Fall 2020 CS 31

Programming Assignment 5
See? Strings!

Time due: 11:00 PM Monday, November 23

Part 1

Go through the following sections of the class zyBook, doing the Participation Activities and Challenge Activities. We will be looking at whether you have ever successfully completed them; it does not matter how many attempts you make before a successful completion (or how many attempts you make after a successful completion if you want to experiment).

- 6.6 through 6.9
- 7.2 and 7.3 (Part 2 does not depend on these)
- 8.1 and 8.2 (Part 2 does not depend on these)

Part 2

There's a world of web pages out there, and people like to have a way to find the ones that are interesting to them. One way to do this involves specifying a set of match rules that can be applied to the text of each page to determine a match score. Those pages with the highest match scores should be the ones that are most interesting.

Let's call the text of a web page a document. For this problem, a match rule consists of either one word, w_{in}, or two words, w_{in} and w_{out}. A document matches a one-word rule if the document contains the word w_{in}; it matches a two-word rule if the document contains w_{in} and does not contain w_{out}. For example, the document consisting of the text

```
Happy families are all alike; every unhappy family is unhappy in its own way.
```

would match a one-word rule with w_{in} being `families`; it would not match a one-word rule with w_{in} being `confusion` (which is a word not contained in the document). That document would match a two-word rule with w_{in} being `family` and w_{out} being `ties` (since the document contains `family` and does not contain `ties`); and it would not match a two-word rule with w_{in} being `family` and w_{out} being `all` (since the document does contain `all`). The document

```
Of late, she felt hate for the fate of her mate.
```

does not match a one-word rule with w_{in} being `ate`, since that document does not contain the word `ate`.

We say that a collection of match rules is in clean form if

- Every word is at least one letter long and contains no characters other than lower case letters.
- No two-word rule has the same word as both its w_{in} word and its w_{out} word.
- For every one-word rule, no other one-word rule and no two-word rule has the same w_{in} word as that one-word rule.
- For every two-word rule, no other two-word rule has both the same w_{in} word as that rule's w_{in} word and the same w_{out} as that rule's w_{out}. (In other words, a collection in clean form has no duplicate two-word rules.)

Given a collection of match rules in clean form and a document, the score of that document is the number of rules in that collection that that document matches. (Notice that if a document matches a rule, the number of ways it does so is irrelevant. For example, if there is a one-word match rule with w_{in} being `unhappy`, then the contribution of this rule to the score for the document

```
Happy families are all alike; every unhappy family is unhappy in its own way.
```

is 1, not 2.)

For this project, you will write two functions: one to put a collection of purported match rules into clean form, and one to determine the score of a document. A collection of match rules will be represented using two arrays and an integer. The integer is the number of match rules in the collection; the component(s) of each match rule are stored in the same position across the two arrays. If there are n match rules, they occupy positions 0 through n-1 of the arrays. The two arrays are

- an array of C strings, holding the w_{in} words
- an array of C strings, holding the w_{out}words

A two-word rule will have its w_{in} word at some position in the first array and its w_{out} word at the corresponding position of the second array. A one-word rule will have its w_{in} word at some position in the first array and an empty string at the corresponding position of the second array.

We guarantee that we will never test your functions with a word in a match rule that is longer than 20 characters, so you are allowed to assume (and do not have to check) that the zero-bytes that denote the ends of the C strings are in fact present in the arrays that we give you. Prior to any of your function definitions, you must define the following constant (outside of any function definitions):

```
        const int MAX_WORD_LENGTH = 20;
```

and use it where appropriate.

Here are the two functions you must implement:

```
int cleanupRules(char wordin[][MAX_WORD_LENGTH+1],
                 char wordout[][MAX_WORD_LENGTH+1],
                 int nRules)
```

The parameters to this function represent a collection of zero or more purported match rules, located in positions 0 through `nRules-1` of the arrays. (Treat a negative `nRules` parameter as if it were 0.) When this function returns, the arrays must represent a collection of match rules in clean form, and the return value of the function is the number of match rules in that collection of match rules in clean form (which might even be zero). The function must transform every upper case letter in the match rule words into its lower case equivalent. The function must remove from the collection every purported match rule for which at least one of these conditions hold:

- A word in the match rule contains a character that is not a letter.
- A w_{in} word is the empty string.
- The w_{in} word and the w_{out} words are the same.

