# 今日头条 2017 春招笔试题——所有查询句子中与给定段落单词匹配量最多的句子

1. 题意：

给定一个英文段落（包含 n 个句子）和 m 次查询（每次查询给一个句子），求与段落中单词匹配量最多的查询。

*重要*：**1. 保证大小写不敏感；2. 不包含标点。**

段落示例：

```
"A bad beginning makes a bad ending",
"A fool may ask more questions in an hour than a wise man can answer in s
even years",
"A friend exaggerates a man virtue an enemy his crimes",
"A good head and an industrious hand are worth gold in any land",

"Always taking out of the meal and never putting in soon comes to the bot
tom"
```

查询示例：

```
"man of gold makes worth land seldom falls ending madness industrious",

"An enemy idle youth exaggerates his friend a needy age",

"bottom A poor man who taking a comes rich wife has never a ruler not a w
ife"
```

2. 解题思路：

由于是找单词匹配量，因此尝试将段落中单词全部存为一个 map 的 key(s)，这样，在计算查询匹配量时，只需遍历其单词，并记录下成功匹配的次数即可。

*（由于没看清上面“重要”提示，此版本适用包含标点符号的场景）*

3. C++代码：

```
#include <iostream>
```

```
#include <string>
#include <string.h>
#include <map>

#define GetStrArrLen(strArr) (sizeof(strArr)/sizeof(strArr[0]))

#define TEST

using namespace std;

const int WORDLEN= 48;

bool IsAlpha(char a){
    return ( (a>='A' && a<='Z') || (a>='a' && a<='z') )? true : false;
}

void GetMatchList(map<string, string> &wordsList, string query[], size_t
 queryLen, map<size_t, int> &matchList){
    size_t strs_len = queryLen;
    char word[WORDLEN];
    for(size_t idx = 0; idx < strs_len;
        idx++){ matchList.insert(pair<size_t,
        int>(idx, 0)); size_t j = 0;
        size_t str_len = query[idx].length();
        int widx = 0;
        memset(word, 0, WORDLEN);
        while(j < str_len){
            if(IsAlpha(query[idx][j])){ word[
                widx++] = query[idx][j]; j++;
                continue;
            }
            if(0 != strlen(word)){
                if(wordsList.end() != wordsList.find(word))
                    matchList[idx]++
                }
            }

            widx = 0;
            memset(word, 0, WORDLEN);
            j++;
        }
```

```
        if(0 != strlen(word)){
                matchList[idx]++;
            }
        }
    }
}
void ExtractWords(string paragraph[], size_t paraLen, map<string,strin
g> &wordsList){
    size_t strs_len = paraLen;
    char word[WORDLEN];
    memset(word, 0, WORDLEN);
    for(size_t idx = 0; idx < strs_len;
        idx++){ size_t j = 0;
        size_t str_len = paragraph[idx].length();
        int widx = 0;
        memset(word, 0, WORDLEN);
        while(j < str_len){
            if(IsAlpha(paragraph[idx][j])){ word[wid
                x++] = paragraph[idx][j]; j++;
                continue;
            }
            if(0 != strlen(word)){
                if(wordsList.end() ==
                    wordsList.find(word)){ wordsList.insert(pair<string,
                    string>(word, word));
                }
            }

            memset(word, 0, WORDLEN);
            widx = 0;
            j++;
        }

        if(0 != strlen(word)){
            if(wordsList.end() ==
                wordsList.find(word)){ wordsList.insert(pair<string,
                string>(word, word));
            }
        }
    }
}
```

```
void ToLower(string strings[], size_t
    strsLen){ size_t strs_len = strsLen;
    for(size_t idx = 0; idx < strs_len;
        idx++){ size_t j = 0;
        size_t str_len = strings[idx].length();
        while(j < str_len){
            if (strings[idx][j] >= 'A' && strings[idx][j] <=
                'Z'){ strings[idx][j] += 32;
            }
            j++;
        }
    }

void Print(string strings[], size_t strsLen, string infoName){
    cout<< "****************"<< infoName << "********************" << en
dl;
    size_t strs_len = strsLen;
    for(size_t idx = 0; idx < strs_len;
        idx++){ cout << strings[idx] << endl;
    }
    cout<< "******************************************" << endl;
    cout << endl;
}

void PrintMap(map<string, string> wordsList){
    map<string, string>::iterator iter = wordsList.begin();
    int count = 0;
    for(; iter != wordsList.end();
        iter++ ){ cout << iter->second <<
        '\t' ;
        if (0 == ++count %
            10){ cout << endl;
        }
    }

    cout << endl;
}

int main()
{
    size_t strArrLen = 0;
    string paragraph[] = {
```

```
        "A bad beginning makes a bad ending",
        "A fool may ask more questions in an hour than a wise man can ans
wer in seven years",
        "A friend exaggerates a man virtue an enemy his crimes",
        "A good head and an industrious hand are worth gold in any land",
        "Always taking out of the meal and never putting in soon comes to
 the bottom"
    };
    strArrLen = GetStrArrLen(paragraph);
    #ifdef TEST
        Print(paragraph, strArrLen, "Paragraph");
    #endif

        ToLower(paragraph, strArrLen);

    #ifdef TEST
        Print(paragraph, strArrLen, "Paragraph");
    #endif

    map<string, string> wordsList;
    ExtractWords(paragraph, strArrLen, wordsList);

    string query[] = {
            "man of gold makes worth land seldom falls ending madness ind
ustrious",
            "An enemy idle youth exaggerates his friend a needy age",
            "bottom A poor man who taking a comes rich wife has never a ru
ler not a wife"
    };
    #ifdef TEST
        Print(query, strArrLen, "Query");
    #endif

    ToLower(query, strArrLen);

    #ifdef TEST
        Print(query, strArrLen, "Query");

    #ifdef TEST
        PrintMap(wordsList);
    #endif
```

```
    map<size_t, int> matchList;
    GetMatchList(wordsList, query, strArrLen, matchList);

    map<size_t, int>::iterator start = matchList.begin();
    size_t ridx = 0;
    int maxCount = 0;
    for(; start != matchList.end();
        start++){ if (maxCount < start-
        >second){
            maxCount = start->second;
        }

    cout << maxCount << "Matched" << endl;
    cout << query[ridx] << endl;
    return 0;
}
```