# 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**

**实验名称： 字符串的模式匹配**

**实验完成人：**

**姓名：**\_\_赵冉\_\_\_\_\_\_**学号：**\_\_2020211731\_\_\_**成绩：**\_\_\_\_\_\_\_\_

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

**日 期： 2021 年 10 月 25 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（见附件）。

**五、实验过程和实验结果**

**串的模式匹配问题**

**(1)问题分析**

子串的定位操作通常称做串的模式匹配，是各种串处理系统中最重要的操作之一。算法的基本思想是，从主串S的第pos个字符起和模式的第一个字符比较，若相等，则继续逐个比较后续字符；否则从主串的下一个字符起再重新和模式的字符比较。依此类推，直至模式T中的每个字符依次和主串S中的一个连续的字符序列相等，则称为匹配成功，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，否则称为匹配不成功，函数值为零。后来，为了提高算法效率，又提出了KMP算法。此次实验主要应用KMP算法进行解决。

**（2）设计方案**

设计两个字符数组接受主串和模式串，接着设计一个KMP算法的入口进行模式匹配，最后返回开始匹配的字符位置，对结果进行打印。

1. **算法（实现代码）**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAXCHAR 100

#define MAXINT 100

#define OK 1

#define ERROR 0

typedef int Status ;

**//得到next数组的函数**

Status getNext(char\*t, int \*next) {

**//得到字符串t的长度**

int lt = strlen(t) ;

**//i为模式串的符号，j为模式串的第二个符号，k为next数组的符号**

int i = 0 , j = 1 ;

while(j < lt + 1 && i < j) {

if(t[i] == t[j]) {

next[j] = i + 1 ;

i ++ ;

j ++ ;

}

else{

if(i != 0) i = next[i - 1] ; **//如果i不是第一个，跳转**

else j ++ ;  **//如果i是第一个，只移动j，进行比较**

}

}

return OK ;

}

**//KMP函数的入口**

Status KMP\_Entrance(char \*s, char \*t, int \*next) {

int i = 0, j = 0 ;

int ls = strlen(s) ;

int lt = strlen(t) ;

**//主串和匹配串开始逐字符比较**

while(i < ls && j < lt) {

if(s[i] == t[j]) {

i ++ ;

j ++ ;

}

else{

**//将t[j]回溯到前缀与后缀相等时前缀末尾的下一个元素**

if(j != 0) j = next[j - 1] ;

else i ++ ;

}

}

**//判断是否匹配**

if(j >= lt) {

return i - lt ;

}

else {

return -1 ;

}

}

int main() {

**//接受两个字符串s和t**

char s[MAXCHAR] ;

char t[MAXCHAR] ;

printf("**请输入主串：**\n") ;

fflush(stdin) ;

scanf("%s", &s) ;

printf("**请输入搜索串：**\n") ;

fflush(stdin) ;

scanf("%s", &t) ;

**//得到KMP算法中的next数组**

int next[MAXINT] = {0};

getNext(t, next) ;

**//进行字符串的比较**

int tag = KMP\_Entrance(s, t, next) ;

**//结果打印**

if(tag != -1) {

printf("**匹配，开始匹配位置为:**%d\n", tag) ;

printf("**ps:主串的起始位置视为0**") ;

