University of St.Gallen

School of Management,
Economics, Law, Social Sciences,
International Affairs and Computer Science

DATABASE OF FILES

7,789 | 8,789: Skills: Programming with Advanced Computer Languages

Simon ÜNES (17-455-122)
Alissa MAZZEI (18-452-680)
Christian ALBERTANO (18-605-279)
Martina GAVIRAGHI (18-610-170)

Under the supervision of

Dr. Mario SILIC

Submitted on 22nd December 2022

Project Abstract

The programme we decided to develop is called **dbfiles.cc** and was written in C++.

The decision to write the programme in C++ was dictated by the fact that we wish to expand our knowledge of this language as we believe that well-written C++ programmes are fast and efficient. The language is more flexible than other languages: it can function at higher levels of abstraction. C++ provides highly optimised standard libraries. It allows access to low-level hardware functionality to optimise speed and minimise memory requirements. In addition, C++ can create almost any type of programme: games, device drivers, HPC, cloud, desktop, embedded and mobile apps and much more. Libraries and compilers for other programming languages are also written in C++. Finally, C++ is a compiled language, which means that the compiler translates the C++ source code into an executable file containing a set of machine instructions.

But specifically, our programme has the purpose of reading several text files and generating a reverse index of the words in all the files. In this way, in a subsequent invocation, the programme will be able to search the files by keywords. It may be operated with the following three sets of command line options:

- **dbfiles index *filename*1 *[filename*2 . . .*]***

In this case, an *indexing function* is executed by the programme. The programme reads the specified files and creates an index in a file called "INDEX". The index is a reverse word index, which basically means that it maps any word to a list of files. A *word* can be defined as a maximum contiguous sequence of alphabetic characters adjacent to any non-alphabetic character (either at the end or at the beginning of the file). In indexing and searching, only words between 3 and 30 characters in length are addressed. The index must be compact in such a way that the list of files per word is stored as a list of short numerical file identifiers.

- **dbfiles search *keyword*1 *[keyword*2 . . .*]***

In this case, the *search* function is run by the programme. The programme returns a list of all the files (file names) that contain all the keywords passed on the command line after reading the index from a file called INDEX. It's crucial to mention that the keyword match should be case-insensitive. No results are output if no indexed file contains all the indicated keywords. The original order passed to the indexing function is followed when printing the names of the matching files. The search function only read the index file. This means that the search function works even if the files are not available at the time the search function is invoked.

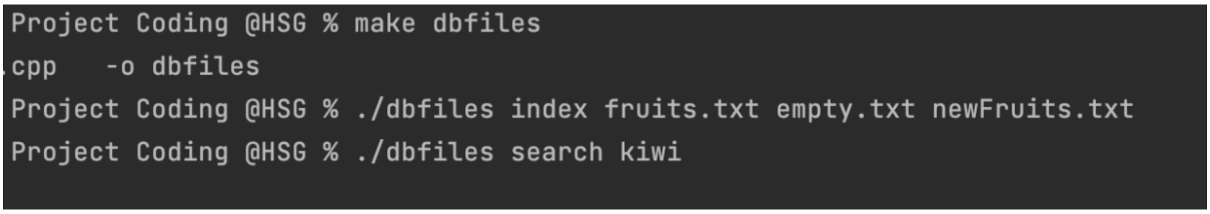
- **docdb search -p *keyword1 [keyword2 . . .]***

Ultimately, in this last case, the programme performs the search function as described before, but instead of sorting the output files in the original indexing order, it produces the matching files in descending order with respect to the total number of matching words. The search function gives priority to the output based on the strength of a match.

