

# CS2002 Practical 3: Binary Trees

130016030  
*University of St Andrews*  
March 6, 2015

## Introduction

The aim of this project is to develop a data structure called binary tree and a program, which allows the user to specify file, and immediately display the how many times every word occurs, in alphabetical order on the screen. As an extension the application should also display the average word length, total number of words and the most common word. It should also accept any length of word.

## Running instructions

There is a Makefile that generates 2 executions:

1. *tests* : it tests the binary search tree implementation by assertions.
2. *wordcount* : displays information that was discribed in introduction section.

To build this project type: `make -f Makefile.in`

```
test@8afbe7d7 ~/Desktop/Practical-C2$ make -f Makefile.in
clang -Wall -Wextra -O1 -O2 -O3 -std=c99 treenode.o test.o -o tests
clang -Wall -Wextra -O1 -O2 -O3 -std=c99 treenode.o wordcount.o -o wordcount
```

# Design and implementation

## 1 treenode.h

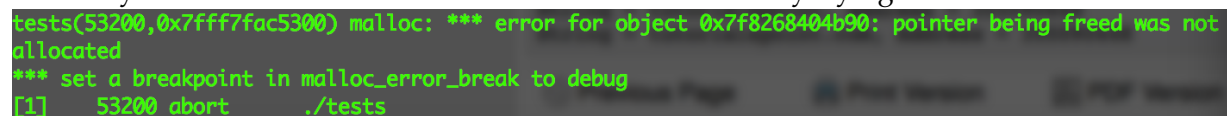
This file contains declarations for the binary tree. This was provided by Chris Jefferson and extended by methods, namely: newNode, printers, comparators, recursive insert, query and printTree. newNode creates a new node with a given value. This method is used by insert function. Custom printers and comparators were added because the value has a type void\*.

## 2 treenode.c

It contains definitions for declarations in treenode.h. This file contains recursive and iterative method to look up the value from the tree. However, insert, free tree, print tree are using only recursive methods. They all could use iterative methods, but due to lack of time they have not been implemented. Nevertheless, those tree operations were still implemented as it was required in the specification.

## 3 test.h and test.c

Header file contains only declarations for test methods. test.c file tests the binary search tree implementation from treenode.h. All of those tests based on the assert method that checks similarity of two values. The method freeTree was checked by trying to double clear it.



```
tests(53200,0x7fff7fac5300) malloc: *** error for object 0x7f8268404b90: pointer being freed was not allocated
*** set a breakpoint in malloc_error_break to debug
[1] 53200 abort ./tests
```

Print tree was checked by printing it in every test. Other tests are straightforward, such as: construct a binary search tree, perform some meaningful methods and check that the output is the same. Testing is described in the same name section.

## 4 wordcount.h and wordcount.c

Header file contains declarations for wordcount.c. It uses the binary search tree from treenode. First of all, it reads the file word by word that is specified by user. Then, every word character has to be valid that means do not contain any symbols apart from apostrophe. If it contains symbol, then it deletes it and concatenates the word. After that, all characters are converted to lower case letters. Then, it adds those values to the tree, print nodes and word statistics.

It has been found that before those words are going to be added to the tree, they have to be put in the new variable. This is done by malloc and memcpy. The size for a new variable has to be slightly bigger, so the program can parse pride.txt file. This is done by adding a constant value to the size.

As an extension the program also displays the average word length, total number of words and the most common word. On top of that, the size of a buffer is allocated dynamically.