}

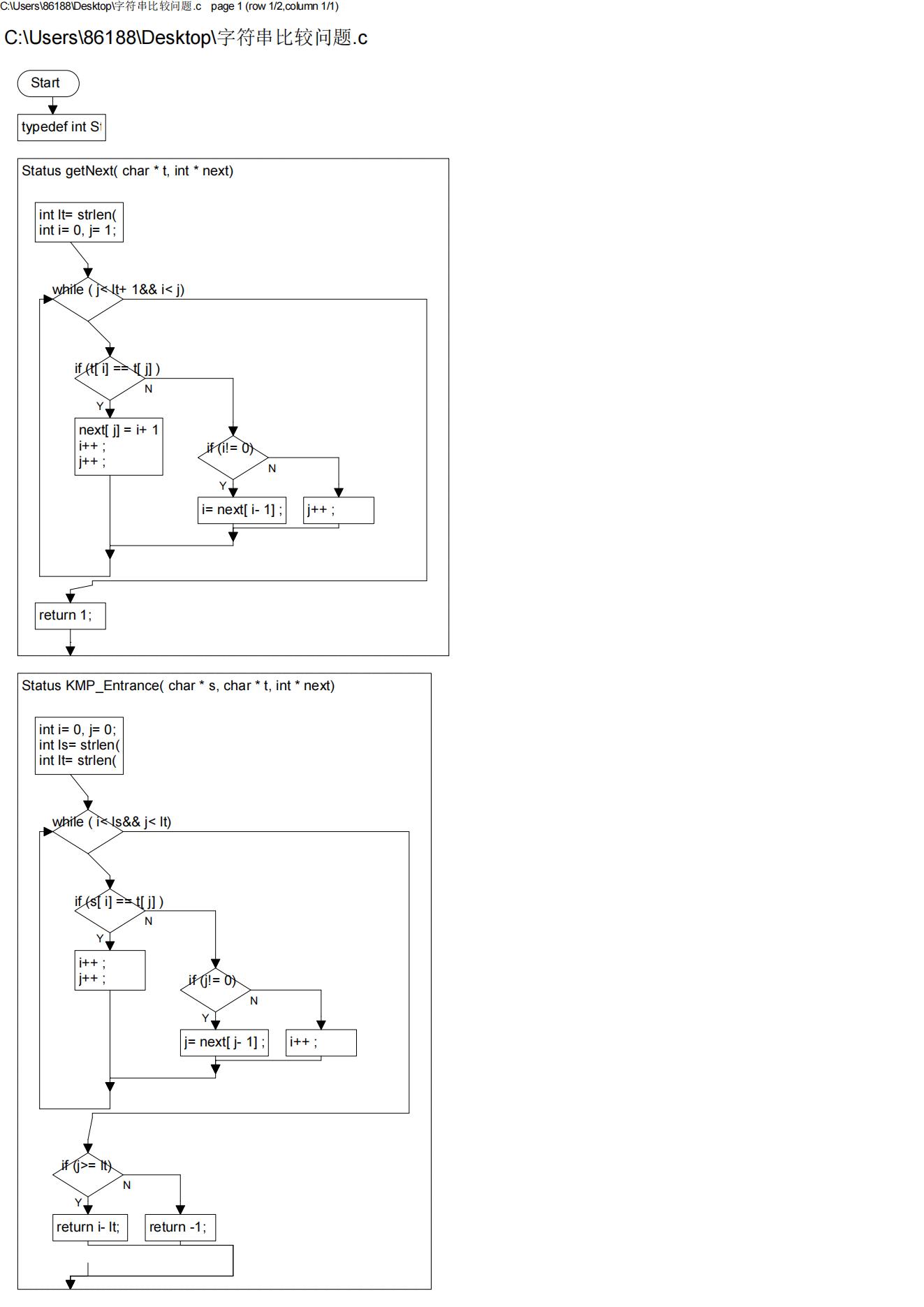
else printf("**主串中无匹配子串**") ;

