《计算机组成原理实验》

实验报告

（实验一）

学 院 名 称 ：	数据科学与计算机学院
学 生 姓 名 ：	任磊达
学　　号 ：	15352285
专业（班级）：	15 移动 3 班
时　　间 ：	2017　年　3　月　31　日

成 绩：

实验一 ： MIPS汇编语言程序设计

一. 实验目的

(1) 认识和掌握 MIPS 汇编语言程序设计的基本方法；

(2) 熟悉 PCSpim 模拟器的使用。

二. 实验内容

查找替换匹配字符或字符串。从键盘输入一字符或字符串与在内存某单元开始字符串相匹配，若找到匹配字符或字符串，则将键盘输入的一字符或字符串将其替换，并统计替换个数，将统计结果和替换后的整个字符串显示在屏幕上，如找不到匹配的则显示“No match!”。如：

”Replace?:”+??

“String?:”+??????????????????

三. 实验器材

Sublime2 ： 代码编辑环境，加Mips插件可进行高亮

PCSpim ： MIPS模拟器，可以运行程序，检查实验寄存器变化

Mars ： 备用MIPS模拟器，依据java虚拟机可在osx，Linux系统调试

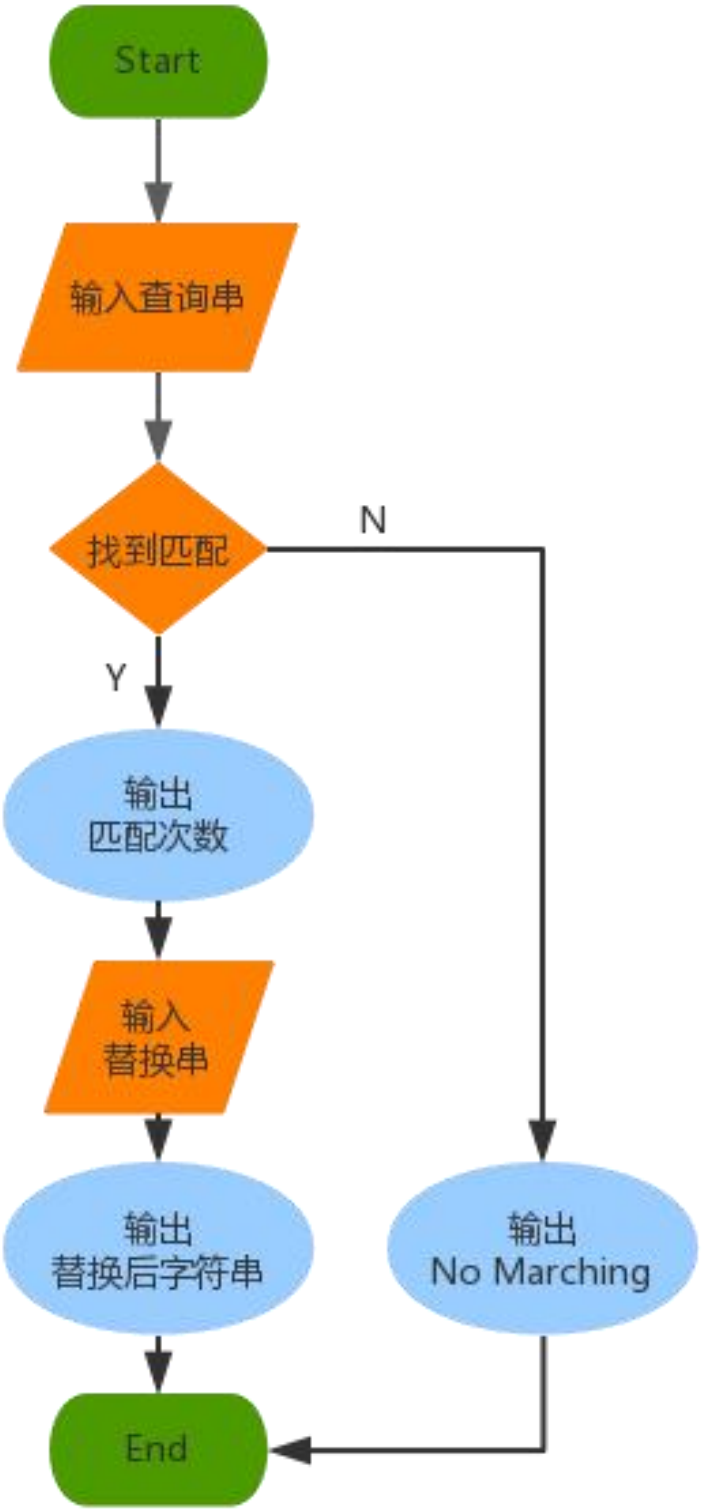
四. 实验分析与设计

PCSpim:

