# 字符串模式匹配

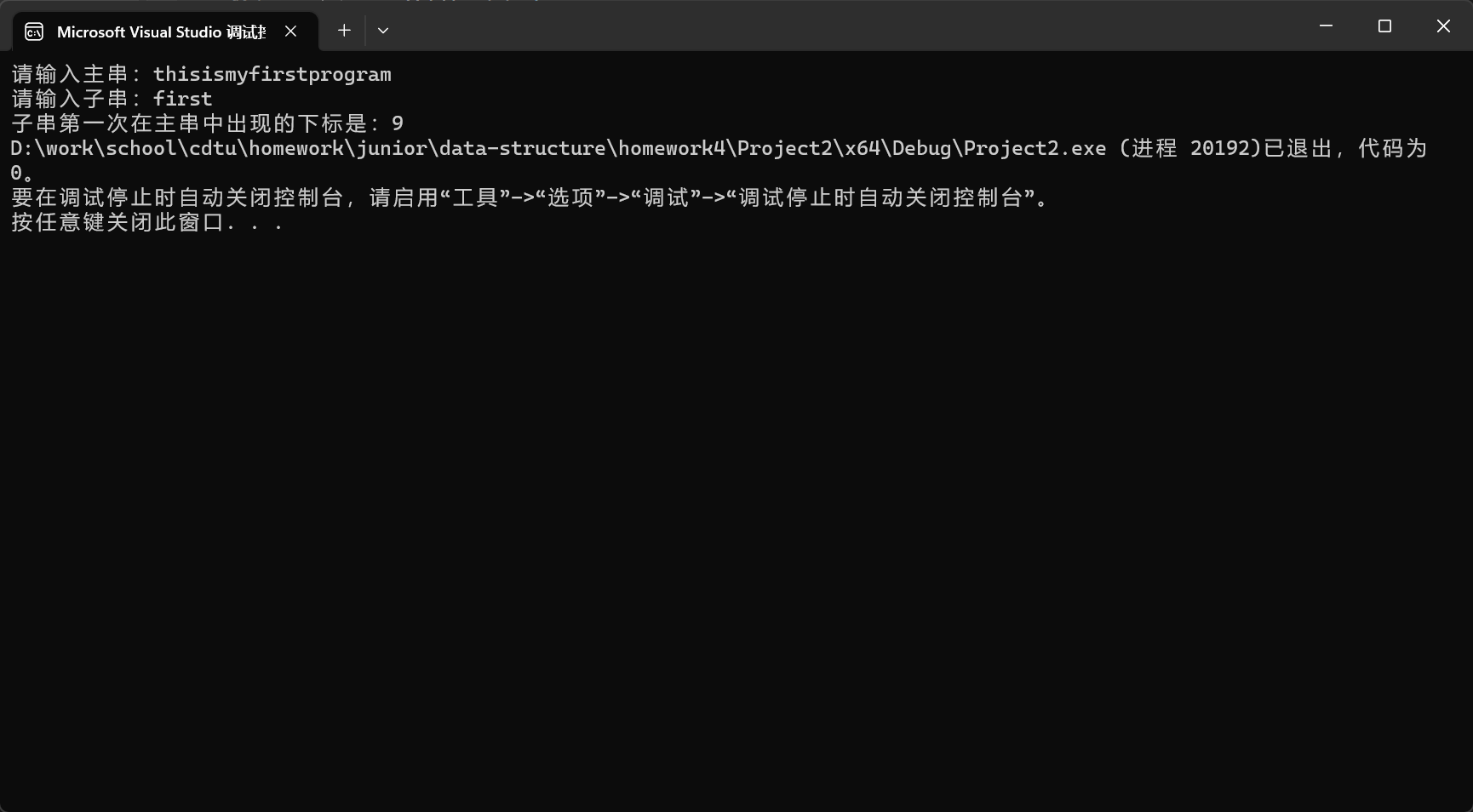
## BF算法

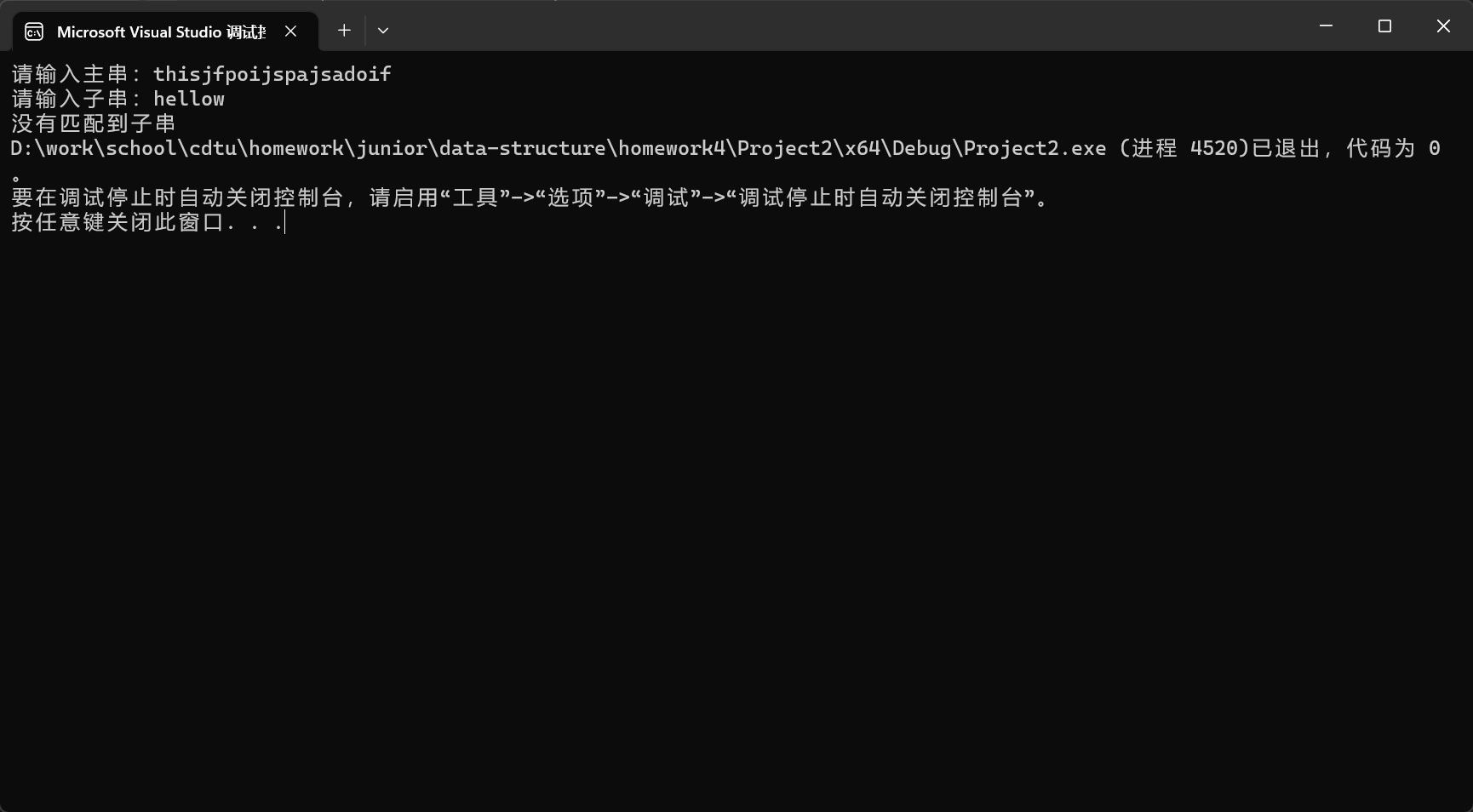
### 源码

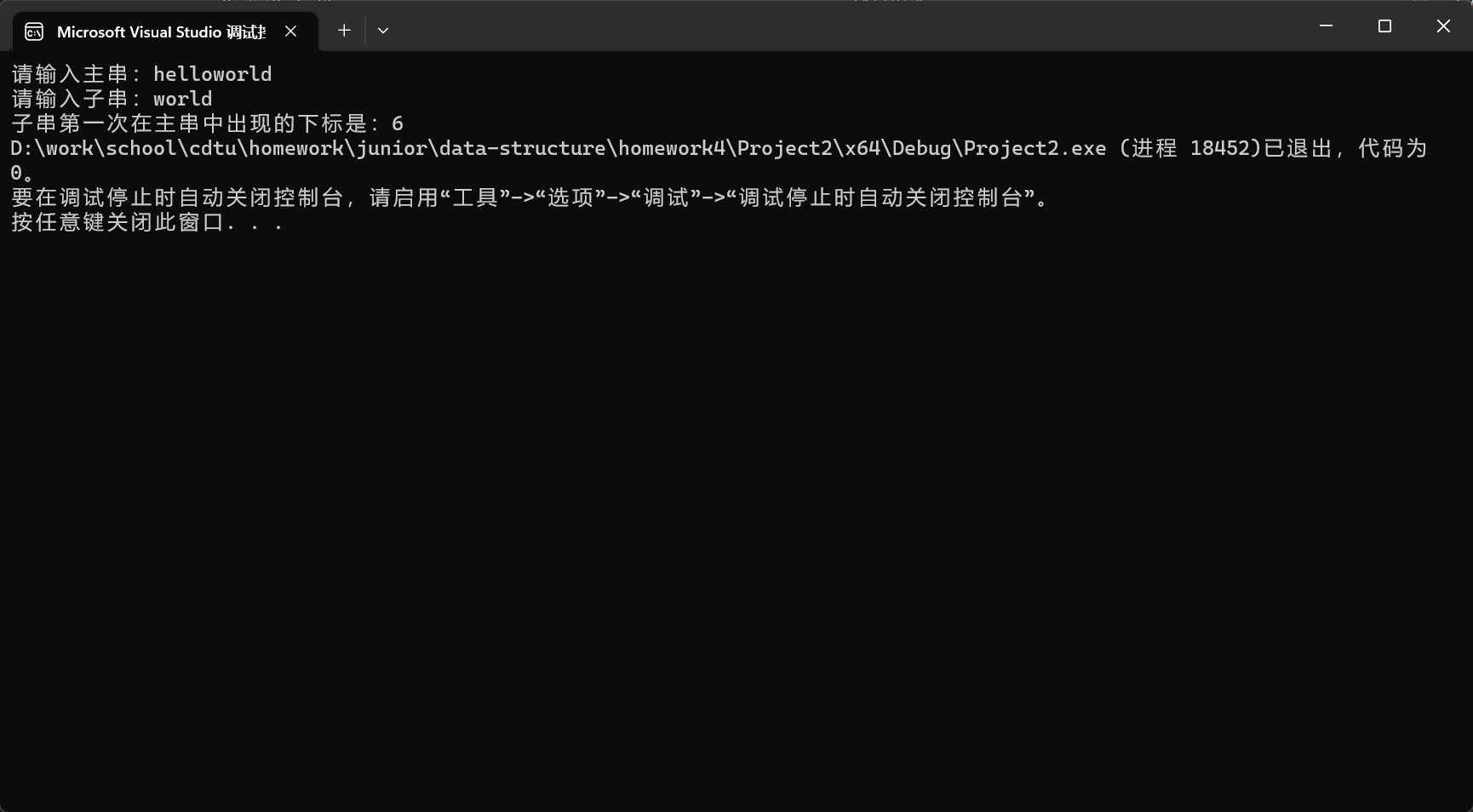
**Main.cpp**

|  |
| --- |
| #include <iostream>  #include <stdlib.h>  using namespace std;  /\*\*  \* 获取子串在主串中第一次出现的起始位置，不存在则返回0  \* param s 主串  \* param t 子串  \* return 返回子串在主串中第一次出现的下标（从1起标），不存在则返回0  \*/  int BF(char\* s, char\* t) {  int start = 0; // 设置主串每次匹配的起始位置  int i = 0, j = 0; // i记录主串某次匹配的位置变动，j记录子串匹配的位置变动  /\*  当主串下标不为空时说明主串还没匹配完，  当子串下标不为空时说明子串还没被匹配完过  当上面两个条件同时满足才执行循环  \*/  while ((s[i] != '\0') && (t[j] != '\0')) {  /\*  如果在某次匹配中，子串能与主串匹配上就一直往下匹配，  直到主串到底，或者子串到底  主串到底说明已经没法匹配了  子串到底说明找到了子串在主串中的位置  \*/  if (s[i] == t[j]) {  i++;  j++;  }  /\*  如果某次匹配没有匹配上，就说明主串的这个位置匹配不上，  匹配的主串起始位置就往下走一位，  并将记录主串某次匹配的位置更新为主串新的匹配起始位置，  记录子串匹配的位置初始化为子串的第一个字符位置  \*/  else {  start++;  i = start;  j = 0;  }  }  /\*  当匹配完成后如果子串的下标匹配记录为\0，  则说明子串匹配成功了，就返回最后这次匹配的主串起始位置+1（方法返回从1起标）。  如果子串下标匹配记录不为\0，说明是主串的下标匹配记录为\0，  表明主串已经匹配到最后一个字符了，也没有匹配到子串，就返回0，表示没有找到子串  \*/  if (t[j] == '\0') {  return start + 1;  }  else {  return 0;  }  }  int main() {  // 给两个字符指针分配内存，方便存字符串  char \*s = (char\*) malloc(1024), \* t = (char\*)malloc(1024);  // 输入  cout << "请输入主串：";  cin >> s;  cout << "请输入子串：";  cin >> t;  if (int index = BF(s,t)) {  cout << "子串第一次在主串中出现的下标是：" << index;  }  else {  cout << "没有匹配到子串";  }  // 内存释放  free(s);  free(t);  return 0;  } |

