Imperial College
London

Software Engineering 1: Introduction to Computing

5 – Text and files (part b: output to text files and more on strings)

Max Cattafi (m.cattafi@imperial.ac.uk)

In previous examples we have seen how to read from text files, the following program illustrates how to create text files and use them for output.

The program is also an example of operations on text input considering said input word by word and of how we can use the `==` operator to compare two strings: we have seen instances of this in past examples, however it is a reminder that in C++ the `==` operator works for strings in the same way as, for instance, for `int` variables.

The program reads in input the names of two text files and then operates on their content. One of the files is assumed to be some text which is then read word by word and stored in a vector of strings. For instance let's assume file `text.txt` with the following content:

```
Call me Ishmael. Some years ago, never mind how long
precisely, having little or no money in my purse, and
nothing particular to interest me on shore, I thought
I would sail about a little and see the watery part of
the world. It is a way I have of driving off the spleen
and regulating the circulation. Whenever I find myself
growing grim about the mouth; whenever it is a damp,
drizzly November in my soul;
```

The other input text file is assumed to contain a few words of which we would like to count the number of occurrences found in the previously mentioned text. For instance let’s assume file `words.txt` with the following content:

```
the
I
in
```

What we would like as an outcome of our program is an output file with name `outwords.txt` (i.e. ‘out’ + the name of the second input file) and the following content:

```
the: 5
I: 4
in: 2
```

The following program is a possible implementation:

```cpp
#include <iostream>
#include <string>
#include <fstream>
#include <cstdlib>
#include <vector>

int count_occurrences(std::vector<std::string> words, std::string w);

int main(){

    std::string textfilename, wordsfilename;

    std::cout << "please enter the names of the input files:" << std::endl;

    std::cin >> textfilename >> wordsfilename;

    std::ifstream infile;
    infile.open(textfilename.c_str());

    if(!infile.is_open()){
        std::cout << "error opening " << textfilename << std::endl;
        exit(EXIT_FAILURE);
    }

    std::vector<std::string> words;
    std::string word;

    while(infile >> word){
        words.push_back(word);
    }

    infile.close();

    // this concludes the operation of reading from the first text file
    // and storing the content word by word in a vector of strings
    // (in this version of the program there will be issues due to
    // punctuation, for instance "shore" will be stored as "shore,"
    // but it's ok for our current purposes)

    // we can reuse the infile object to read from a different file
    // (notice that first it has been closed above)
    infile.open(wordsfilename.c_str());

    if(!infile.is_open()){
        std::cout << "error opening " << wordsfilename << std::endl;
        exit(EXIT_FAILURE);
    }


    // we declare an object of type std::ofstream
    // representing an output text file (we call it
    // outfile, it could be called in some other way)
    std::ofstream outfile;

    // we create the name for the output file, notice
    // the use of + to concatenate two strings
    std::string outfilename = "out" + wordsfilename;

    // this operation creates a new file;
    // note that if a file of the same name
    // already exists it is overwritten!
    outfile.open(outfilename.c_str());

    if(!outfile.is_open()){
        std::cout << "error opening " << outfilename << std::endl;
        exit(EXIT_FAILURE);
    }

    while(infile >> word){
        // for each word in the second file we count the occurrences
        int occ = count_occurrences(words, word);

        // we print to the output file in a way very similar to how
        // we would print to std::cout
        outfile << word << ": " << occ << std::endl;
    }

```

```cpp
    // we close the file objects
    infile.close();
    outfile.close();

    // note that if you forget to close the output file
    // it can happen that nothing is printed to the file
    // or that the printing is incomplete;
    // this is because the printing is bufferised and may be
    // actually executed only when the file is closed

    return 0;
}

int count_occurrences(std::vector<std::string> words, std::string w){
    int occurrences = 0;
    for(int i = 0; i < words.size(); i++){
        // notice the straightforward use of the == operator
        if(words[i] == w){
            occurrences++;
        }
    }
    return occurrences;
}
```

Exercise

Test the program using the input in the example. Test it again on some other input.

Exercise

Edit the program so that if there are words in the second input file which have 0 occurrences, they are not included in the output file.

Operations on C++ string can also involve the content of the string considered letter by letter. In particular, several operations are available on strings in order to deal with letters that are similar to operations that are available on vectors.