是一款著名的Mips进程模拟器，通过导入MIPS汇编代码，可以F5开始运行或F10单步调试。在程序界面分为寄存器部分，代码部分，存储器部分以及调试信息部分。默认.global 引入函数名需要是main，当执行到某一步时，会显示内存地址，指令代码以及汇编指令。另外与Mars稍有不同的地方在于单个字符的ascii码需要以数字的形式赋值给变量。

程序设计:

对于字符串的匹配，有朴素 $O(n^2)$ 的循环匹配算法，有通过字符串后缀数组，KMP 等优化的高效算法。因为本次实验主要目的在于练习 Mips 编程，于是采用朴素 $O(n^2)$ 的算法进行具体的实现。具体而言在输入字符串的基础上，从字典字符串的每个位置起点开始匹配。当匹配成功，计数器+1 并将该处起始点放入队列中。另外考虑到了输入的 replace 字符串长度大于原被替换字符串大小的情况，如果直接覆盖有可能溢出替换边界，于是重新开了一个字符串进行替换匹配。具体程序流程如下图所示：


Start
输入查询串
找到匹配
N
Y
输出
匹配次数
输入
替换串
输出
替换后字符串
输出
No Marching
End

图表 1 算法流程图

实验变量设计			
变量名	**变量含义**	**初始值**	**对应寄存器**
putInString	存储输入字符串		$s2,length($s1)
replaceString	存储更换字符串		$s6,length($s5)
savePosition	队列，存储匹配起始位置		$s4
putInSize	默认输入长度	100	
dictionary	查询字典，待匹配串	"helloworld myworld\n"	$3
replaced	替换后字符串		$8
putInTips/replaceTips/FailedSign/matchingfor/times_n	提供对应名字的字符串输入提示		

实验步骤

1．配置编辑，编译环境

2. 编写源代码（task1.asm）

A. 调用系统函数，显示提示信息

B. 调用系统函数，输入查找字符串

C. 给通用寄存器指定对应内容，数值型用add，字符串型用la

```
1       add $s0,$zero,$zero # save show times
2       add $s1,$zero,$zero # save length of string
3       la   $s2,putInString # save putinString address
4       la   $s3,dictionary   # save dictionary address
5       la   $s4,savePosition
6       add $s5,$zero,$zero # save length of replace
7       la   $s6,replaceString
8       addi$s7,$zero,10      # save '\n'
9       la   $s8,replaced      # save replaced
```

D. 通过循环模拟strlen，获取输入字符串长度

```
1       # obtain length of the input string
2       addi $t0,$s2,0
3       Strlen:
4           lb $t1,0($t0)   # attention ! use lb to obtain char
5           beq $t1,$s7,ExitStrlen
6           addi $t0,$t0,1
7           addi $s1,$s1,1
```

```
8           j Strlen
9       ExitStrlen:
```

E. 在自动中逐个匹配字符，看是否存在满足字符解

E.1 从寄存器内读取字符

```
1           lb $t2,0($t0)
2           lb $t3,0($t1)
```

E.2 判断字符串边界，匹配情况，不匹配跳转

```
1           beq $t2,$zero,ExitFind    # end of dictionary
2           bne $t2,$t3,AddPosition # not equal at first place


1 AddPosition:
2           addi $t0,$t0,1
3           j Find
```