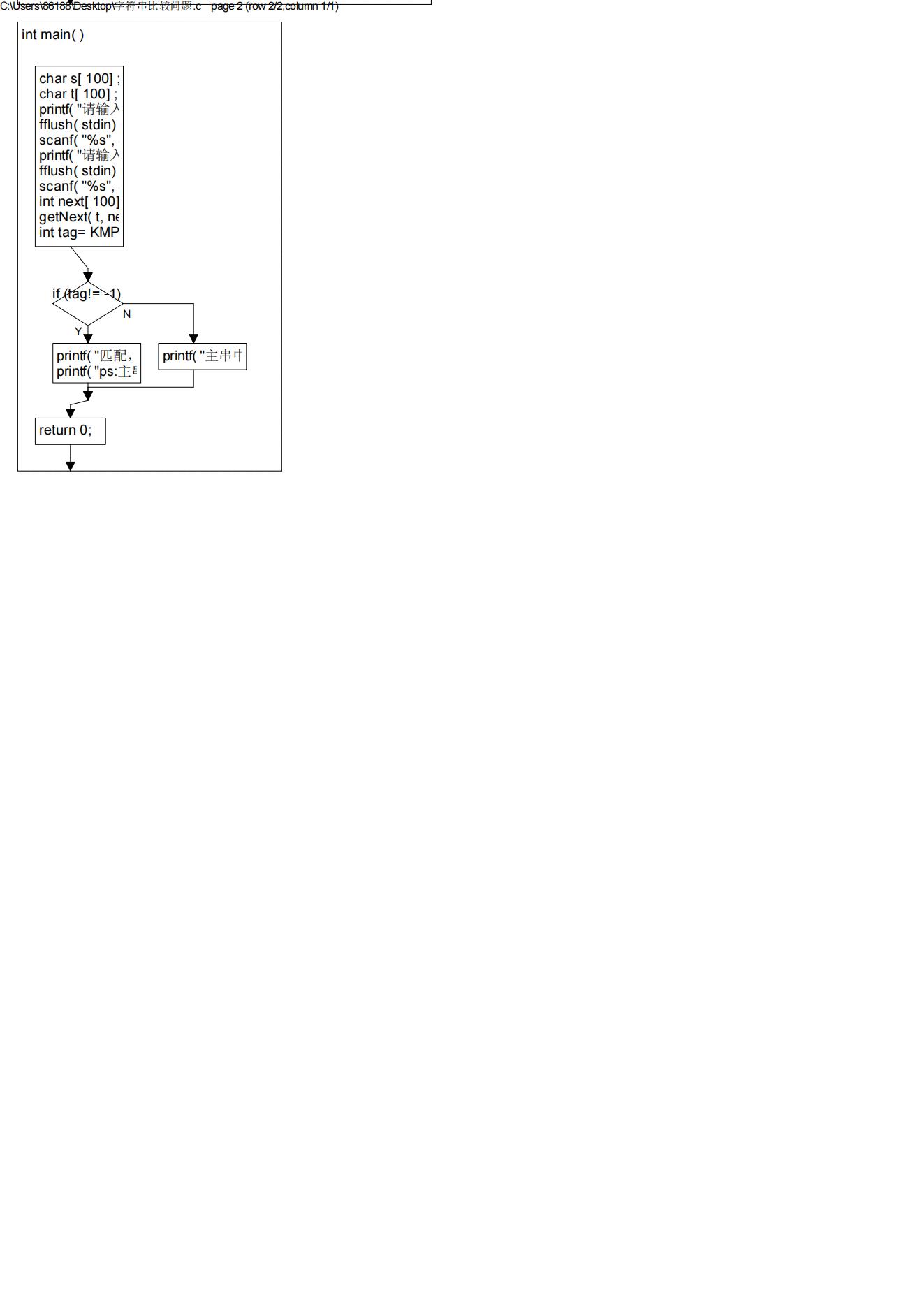
return 0 ;

