**项目要求**

1. 对源文件（\*.txt,\*.cpp,\*.h,\*.cs,\*.html,\*.js,\*.java,\*.py,\*.php等，文件夹内的所有文件）统计字符数、单词数、行数、词频，统计结果以指定格式输出到默认文件中，以及其他扩展功能，并能够快速地处理多个文件。

2. 使用性能测试工具进行分析，找到性能的瓶颈并改进

**编程思路**

  首先，统计字符是基本的文件操作，我选择每次读取一个字符，这也便于随后操作。单词数的主要问题是对单词的判断，因为题目还给出了各种条件，而单词的数量又相当多，所以代码要写的简介，尽量不要调用函数。行数可以通过读取的换行符数目来判断。统计词频的难点在于找到合适的结构存储，插入和查找要快，也要便于后续排序。词频的统计，涉及到两个词的关联，要选择合适的数据结构存储。一开始根据我在网上查找的资料，字典树似乎是对于词频统计应用的一个比较好的选择，所以我首先尝试自己实现了一个字典树的结构，并成功存储单词，但后续发现字典树在读取和排序方面并无优势，而又很难表示出词组的结构。那么另一种方法就是hash存储，可以有很快的存储读取速度，同时也方便表示词组。查阅资料，C++有unordered\_map的标准模板库，内部是通过hsah函数存储的，考虑到自己写的话不一定能找到合适的hash，正确使用下，STL不比自己实现要慢，所以选择使用unordered\_map。那么接下来的问题是Key和Value分别是什么。

定义如下结构



**实现结构**

一.总体结构

 1.通过命令行读取参数，遍历文件夹中的文件

2.记录字符数，行数，单词数，判断单词，存储单词和词组

3.排序获得频率前十的单词，词组，输出文件

二.实现函数



三.具体代码

addWord是最主要也是占用时间最长的函数，代码如下

复制代码

void addWord(string &word, string &word\_pre, string &word\_r, string &word\_pre\_r)

{

int wordlen = word.length() - 1;

int pfixlen = 0;

string postfix, phraseKey;

wMap::iterator tempit;

for (int i = wordlen;; i--)

{

if (word[i]<'0' || word[i]>'9')

{

pfixlen = wordlen - i;

//获得数字后缀长度

break;

}

}

postfix = word.substr(wordlen - pfixlen);

word = word.substr(0, wordlen - pfixlen);//从小写的word得到wordKey

wordsDic[word].appearNum++;//生成单词map

if (wordsDic[word].value.empty() || wordsDic[word].value>word\_r)//选择字典序最小的字符串记录

wordsDic[word].value = word\_r;//记录真实值

if (!word.empty() && !word\_pre.empty())

{

phraseKey = word\_pre;

phraseKey.push\_back('-');

phraseKey += word;//得到phraseKey

phraseDic[phraseKey].appearNum++;//生成词组Map

if (phraseDic[phraseKey].Aword.empty())//选择字典序最小的短语记录

{

phraseDic[phraseKey].Aword = word\_pre\_r; phraseDic[phraseKey].Bword = word\_r;

}

else if (phraseDic[phraseKey].Bword > word\_r)

phraseDic[phraseKey].Bword = word\_r;

else if (phraseDic[phraseKey].Aword > word\_pre\_r)

phraseDic[phraseKey].Aword = word\_pre\_r;

}

}

复制代码

样例输出结果