E.3 首字符匹配，判断剩余字符是否匹配

这里值得注意的是使用两个零食寄存器来扫字符串，并且先考虑边界是否溢出再考虑当前对应位置字符串是否匹配。最后如果匹配则填入队列，并计数器+1，

```
1           addi $t4,$t0,1
2           addi $t5,$t1,1
3           addi $t6,$zero,1 # save length
4       CheckFullString:
5           beq $t6,$s1,AddCount
6           lb $t2,0($t4)
7           lb $t3,0($t5)
8           beq $t2,$s7,AddPosition
9           beq $t3,$s7,AddPosition # check boundary
10          bne $t2, $t3,AddPosition
11          addi $t4,$t4,1
12          addi $t5,$t5,1
13          addi $t6,$t6,1
14          j CheckFullString
15      AddCount:
16          addi $s0,$s0,1
17          sw $t0,($t7)
18          addi $t7,$t7,4
```

F. 总结字符串匹配情况

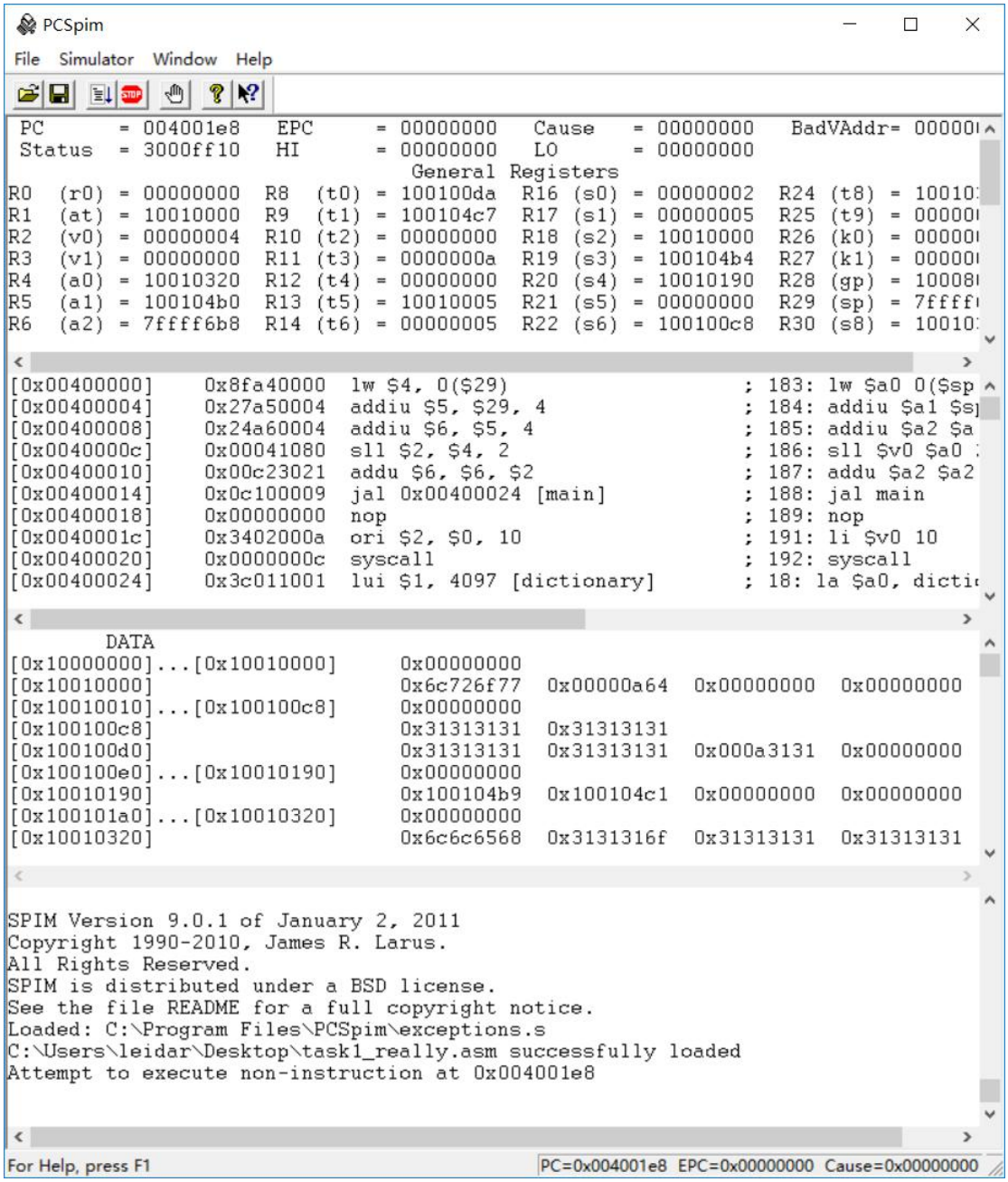
F.1 寻找到字符串成功匹配

F.1.1 输入替换字符串

F.1.2 使用新字符串逐位赋值，最后输出。

F.2 未寻找到成功匹配，输出No Matching

3. 使用PCSpim进行编译调试操作



4. 编写C++代码进行对比测试（见task1.cpp及文末代码）

5. 尝试相关实验代码编写与调试（见sort.asm及实验心得）

实验结果：

A. 测试比原有字符串短的字符串

helloworld myworld

String?:world

matching for 2 times

Replace?:rld

hellorld myrld

测试成功，可以得到比原有字符串短的字符串

B. 测试比原有字符串长的字符串

helloworld myworld

String?:world

matching for 2 times

Replace?:peking

hellopeking mypeking

测试成功，可以得到比原有字符串长的字符串

C. 测试重复出现数据串：

helloworld myworld

String?:l

matching for 4 times

Replace?:ll

helllloworlld myworllld

测试成功，由于不是在原有字符串上操作，可以避免无限循环的安全隐患

D. 测试未能找到的字符串

helloworld myworld

String?:aorld

No match!