The following example is a program which detects whether a text is a lipogram in a certain letter (a lipogram is a text that doesn't contain any instances of a certain letter).

The program reads from the user the name of a text file, reads the input from the file and detects and prints out any words that do not respect the constraints of the lipogram (if any) or prints that the text is indeed a lipogram (if that's the case).

The part of the program that operates on strings on a letter by letter basis is within function `contains_letter`, see also further explanations below.

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib>

bool contains_letter(std::string word, std::string letter);

int main(){

    std::ifstream infile;
    std::string filename;
    bool is_lipo = true;

    std::cout << "enter name of input file" << std::endl;
    std::cin >> filename;

    std::string letter;
    std::cout << "enter letter to check for lipogram" << std::endl;
    std::cin >> letter;

    infile.open(filename.c_str());

    if(!infile.is_open()){
        std::cout << "error opening file" << std::endl;
        exit(EXIT_FAILURE);
    }

    std::string word;

    while(infile >> word){
        if(contains_letter(word, letter)){
            is_lipo = false;
            std::cout<<"word containing "<<letter<<" found: "<<word<< std::endl;
        }
    }
    infile.close();

    if(is_lipo){
        std::cout << "the input is a lipogram in " << letter << std::endl;
    }

    return 0;
}

// note that the following function doesn't work
// for both uppercase and lowercase letters
// (but for now it's ok for our purposes)

bool contains_letter(std::string word, std::string letter){
    // strings have a size() member function which returns
    // the lenght of the string (similar to vectors)
    for(int i = 0; i < word.size(); i++){
        // the value of wordi will be reset at each iteration
        // because it's declared inside the loop
        std::string wordi;

        // the following instruction makes it so that wordi
        // is a string that contains one letter which is
        // the letter at index i in string word
        // (similar to vectors: strings can be indexed with []
        // and characters can be added using push_back)
        wordi.push_back(word[i]);

        // we compare string wordi and string letter
        if(wordi == letter){
            return true;
        }
    }
    // see below for more details about the code inside the loop
    return false;
}
```

Consider function `contains_letter`, the question is why we need to to write the code inside the loop as:

```cpp
    std::string wordi;
    wordi.push_back(word[i]);
    if(wordi == letter){
        return true;
    }
```

Instead of just:

```cpp
    // this wouldn't work!
    if(word[i] == letter){
        return true;
    }
```

The answer is that in C++ when we index a string the return value is not a string, but a variable of a different type (`char`) and strings and chars cannot be directly compared. Therefore those operations essentially convert the char we obtain from indexing the string into a string, so that all our types are consistent.

Another complication is related to the kind of quotes that needs to be used respectively for strings and for characters. Consider the following examples:

```
std::string s = "apple";
// this will not work!
// s[0] is a char while "a" is a (C-style) string
// because of the double quotes (")
if(s[0] == "a"){
    // something
}
```

```
std::string s = "apple";
// this will work, now 'a' is a char
// because of the single quotes (')
if(s[0] == 'a'){
    // something
}
```

Exercise

Test the program in order to determine whether the following text is a lipogram in ‘s’.

```
Mary had a little lamb
With fleece a pale white hue
And everywhere that Mary went
The lamb kept her in view
```

Then change the text to include some words containing ‘s’ and test the program again.

Exercise

Write a program that reads from the user the name of a text file, reads the text contained in the file and then counts how many occurrences for ‘a’, ‘e’, ‘i’, ‘o’ and ‘u’ (each both lowercase and uppercase) are contained in the text.

Exercise on text, strings and structured data types

You know how rental ads often sound like "exclusive cosy bedsit in a quiet area" when they actually mean "expensive small room in a remote area"?

Write a program which reads from the user the name of two files, one containing some advertisement text like that, and the other containing some sort of dictionary with the real meaning of the words, for example:

```
exclusive expensive
cosy small
bedsit room
quiet remote
```

The program reads the content of the files and stores it in memory, then translates the ad to something more realistic using the dictionary and finally prints on an output file the result.

In order to store the content of the file with the dictionary, define a `dict_item` structured type (with two fields of type string), the content of the file will be stored in a vector of `dict_item` elements.

Define and use in the program a function `dict_search`. The function takes in input said vector of `dict_item` elements and a string to look up and should return the index of the item containing the string, if found, and -1 otherwise.