Usage

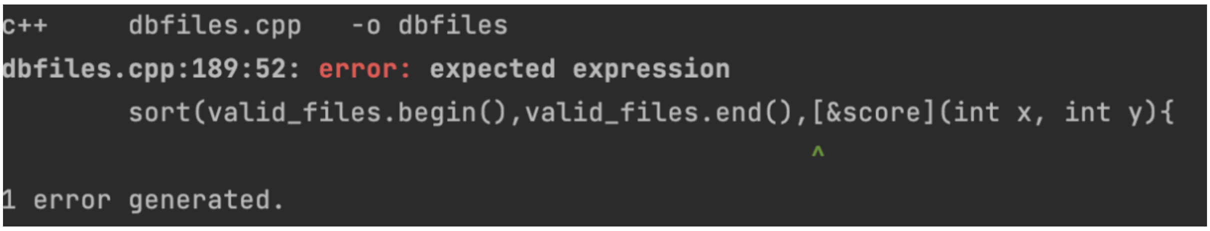
Compile file: **make dbfiles**

We create a MakeFile to compile our programme and then execute it.



```
Project Coding @HSG % make dbfiles
cpp   -o dbfiles
Project Coding @HSG % ./dbfiles index fruits.txt empty.txt newFruits.txt
Project Coding @HSG % ./dbfiles search kiwi
```

The MakeFile avoids an 'error' at line 189:52, because we used a lambda expression so that the programme can be faster, and the complexity can be reduced. Everything works correctly. The lambda expression is not recognised if you compile it with gcc or g++ (an expression is expected).



```
c++     dbfiles.cpp   -o dbfiles
dbfiles.cpp:189:52: error: expected expression
        sort(valid_files.begin(),valid_files.end(),[&score](int x, int y){
                                                   ^

1 error generated.
```

Execute file example:

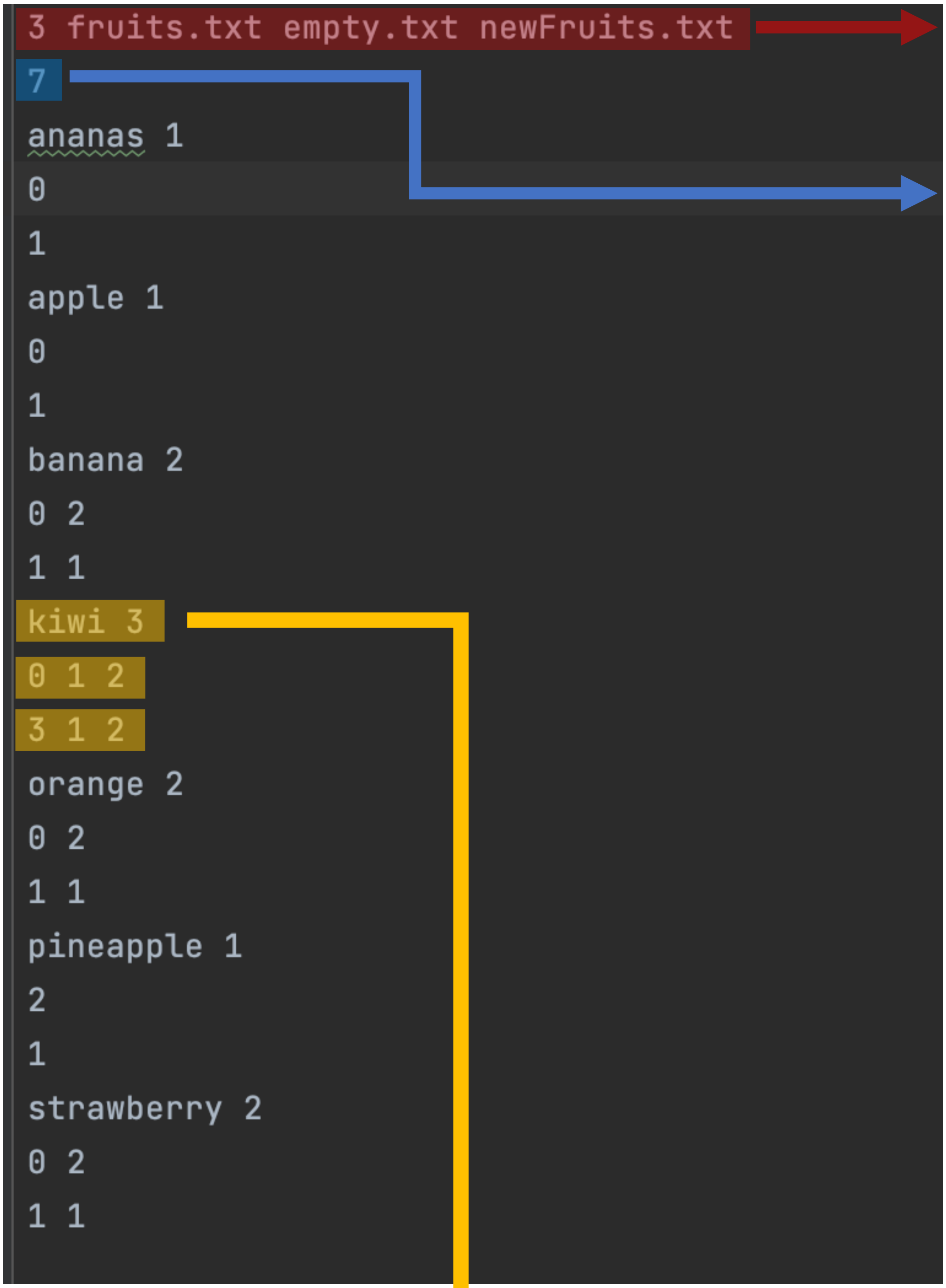
First feature

You can pass the desired number of files

./dbfiles index fruits.txt empty.txt newFruits.txt

By using the 3 txt files we created, an INDEX file will be generated (see image below).

We will explain the meaning of each digit in detail (see comments).



```
3 fruits.txt empty.txt newFruits.txt
7
ananas 1
0
1
apple 1
0
1
banana 2
0 2
1 1
kiwi 3
0 1 2
3 1 2
orange 2
0 2
1 1
pineapple 1
2
1
strawberry 2
0 2
1 1
```

We passed 3 files txt: fruits.txt, empty.txt and newFruits.txt

In these files, there are 7 unique words. Note: Repetition is calculated as a single word.

Example of word “kiwi”

Next to kiwi is the number 3, which means that the word kiwi occurs at least once in each of these files.

Below, the occurrence of the word kiwi in each file is shown (based on 0 counting positioning in python).

In file 0 (fruits.txt) kiwi occurs 3 times.

In file 1 (empty.txt) kiwi occurs 1 time.

In file 2 (newFruits.txt) kiwi appears 2 times.

Everything explained above is the same for each word.

Second feature

./dbfiles search "kiwi"

It indicates in which file the word kiwi is present.

Outputs:

- **fruits.txt**
- **empty.txt**
- **newFruits.txt**

It is also possible to search by combining two keywords and it will be displayed in which files the two words both occur (AND).

./dbfiles search "kiwi" and "orange"

In our case, both are present in:

- **fruits.txt**
- **newFruits.txt**

empty.txt is not an output since the word orange is not included in that file.

Third feature

./dbfiles search -p "kiwi"

It outputs the matching files in decreasing order of the total count of matching words.

Output:

fruits.txt (it contains 3 occurrences)

newFruits.txt (it contains 2 occurrences)

empty.txt (it contains 1 occurrence)

Note: you can pass as many files as you want, with as many words as you want, even a book as a txt file. This is possible because we used vectors to automatically expand. We have created four txt files. But it is also possible to create more txt. files and pass them as argc.

The whole code is commented.

Group members

Ünes Simon (Coding@HSG username: SimonUnes)

Mazzei Alissa (Coding@HSG username: AlissaMazzei)

Albertano Christian (Coding@HSG username: allegriout)

Gaviraghi Martina (Coding@HSG username: MartiG)