测试成功

五. 实验心得

1. 本次实验中首先遇到的是设计层面的问题。对于一串字符串的替换，如果在源内存空间进行操作，很容易造成覆盖原有内存非替换部分（当替换字符串长度大于原有字符串时候），造成无限替换以至于程序的死循环（如‘1’替换成 11）。针对这个问题本次实验中初步采用的是用数据结构存储第一步查找得到的位置信息，在 replace 时候直接输出现有字符串。（见 p12 直接输出函数）。后期修改

```
1                   li $v0,11
2                   syscall
```

程序中这两句话，改为

```
1                   sb $a0,0($t8)
2                   addi $t8,$t8,1
```

之后便可以存入新的字符串。值得注意的是这里的字符串每个字符只需要 8 个 bits，也就是每次在地址空间只需要移 1 位。

2. 本次实验作为初次接触 MIPS 编程项目，同样在代码编写环节犯下了一定的错误。比如

a. 最开始没有很好的分清 lw 和 sw 的区别，认为只要调换顺序便可以等价。而忽视了这两个操作对象的不同之处。同样最初也没有弄清楚 a0 和 v0 的关系。这些问题的解决方案是在程序中输出变量值，可以清晰的看到是否正确的进行了 IO，赋值等操作。

b. 部分语句加载字符串、数组空间时候使用了 add、li 来加载，而不是通过 la 指向字符串首地址。而后在接下来的循环中混淆了地址指针和地址指针所指向信息（相比之下 C 指针的取值符显得更加明显），花费了很长时间进行调试。这个问题的解决方案是对于逐条语句分析，并补充注释信息。

c. 同时在程序运行过程中遇到了一些小问题，仔细思考之后能很顺利的解决：

如

- Mips 修改程序之后需要重新 open；
- 通过 li $v0 8 输入的字符串是带有‘\n’的。
- 字符串边界条件的判断是匹配的前提条件
- Mars 编译成功的程序在 PCM 内认为‘\n’不能单独出现
- 遗漏 syscall 调用，使得程序没有输入输出
- 遗漏字符大小信息，lb，lw 之间赋值出现误差（输出可以明显地检查到）

3. 另外我也尝试完成了排序实验，**从内存读取 10 个无符号字数并从大到小进行排序，排序结果保存在内存中。**（见 loop.asm）

思路： 首先给内存中读入十个数字，然后通过两重循环进行冒泡排序。然后将结果输出到系统标准输出。其中为了优化数据的 swap 操作使用 xor 亦或三次。

```
29                  xor $s0,$s0,$s1
30                  xor $s1,$s0,$s1
31                  xor $s0,$s0,$s1
```

另外值得注意的一点是第二层循环遍历数组时候，使用 la 将 array 的地址赋值给$t3，然后对于每个数 lw 读入数据到$s12 寄存器中，在 swap 函数内部进行$12 的比较。原因在于$t3 中存的是地址，不能直接比较对应元素大小。

