Computer Programming

Final Assignment for 2014:

a C program for text processing

Due date: 2015-01-05

Instructions

1. Read this document carefully.
   Understand what the requirements are. There are three parts: everyone should do Part 1 and Part 2. Part 3 is optional.

2. Work on finishing Part 1 first. Only after that is working, then you can try and work on Part 2 (and then optional Part 3).

3. Finally, please submit 1 (only **ONE**) working C program for marking:

Send:	your final .c files only
Email subject:	“Final assignment for ID*, NAME*”
Deadline:	2015-01-05, 23.59
Email to:	ivm@ustc.edu.cn

* replace with your student ID and name

Requirement (Part 1)

- A standard C program that can compile and run under Ubuntu Linux.
- The program should read in a text file.
- It must then compute and print the following information:
  1. The Flesch Reading Ease Score
  2. The Gunning Fog Index
  3. The 10 'hardest' words in the text
  4. The 10 most frequent 'hard words' found in the text
  5. The total number of words in the text, and the total number of unique words

I will explain each of these on the next few pages.

Explanation (Part 1)

1. The Flesch Reading Ease Score and
2. The Gunning Fog Index are two ways of measuring the complexity of writing.

The Flesch score, F, is defined as:

$$F = 206.835 - \left(1.015 \times N_w / N_s\right) - \left(74.6 \times N_x / N_w\right)$$

The Gunning Fog index, G, is defined as:

$$G = 0.4 \times \left\{\left(N_w / N_s\right) + 100 \times \left(N_c / N_w\right)\right\}$$

N_x=number of syllables
N_w=number of words
N_s=number of sentences
N_c=number of complex words (words with > 2 syllables)

Explanation (Part 1) (continued)

3. The 'hardest' words are defined as having the most syllables. You print out the 10 words with most syllables.
4. The 10 most frequent 'hard words' means you count how many times each word with >2 syllables appears in the text, then you print out the 10 hard words that appear most.
5. The total number of words is just N_w. The total number of unique words means the number of different words in the text.

Let us see an example to calculate F and G...

Example (Part 1)

"Mary had a little lamb whose fleece was white as snow. Everywhere that Mary went that lamb was sure to go."

Word	Syllables	Count	Word	Syllables	Count	Word	Syllables	Count
Mary	2	2	fleece	1	1	that	1	2
had	1	1	was	1	2	went	1	1
a	1	1	white	1	1	sure	1	1
little	1	1	as	1	1	to	1	1
lamb	1	2	snow	1	1	go	1	1
whose	1	1	Everywhere	4	1			

N_x=number of syllables = 26
N_w=number of words = 21
N_s=number of sentences = 2
N_c=number of complex words (words with > 2 syllables) = 1
N_c=Number of unique words = 17
Sentence 1 = 11 words, sentence 2 = 10 words.

Example (Part 1) (continued)

N_x=number of syllables = 26
N_w=number of words = 21
N_s=number of sentences = 2
N_c=number of complex words (words with > 2 syllables) = 1
N_c=Number of unique words = 17

$$F = 206.835 - \left(1.015 \times N_w / N_s\right) - \left(74.6 \times N_x / N_w\right)$$

*F=206.835-(1.015*21/2)-(74.6*26/21)=103.82*

$$G = 0.4 \times \left\{\left(N_w / N_s\right) + 100 \times \left(N_c / N_w\right)\right\}$$

G=0.4((21/2)+100*(1/21))=6.1*

F Score	Meaning
90.0–100.0	easily understood by an average 11-year-old student
60.0–70.0	easily understood by 13- to 15-year-old students
0.0–30.0	best understood by university graduates

G Score	Meaning
0-8	Every English reader can understand it
8-12	Most people can understand it
> 12	For a specialised audience

For this analysis, we need to have a way of counting syllables. Because this might be difficult, I will give you a function you can use to count syllables (it's based on something I downloaded):

```
#include <stdio.h>
#include <string.h>
#define false 0
#define true 1

int count_syllables(char *word)
{
        char vowels[] = {'a', 'e', 'i', 'o', 'u', 'y'};
        int lastWasVowel = false, foundVowel = false;
        int numVowels = 0;
        int i,j;
        char c,v;
        int len=strlen(word);
        for(i=0;i<len;i++) /*loop through each character in the word*/
        {
            c=word[i];
            foundVowel = false;
            for(j=0;j<6;j++) /*run through the vowels*/
            {
                v=vowels[j];
                if ((v == c) && lastWasVowel) /*ignore dipthongs (double vowels)*/
                {
                    foundVowel = true;
                    lastWasVowel = true;
                    break;
                }
                else if ((v == c) && !lastWasVowel)
                {
                    numVowels++;
                    foundVowel = true;
                    lastWasVowel = true;
                    break;
                }
            }
```

Continued on next page...

```
            if (!foundVowel) /*no vowel found*/
                lastWasVowel = false;
        }

        /* remove es at the end of the word, because it's often silent*/
        if ((numVowels>1) && (len > 2) && strcmp(&word[len-2],"es")==0)
            numVowels--;
        /* remove a silent e at the end of a word*/
        else if ((numVowels>1) && (len > 1) && word[len-1]=='e')
            numVowels--;

        return numVowels;
}

/* this main function is so you can test the syllable counter*/
int main(int argc, char *argv[])
{
     if (argc<2)
          printf("Insufficient command line args\n");
     else
          printf("We got %d syllables from [%s]\n",count_syllables(argv[1]),argv[1]);
}
```

```
$ ./syllablecount January
We got 3 syllables from [January]
$ ./syllablecount often
We got 2 syllables from [often]
$ ./syllablecount March
We got 1 syllables from [March]
$ ./syllablecount July
We got 2 syllables from [July]
$ ./syllablecount China
We got 2 syllables from [China]
```

Note: the code makes some mistakes but it is good enough to use.