}

**（4）设计图**

算法设计图如下：



****

1. **程序**

下述为程序截图：

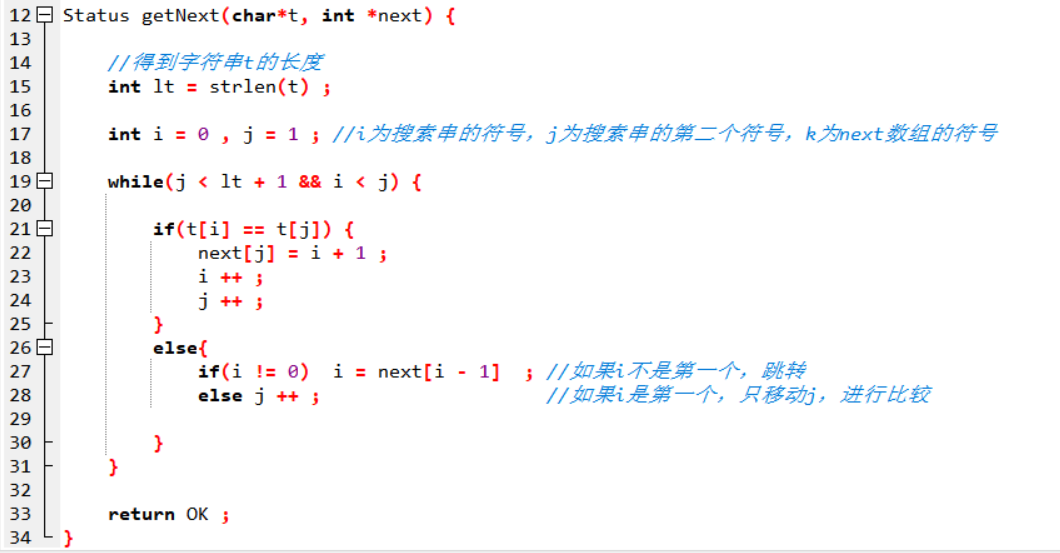


图1.1得到getNext()函数

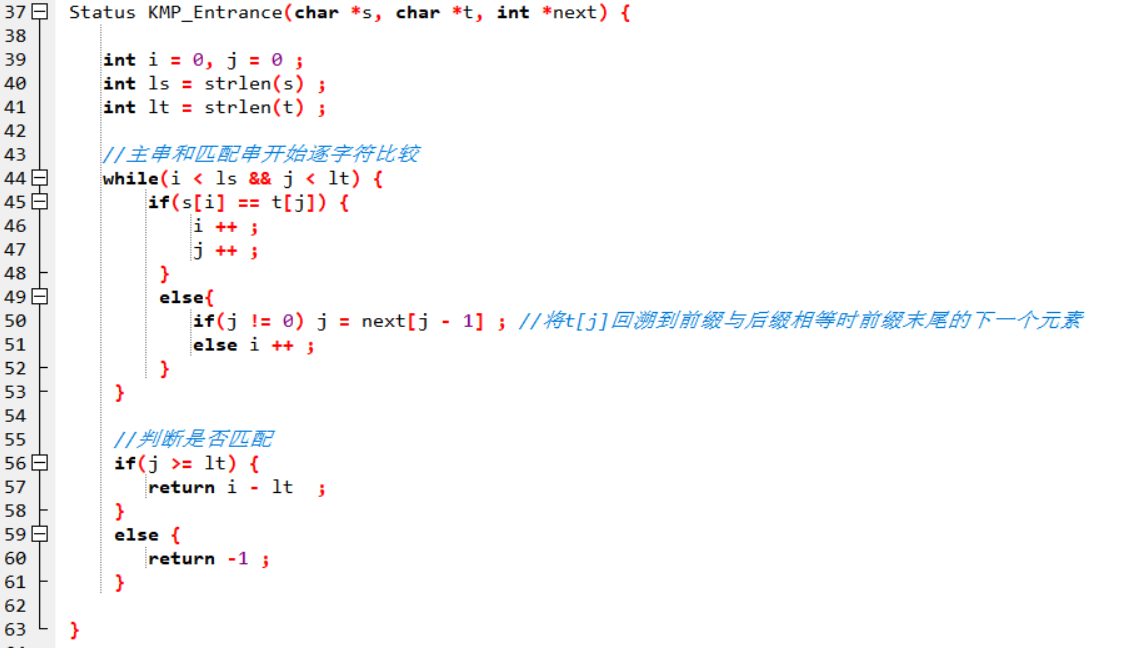


图1.2KMP函数的接口

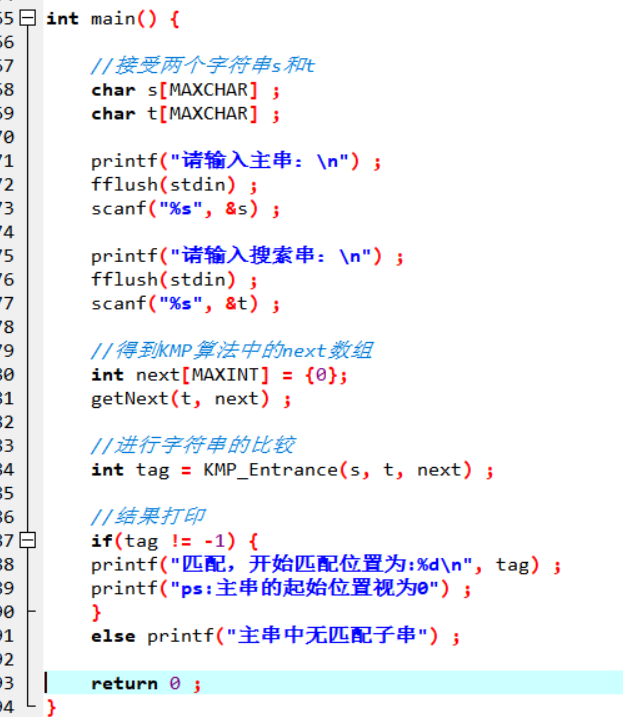


图1.3main()函数

1. **调试过程截图**

调试过程中遇到的问题截图：

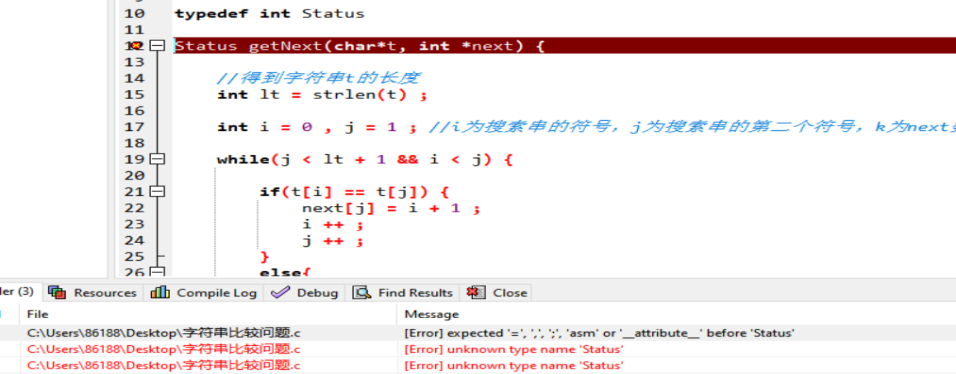


图1.4代码调试截图（1）

1. **运行结果截图**



图1.5

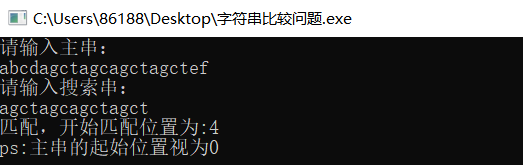


图1.6

**六、实验心得**

此次实验重点考察对于字符串的理解。由于计算机上的非数值处理的对象基本上是字符串数据，而又因为当今我们使用的计算机的硬件结构主要是反映数值计算的需要，因此，在处理字符串数据时比处理整数和浮点数要复杂，所以学习和理解字符串是必须且必要的。

通过这次实验，我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。了解字符串在机内的存储方式，能更好地帮助我们理解字符串的本质，更好地对其进行操作。

本次实验我们主要解决的问题是串的模式匹配算法。按照算法的基本思想，需要遍历主串和模式串逐个比对，它的时间复杂度为O（mn），为了降低算法的复杂度，研究者们提出了KMP算法。此算法可以在O（n+m）的时间数量级上完成串的模式匹配操作。其改进在于，每当一趟匹配过程中出现字符比较不等时，不许回溯i指针，而是利用已经得到的“部分匹配”的结果将模式向右“滑动”尽可能远的一段距离后，继续进行比较。算法的关键在于求得next数组的值，对于这一块的理解，我是借助数学归纳法理解的。

这次实验我不仅对串有了更深的理解，同时还学习了一种全新的串匹配算法KMP，收获颇丰。