout<<i-t.length<<endl;**

**return i-t.length;**

**}**

**else**

**{**

**return 0;**

**}**

**}**

**//求t的next**

**void getn(sst t,int next[])**

**{**

**int i=1,j=0;**

**next[1]=0;**

**while(i<t.length)**

**{**

**if(j==0||t.ch[i]==t.ch[j])**

**{//相等的情况**

**i++;**

**j++;**

**//pi=pj,next[j+1]=next[j]+1**

**next[i]=j;**

**}**

**else**

**{**

**j=next[j];**

**}**

**}**

**}**

**//next优化**

**void getnv(sst t,int nextv[])**

**{**

**int i=1,j=0;**

**nextv[1]=0;**

**while(i<t.length)**

**{**

**if(j==0||t.ch[i]==t.ch[j])**

**{**

**i++;**

**j++;**

**if(t.ch[i]!=t.ch[j])**

**{**

**nextv[i]=j;**

**}**

**else**

**{**

**nextv[i]=nextv[j];**

**}**

**}**

**else**

**{**

**j=nextv[j];**

**}**

**}**

**}**

**int main()**

**{**

**sst s,t;**

**s=inputs();**

**t=inputs();**

**getn(t,next);**

**//getnv(t,next);**

**//cout<<s.length<<" "<<t.length<<endl;**

**/\*for(int i=1;i<=s.length;i++)**

**{**

**cout<<s.ch[i];**

**}**

**cout<<endl;**

**for(int i=1;i<=t.length;i++)**

**{**

**cout<<t.ch[i];**

**}\*/**

**int k=mats(s,t);**

**//朴素算法**

**if(k)**

**{**

**cout<<"匹配成功,在"<<k<<endl;**

**}**

**else**

**{**

**cout<<"匹配失败"<<endl;**

**}**

**int k2=mats2(s,t,next);**

**//kmp**

**if(k2)**

**{**

**cout<<"匹配成功,在"<<k2<<endl;**

**}**

**else**

**{**

**cout<<"匹配失败"<<endl;**

**}**

**/\***

**for(int i=1;i<=t.length;i++)**

**{**

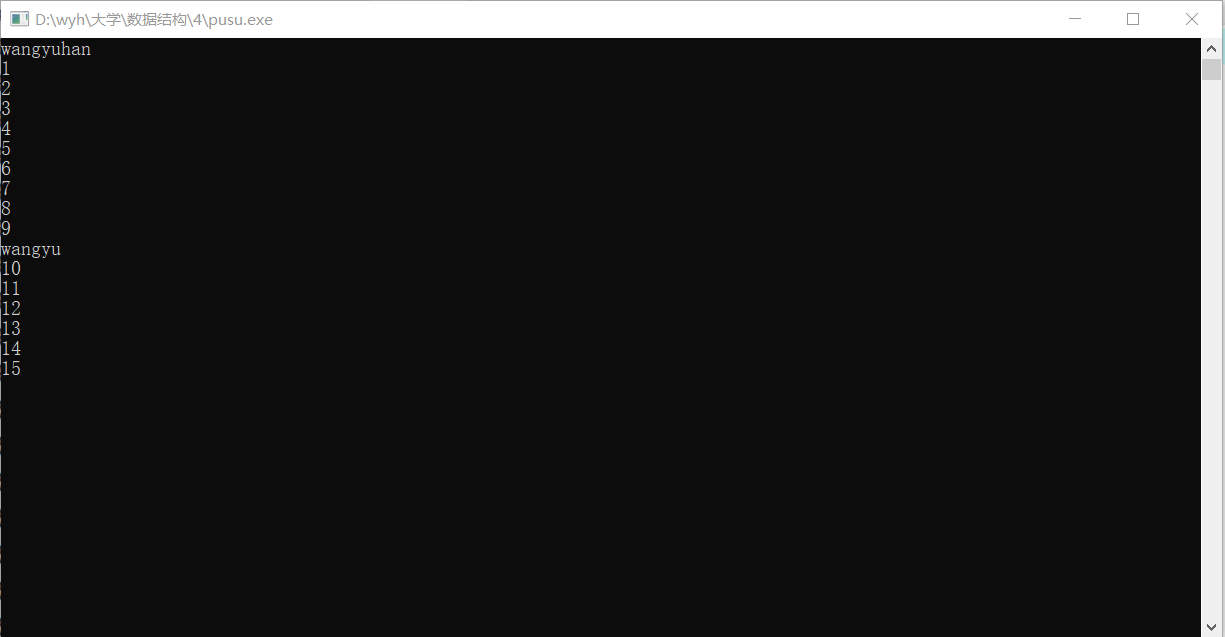
**cout<<next[i]<<" ";**

**}\*/**

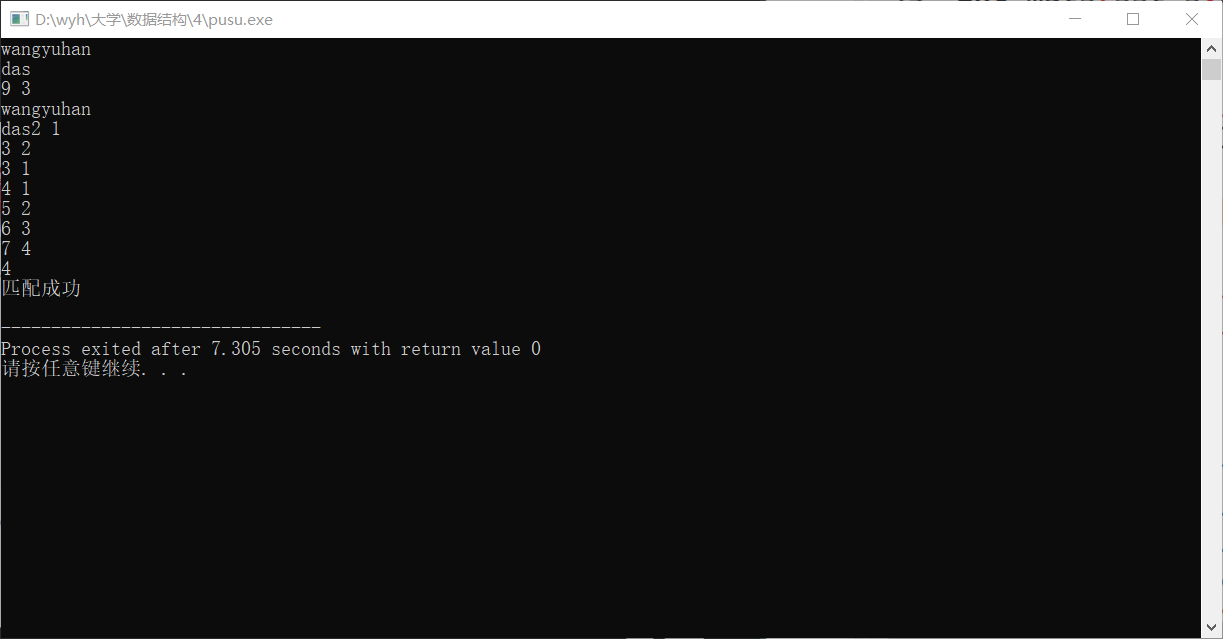
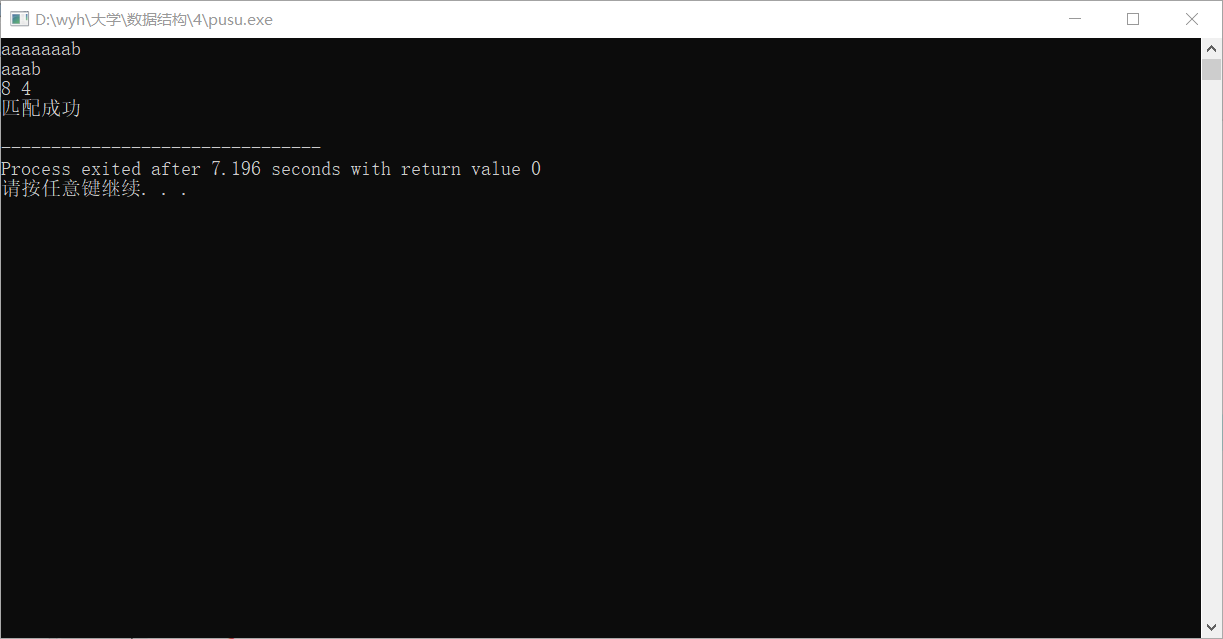
**return 0;**