### 运行结果







## KMP算法

### 源码

**Main.cpp**

|  |
| --- |
| #include <stdlib.h>  #include <iostream>  using namespace std;  // 用于装载数组  struct Arr {  int\* data; // 数组指针  int length; // 数组长度  };  /\*  获取子串的next数组  param t 子串的首地址  return 返回next数组  \*/  Arr\* getNext(char\* t);  /\*  通过KMP算法获取子串在主串中第一次出现的下标（从0起标），未找到返回-1  param s 主串  param t 子串  return 子串在主串中第一次出现的下标（从0起标），未找到返回-1  \*/  int KMP(char\* s, char\* t);  int main() {  // 创建字符串接收变量  char\* s = (char\*) malloc(1024);  char\* t = (char\*)malloc(1024);  // 终端输入  cout << "请输入主串：";  cin >> s;  cout << "请输入子串：";  cin >> t;  // 输出结果  int index = KMP(s, t);  if (index == -1) {  cout << "没有在主串中找到子串";  }  else {  cout << "子串在主串中第一次出现的下标为：" << index;  }    // 释放内存  free(s);  free(t);  return 0;  }  /\*  获取子串的next数组  param t 子串的首地址  return 返回next数组  \*/  Arr\* getNext(char\* t) {  Arr\* next = new Arr{ nullptr,0 };  // 计算数组长度  while (t[next->length] != '\0') {  next->length++;  }  // 给next数组分配空间  next->data = (int\*)malloc(sizeof(int) \* (next->length));  next->data[0] = 0;  next->data[1] = 0;  // 循环对应子串的每个字符下标  for (int i = 2; i < next->length; i++) {  int max = i - 1; // 当前子串字符下标需要比对的前后缀的最长长度  next->data[i] = 0; // 假想没有前后缀  // 根据前后缀最大长度，从最大前后缀一直比较到最小前后缀，找到下次对比子串的起始指针  for (int j = max; j >= 1; j--) {  int flag = true;  // 比较前后缀  for (int k = 0; k < j; k++) {  // 依次比较前后缀，如果没配对就表示本次循环的前后缀长度不是最长的  if (t[k] != t[max - j + k + 1]) {  flag = false;  break;  }  }  // 如果flag为真，则写入偏移量，并结束找最大前后缀  if (flag) {  // 将最大前后缀长度赋值给next数组并退出  next->data[i] = j;  break;  }  }  }  return next;  }  /\*  通过KMP算法获取子串在主串中第一次出现的下标（从0起标），未找到返回-1  param s 主串  param t 子串  return 子串在主串中第一次出现的下标（从0起标），未找到返回-1  \*/  int KMP(char\* s, char\* t) {  // 获取子串的next数组  Arr\* next = getNext(t);  // 分别为主串的比较下标记录与子串的比较下标记录  int start = 0, t\_i = 0;  // 当主串比较到完，或者子串比较完就终止循环  // 子串比较完说明找到了子串的下标，主串比较完说明没有找到  while (s[start] != '\0' && t[t\_i] != '\0') {  if (s[start] == t[t\_i]) {  t\_i++;  }  else {  t\_i = next->data[t\_i + 1];  }  start++;  }  // 子串比较完说明找到了子串的下标  if (t[t\_i] == '\0') {  // 返回子串第一次出现的下标  // 由于主串下标在比较时会往后走，所以返回时要减去子串的长度  return start - next->length;  }  // 子串没有比较完说明没有在主串中找到子串，返回-1  return -1;  } |

### 运行结果