<-correct

incorrect->

```
$ ./syllablecount absolutely
We got 5 syllables from [absolutely]
$ ./syllablecount USTC
We got 1 syllables from [USTC]
$ ./syllablecount everywhere
We got 4 syllables from [everywhere]
```

Warnings (Part 1)

Real text is full of punctuation symbols. You need to remember this when separating words and sentences. Here are some examples:

Cro-magnon	two words
brute-force	two words

(for this lab, consider a hyphen to be the same as a space)

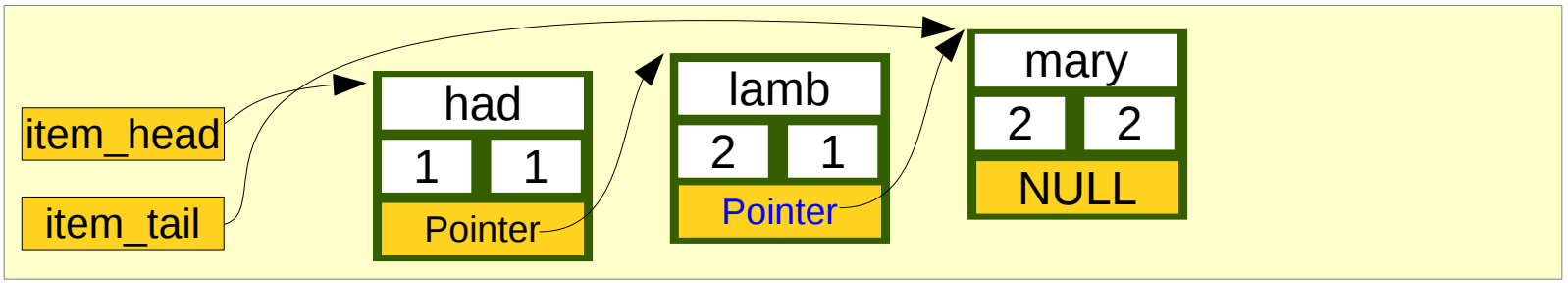
He wasn't happy	three words
Are you? Not me	four words, two sentences
Today, tomorrow	two words, one sentence
And (a “quote”)	three words, one sentence
i.e. one, two, three	four/five words, one/two sentences
as follows: him, her	four words, two sentences

Don't forget that a '\n' doesn't mean anything in English. It should **not** be counted as ending a sentence!

Method (Part 1)

Please do these steps in the sequence shown

1. Try, test, modify and understand the example linked list code (ll.c).
2. Change the code so that each record in the list contains a string (pointer to a character array) and an integer. You are going to use each record to hold one English word, a count of how many times the word occurs, and a syllable count.
3. You need to scan through the text file and extract every word from the text. For each word, see if it already exists in your linked list. If it does exist, increment the counter for that word. If it doesn't exist, then add it to your list. Always have one record per new word.
4. You should probably make sure that the list is always stored in alphabetical (or reverse-alphabetical) order.



Method (Part 1) (continued)

Notes:

1. A good solution is efficient.
2. Make sure all of the words are converted to lower case before you add them to your list.
3. Test your code with some short examples. Include some difficult punctuation and other 'problems'. You might need to try a lot of examples before your code works 100%.

Assignment (Part 2)

Only do this after the Part 1 is working

Your program needs to print out some MORE information: I want to know what is the most common sentence beginning (i.e. what words are found most often at the start of the sentence). I want to know the most frequent 2, 3, 4 and 5 words.

This is how you should present the information:

```
$ ./my_assignment charles_dickens.txt

Analysing the file “charles_dickens.txt”

The 5 analysis items for Part 1 are printed here.
The new information is printed below...

The most frequent sentence beginnings are:
[2 words, used in 9.41% of sentences]:  “Now then”
[3 words, used in 2.13% of sentences]:  “After all had”
[4 words, used in 0.49% of sentences]:  “It was then that”
[5 words, used in 0.02% of sentences]:  “Apart from the obvious fact”
```

Assignment (Part 3)

This is OPTIONAL.
Only do this after Part 1 & Part 2 are working
You can get extra marks if you do <u>one</u> of these in a <u>separate</u> program (so you should send me <u>two</u> .c programs)

1. Plot a graph of word distributions in PGM format. For example, create a bar chart of the percentage of words beginning with a–z. *You don't need to include the letters a–z in the image.*

2. Create a word tree. This is like a dictionary, but tells you the usual "next word". Only do it for words > 3 letters in length (and only print out the most common "next words", not all "next words").
Here is an example if we ask the program about the word 'brown':

- "brown" was used 153 times (0.6%), it is the 89th most popular word.
"brown" was followed by the following words (only listing words > 1%):

Followed by	Percentage
dog	2.1%
eyes	1.2%
out	1.1%

It means "dog" was used 2.1% of times after the word "brown" was used

Final Information

- Don't work together. You can share ideas, but not code.
- Look on my website. The file `assignment.zip` contains some useful resources and code you can use.
- If you have any problem completing the assignment, tell me or the teaching assistants earlier.