【程序代码】

MIPS程序代码

```
1 ###################################################
2 # 查找替换匹配字符或字符串。
3 # 从键盘输入一字符或字符串与在内存某单元开始字符串相匹配，
4 # 若找到匹配字符或字符串，
5 # 则将键盘输入的一字符或字符串将其替换，
6 # 并统计替换个数，
7 # 将统计结果和替换后的整个字符串显示在屏幕上，
8 # 如找不到匹配的则显示“No match!”。如：
9 # ”Replace?：”+??
10 # “String?：”+??????????????????
11 ###################################################
12
13 .text
14 .globl main
15
16 main:
17
18      la $a0, dictionary
19      li $v0, 4
20      syscall      # show tips for putin string
21
22      la $a0, putInTips
23      li $v0, 4
24      syscall      # show tips for putin string
25
26      la $a0, putInString
27      la $a1, putInSize
28      li $v0, 8
29      syscall      # putin string
30
31      add $s0,$zero,$zero # save show times
32      add $s1,$zero,$zero # save length of string
33      la   $s2,putInString # save putinString address
34      la   $s3,dictionary   # save dictionary address
35      la   $s4,savePosition
36      add $s5,$zero,$zero # save length of replace
37      la   $s6,replaceString
38      addi$s7,$zero,10      # save '\n'
39      la   $s8,replaced      # save replaced
40
```

```
41      # obtain length of the input string
42      addi $t0,$s2,0
43      Strlen:
44          lb $t1,0($t0)   # attention ! use lb to obtain char
45          beq $t1,$s7,ExitStrlen
46          addi $t0,$t0,1
47          addi $s1,$s1,1
48          j Strlen
49      ExitStrlen:
50
51      # obtain begin and end position of each matching
52      addi $t1,$s2,0   # begining of input character
53      addi $t0,$s3,0   # begining of dictionary
54      addi $t7,$s4,0   # begining of position address
55
56      Find:
57          lb $t2,0($t0)
58          lb $t3,0($t1)
59
60          beq $t2,$zero,ExitFind   # end of dictionary
61          bne $t2,$t3,AddPosition # not equal at first place
62
63          addi $t4,$t0,1
64          addi $t5,$t1,1
65          addi $t6,$zero,1 # save length
66      CheckFullString:
67          beq $t6,$s1,AddCount
68          lb $t2,0($t4)
69          lb $t3,0($t5)
70          beq $t2,$s7,AddPosition
71          beq $t3,$s7,AddPosition # check boundary
72          bne $t2, $t3,AddPosition
73          addi $t4,$t4,1
74          addi $t5,$t5,1
75          addi $t6,$t6,1
76          j CheckFullString
77      AddCount:
78          addi $s0,$s0,1
79          sw $t0,($t7)
80          addi $t7,$t7,4
81      AddPosition:
82          addi $t0,$t0,1
83          j Find
84      ExitFind:
```

```
85
86
87      # output replace function
88      beq $s0,$zero,NotFound
89          la $a0,matchingfor # .asciiz"matching for "
90          li $v0,4
91          syscall
92          add $a0,$s0,$zero
93          li $v0,1
94          syscall # show findTimes
95          la $a0,times_n    #.asciiz" times\n"
96          li $v0,4
97          syscall
98          la $a0,replaceTips
99          li $v0,4
100           syscall # show replace tips
101           la $a0, replaceString
102           la $a1, putInSize
103           li $v0, 8
104           syscall # putin replace string
105
106           addi $t1,$s3,0 # begin of Dictionary
107           addi $t7,$s4,0 # begin of startposition queue
108           addi $t8,$s8,0 # begin of replaced string
109           ReplacePrint:
110               lb $t2,0($t1)
111               lw $t4,($t7)
112               beq $t2,$zero,ExitReplacePrint
113               bne $t1,$t4,printChar
114               # enter matching & replace area
115               addi $t0,$s6,0
116               printReplace:
117                   lb $t3,0($t0)   # attention ! use lb to obtain char
118                   beq $t3,$s7,ContinuePrint # '\n'
119                   add $a0,$zero,$t3
120                   sb $a0,0($t8)
121                   addi $t8,$t8,1
122                   addi $t0,$t0,1
123                   j printReplace
124               ContinuePrint:
125                   addi $t7,$t7,4   # turn to next start position
126                   add $t1,$t1,$s1 # add the length of string
127                   j ReplacePrint
128               # end of matching & replace area
```

```
129                 printChar:
130                     add $a0,$zero,$t2
131                     sb $a0,0($t8)
132                     addi $t8,$t8,1
133                     addi $t1,$t1,1
134                     j ReplacePrint
135             ExitReplacePrint:
136             la $a0, replaced
137             li $v0, 4
138             syscall         # show tips for putin string
139
140             j Exit
141         NotFound:
142             la $a0,FailedSign
143             li $v0,4
144             syscall
145         Exit:
146
147 .data
148         putInString: .space 200         # find string
149         replaceString:.space 200        # replace string
150         savePosition:.space 400         # savePosition for matching start and end
151         replaced:       .space 400      # replaced string
152
153         putInSize :    .word 100        # length of putinString
154
155         dictionary: .asciiz"helloworld myworld\n"   # matching string
156
157         putInTips:    .asciiz"String?:"     # tip for input searching string
158         replaceTips:.asciiz"Replace?:"    # tip for asking changing string
159         FailedSign: .asciiz"No match!\n" # if not find this,show failedSign
160         matchingfor:.asciiz"matching for "
161         times_n:      .asciiz" times\n"
```