**}**

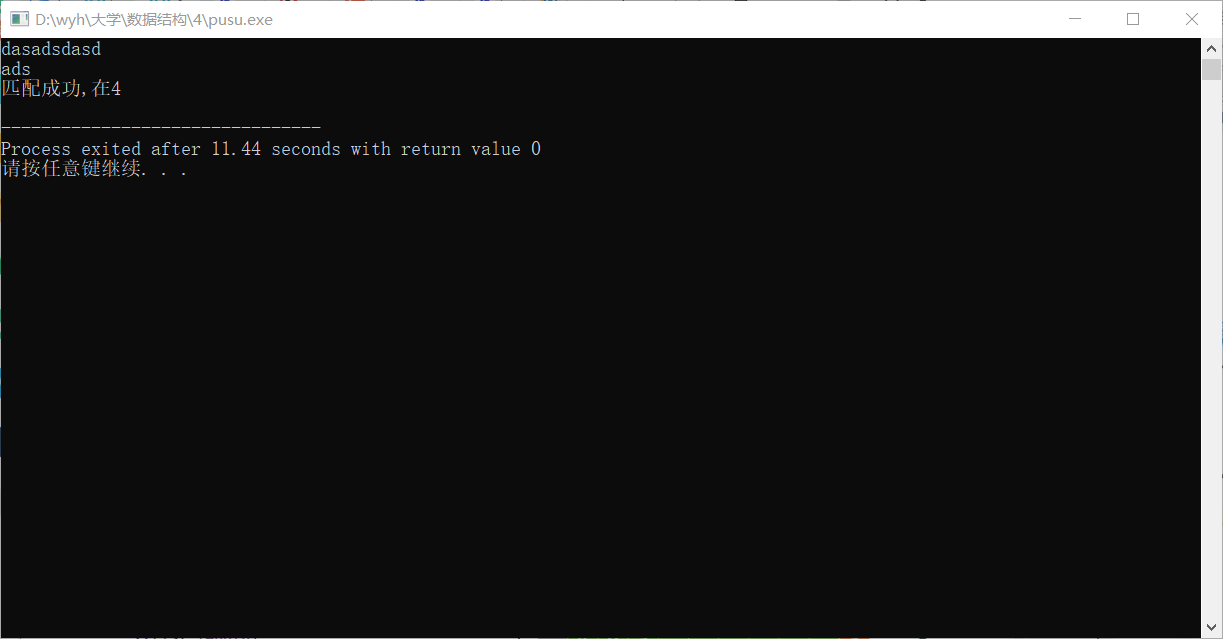
**1.6调试过程截图**

****

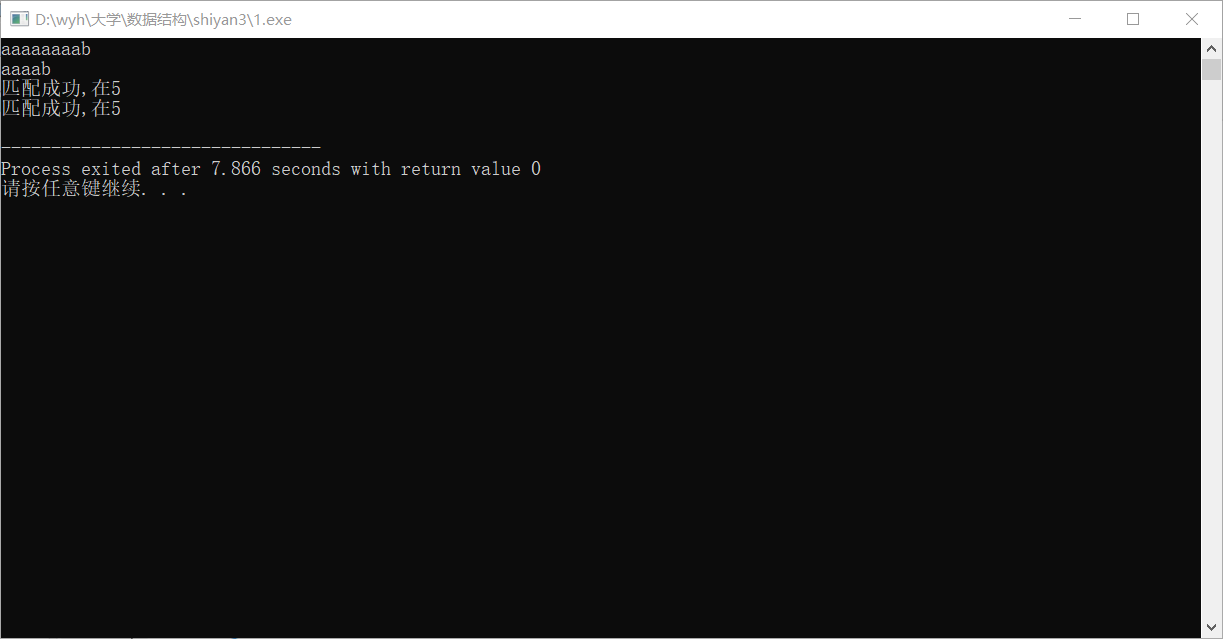
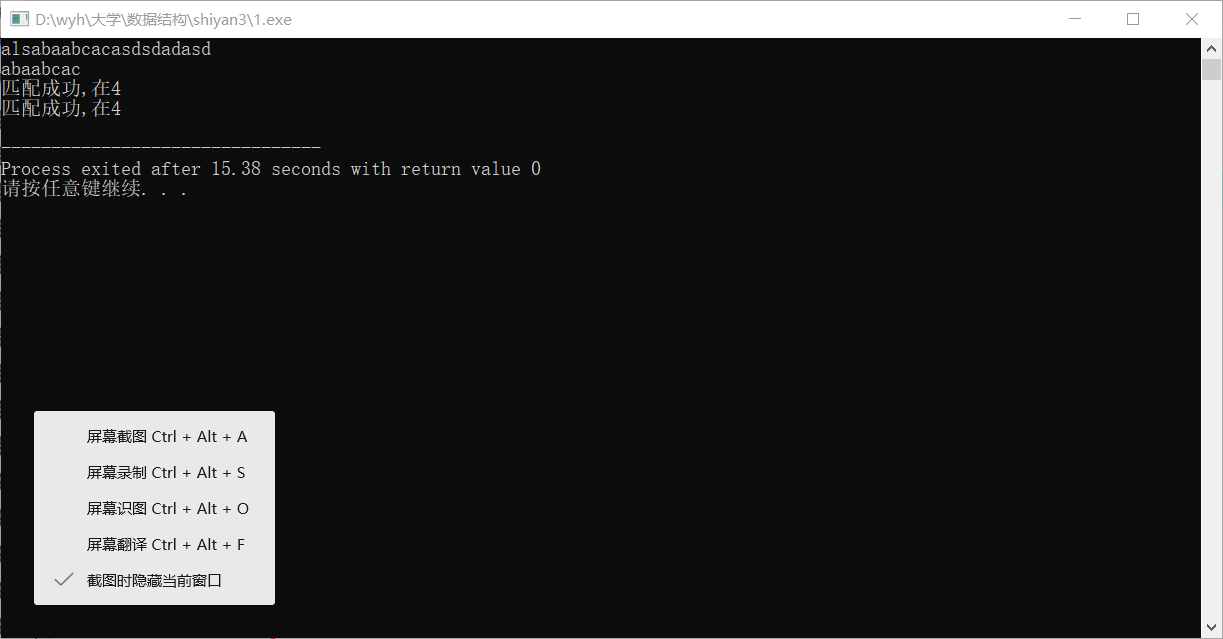
**一开始用cin输入有问题**

****

**1.7结果截图**

****

**仅朴素模式**

****

1. **实验心得**

朴素算法，其实现过程没有任何技巧，就是简单粗暴地拿一个串同另一个串中的字符。朴素算法简单并且容易理解，但是有些情况下时间效率不高，最好情况下时间复杂度为O(m)，平均时间复杂度为O(m)，最坏情况下时间复杂度为O(n×m)为了克服主串下标i在若干个字符序列比较相等后，只要有一个字符比较不相等便需要把下标i的值回退（回溯）的缺点，提出了改进的匹配算法KMP。而KMP算法的主要思想是当某一个字符与主串不匹配时，我们应该知道j指针要移动到哪里，从而对其进行优化。当然期间也碰到了一些问题，比如scanf不熟悉代替了cin导致了输入的问题，对c语言的语法还是要扎实。

通过本次实验，我对串有了深刻的了解，知道了串存储结构，也就是存储字符串的数据结构，也了解了各种串相关的算法。基本掌握了关于字符串的一些基本操作，例如串赋值、串比较、求串长、串联结、以及求子串，等等操作。我还了解到，串在机内的3种表示方法分别为定长顺序存储、堆分配存储、块链存储表示。

同时还明白了算法的优化，在今后的编程过程中要更注重代码复杂度，用更多的注释，让自己的思路更清晰。