## Testing

```
test@8afbe7d7 ~/Desktop/Practical-C2$ ./wordcount
Enter the name of file you wish to see
> tiny.txt
(a,2)
(and,3)
(cat,2)
(cat's,1)
(cats,1)
(do,2)
(dog,2)
(dog's,1)
(dogs,1)
(not,1)
(said,1)
(speak,1)
(the,2)
(together,1)
(work,1)

The most common word "and" occurs 3 times
Total number of characters in all words: 75
Average word length: 3
Total number of words: 22%
```

Similarly to the example provided in the project specification, output is as same as expected. This example demonstrates the output for tiny.txt.

```
test@8afbe7d7 ~/Desktop/Practical-C2$ ./tests
Start test #1
0 4 5 6
-1 4 5 6
Test #1 finished without errors

Start test #2

Test #2 finished without errors

Start test #3
-8 0 3 6
Test #3 finished without errors

Start test #4
-7 0 4 9
Test #4 finished without errors

Start test #5
-7 -1 0 70
Test #5 finished without errors

Start test #6
Test #6 finished without errors
```

This is the output of the test file. It shows that the binary search tree was implemented correctly.

```
test@8afbe7d7 ~/Desktop/Practical-C2$ ./wordcount
Enter the name of file you wish to see
> none
Error while opening the file.
: No such file or directory
```

This error message shows that there is no such file in current directory.

```

(yielding,2)
(yieldingcertainly,1)
(york,1)
(you,1416)
(you'll,1)
(youand,2)
(yoube,1)
(youbut,2)
(youhad,1)
(youhow,1)
(your,1)
(young,130)
(younge,4)
(younger,30)
(youngest,13)
(your,464)
(yours,19)
(yourself,49)
(yourselfand,1)
(yourselves,2)
(youth,9)
(youths,1)
(zip,1)

The most common word "the" occurs 4496 times
Total number of characters in all words: 552442
Average word length: 4
Total number of words: 124444%

```

The is the output of the pride.txt. As you can see, it has some incorrect words. For example, yieldingcertainly. However, it can be seen that in pride.txt yieldingcertainly is written as yielding–certainly. This was my previous solution when program separated the word by spaces.

```

(would,471)
(wound,2)
(wounded,3)
(wounding,2)
(wretched,10)
(wretchedly,1)
(wretchedness,2)
(write,39)
(writer,3)
(writes,1)
(writing,16)
(written,22)
(wrong,15)
(wrote,21)
(www,6)
(yards,1)
(yawn,2)
(yawned,1)
(yawning,1)
(year,29)
(years,34)
(years',2)
(yes,75)
(yesterday,13)
(yet,73)
(yield,4)
(yielded,2)
(yielding,3)
(york,1)
(you,1426)
(you'll,1)
(young,130)
(younge,4)
(younger,30)
(youngest,13)
(your,465)
(yours,23)
(yourself,50)
(yourselves,2)
(youth,9)
(youths,1)
(zip,1)

The most common word "the" occurs 4507 times
Total number of characters in all words: 552442
Average word length: 4
Total number of words: 125234%
test@8afbe7d7 ~/Desktop/Practical-C2$