直接输出替换后字符串，未保存到内存中函数

```
1               ReplacePrint:
2                   lb $t2,0($t1)
3                   lw $t4,($t7)
4                   beq $t2,$zero,ExitReplacePrint
5                   bne $t1,$t4,printChar
6                   addi $t0,$s6,0
7                   printReplace:
8                       lb $t3,0($t0)   # attention ! use lb to obtain char
9                       beq $t3,$s7,ContinuePrint
10                      add $a0,$zero,$t3
```

```
11                      li $v0,11
12                      syscall
13                      addi $t0,$t0,1
14                      j printReplace
15                  ContinuePrint:
16                      addi $t7,$t7,4    # turn to next start position
17                      add $t1,$t1,$s1 # add the length of string
18                      j ReplacePrint
19                  printChar:
20                      add $a0,$zero,$t2
21                      li $v0,11
22                      syscall
23                      addi $t1,$t1,1
24                      j ReplacePrint
25              ExitReplacePrint:
```

C++对应程序实验代码

```
 1 #include <iostream>
 2 #include <string>
 3 #include <queue>
 4 using namespace std;
 5
 6 int main()
 7 {
 8       // variable initialize
 9       string dictionary = "helloworld myworld\n";
10       string search, replace,replaced;
11       queue<int> begPosition;
12       int cnt = 0;
13
14       // help show
15       cout << dictionary ;
16       cout << "String?:";
17       // test string
18       cin >> search;
19       for(int idx = 0; idx < dictionary.size(); idx++)
20       {
21           if(dictionary[idx] == search[0]) // test whether idx matches
22           {
23               cnt++;
24               int putinIdx = 1;
25               for(int firstPos = idx+1; putinIdx < search.size(); )
26               {
```

```
27                      if(firstPos == dictionary.size() || dictionary[firstPos] != search[putinIdx])
28                      {
29                          cnt--;
30                          break;
31                      }
32                      firstPos++;
33                      putinIdx++;
34                  }
35                  if(putinIdx == search.size())
36                  {
37                      begPosition.push(idx);
38                  }
39              }
40          }
41          if(cnt)
42          {
43              cout << "matching for " << cnt << " times" << endl;
44              cout << "Replace?:";
45              cin >> replace;
46              for(int idx = 0; idx < dictionary.size() - 1; idx++)
47              {
48                  if(begPosition.size() && idx == begPosition.front())
49                  {
50                      replaced += replace;
51                      idx += search.size() - 1;
52                      begPosition.pop();
53                  }
54                  else
55                  {
56                      replaced += dictionary[idx];
57                  }
58              }
59              cout << replaced << endl;
60          }
61          else
62          {
63              cout << "No match!" << endl;
64          }
65          return 0;
66      }
```