```

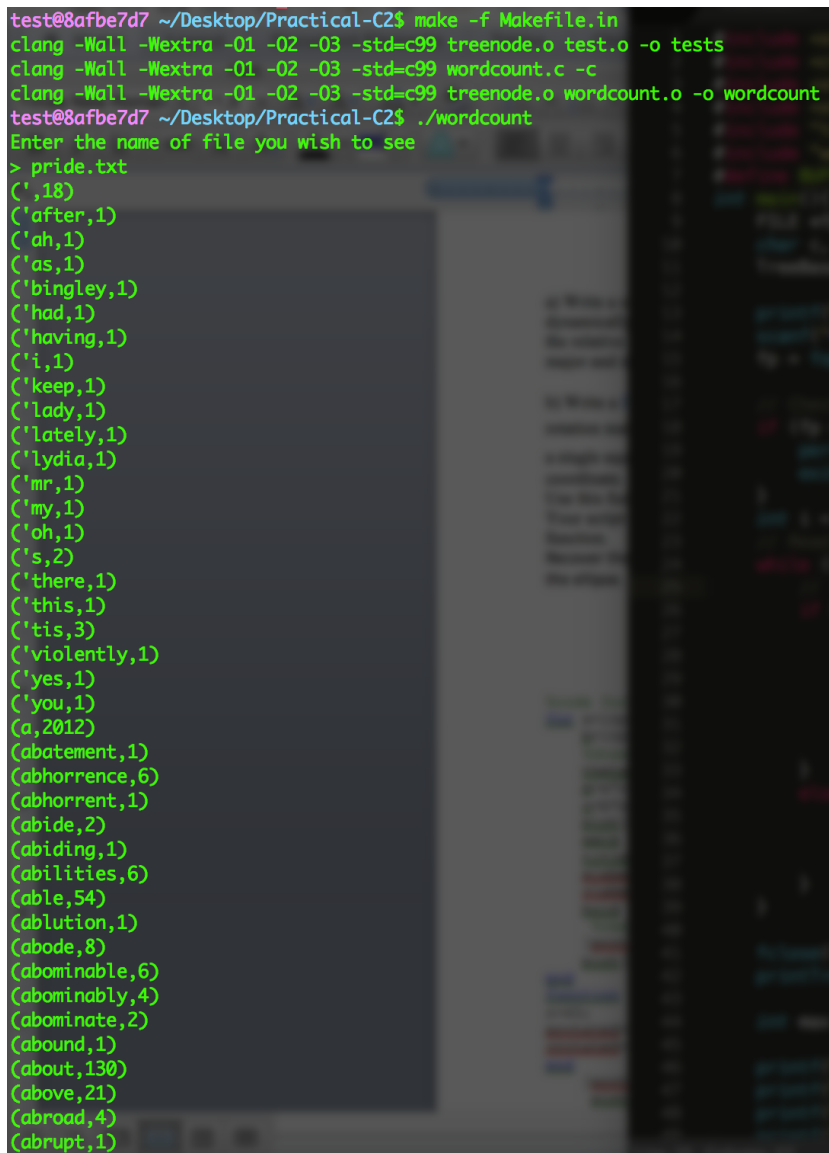
This is a new solution, where program reads the whole word by characters and then adds it to the word. In this case, the output is more valid than in the previous approach.

## Conclusion

To conclude, through the development of this binary search tree, word count application and the implementation of features with different difficulties, helped for better understanding of a C language in general and pointers especially. However, there are several drawbacks with this solution.

One of them is in order to find the most common word in file, the program has to traverse the whole tree to find maximum count variable and then traverse it again to find respected word. This could be done in one traversal and hence improve this program. Nevertheless, this solution still prints the correct common word.

Another disadvantage of the program is the word definition. as you can see from the pride.txt output, the program accepts all apostrophes. This was a requirement from specification where cats and cat's has to be different. Making a better word definition could definitely benefit this application.



```
test@8afbe7d7 ~/Desktop/Practical-C2$ make -f Makefile.in
clang -Wall -Wextra -O1 -O2 -O3 -std=c99 treenode.o test.o -o tests
clang -Wall -Wextra -O1 -O2 -O3 -std=c99 wordcount.c -c
clang -Wall -Wextra -O1 -O2 -O3 -std=c99 treenode.o wordcount.o -o wordcount
test@8afbe7d7 ~/Desktop/Practical-C2$ ./wordcount
Enter the name of file you wish to see
> pride.txt
('',18)
('after,1)
('ah,1)
('as,1)
('bingley,1)
('had,1)
('having,1)
('i,1)
('keep,1)
('lady,1)
('lately,1)
('lydia,1)
('mr,1)
('my,1)
('oh,1)
('s,2)
('there,1)
('this,1)
('tis,3)
('violently,1)
('yes,1)
('you,1)
(a,2012)
(abatement,1)
(abhorrence,6)
(abhorrent,1)
(abide,2)
(abiding,1)
(abilities,6)
(able,54)
(ablution,1)
(abode,8)
(abominable,6)
(abominably,4)
(abominate,2)
(abound,1)
(about,130)
(above,21)
(abroad,4)
(abrupt,1)
```