In addition, if a one-word rule has the same w_{in} as other rules (whether one-word or two-word), remove from the collection all such other rules, leaving just one instance of the one-word rule. For example, if there are three one-word rules, all with the w_{in} being `family`, and two two-word rules, both with the w_{in} word being `family` and one of those two with the w_{out} word being `ties` and the other with the w_{out} word being `first`, then remove from the collection four of those rules, keeping only one of the one-word rules.

Also, if a two-word rule has both the same w_{in} word as other two-word rules' w_{in} words and the same w_{out} word as those other rules' w_{out} words, and no one-word rule has the same w_{in} word as those rules' w_{in} words, then remove from the collection all such other two-word rules, keeping only one of those two-word rules.

Suppose, say, r is the value returned by this function. Then the remaining match rules in the collection of match rules in clean form that results from calling this function must be located in positions 0 through r−1 of the arrays, although they need not be in the same order they were in when the function was called. The contents of positions r though `nRules-1` of the arrays may be whatever you like; they don't even have to be valid C strings.

As an example, if elements 0 through 11 of the two arrays passed to the function were

```
"confusion"    ""
"FAMILY"       "TIES"
"charm"        "confusion"
"hearty"       "hearty"
"house"        "intrigue"
"worn-out"     "younger"
"family"       "first"
"charm"        ""
"ties"         "family"
""             "frightened"
"charm"        ""
"FaMiLy"       "tIeS"
```

and the function were called with a last parameter of 12, then the function must return 6, and elements 0 through 5 of the arrays must contain some permutation of the following match rules:

```
"confusion"    ""
"family"       "ties"
"charm"        ""
"ties"         "family"
"house"        "intrigue"
"family"       "first"
```

(and it doesn't matter what's in elements 6 though 11 of each array).

Your function must not assume any particular limit on `nRules`, the number of purported match rules.

```
int determineScore(const char document[],
                   const char wordin[][MAX_WORD_LENGTH+1],
                   const char wordout[][MAX_WORD_LENGTH+1],
                   int nRules)
```

The last three parameters represent a collection of match rules that this function is allowed to assume is in clean form. (In other words, it need not check this, and is allowed to not work correctly or even do something undefined if the collection is not in clean form.) The first argument is a C string containing the entire text of a document. You are allowed to assume that no document is longer than 250 characters, not counting the zero byte. (We guarantee we will not test your function with documents that exceed this limit.) The function must return the score of the document (i.e., the number of rules that the document matches). Treat a negative `nRules` parameter as if it were 0.

In the document, words are separated by one or more space characters (i.e., `' '`), upper case letters are to be treated as if they were lower case, and all non-alphabetic characters other than spaces are to be ignored — they do not act as word separators. Thus, the document

```
I'm upset that on Nov. 13th, 2020, my 2 brand-new BMW M850is were stolen!!
```

must cause the function to return the same value as this would:

```
im upset that on nov th my brandnew bmw mis were stolen
```

Notice that the function is not allowed to modify the characters in the document. It can, of course, make a copy of all or part of the document into local variables of its own if it wants. Since g31 enforces the Standard C++ requirement that the size of a declared array must be known at compile time, and your program must build under that compiler, you must not declare any local arrays whose size is not known at compile time. Thus, you must not do something like this:

```
void somefunction(char someword[])
{
    char a[strlen(someword)+1]; // Error! strlen(someword) not known at compile time
```

But because you are allowed to assume that no document is longer than 250 characters, this is no burden on you: You can determine an upper bound known at compile time for any local array you might declare.

The document might contain words that are longer than 20 characters, even though we guarantee that no word in a match rule will be longer than 20 characters. Your function must not assume any particular limit on `nRules`, the number of rules.

Your program must not use any `std::string` objects (C++ strings); you must use C strings. Your program must not use any containers from the C++ library such as vectors, lists, etc., and you must not use dynamic allocation with `new`, `malloc`, etc. Your functions must not use any global variables whose values may be changed during execution (so global constants like `MAX_WORD_LENGTH` are allowed).

The source file you turn in will contain the two required functions and a main routine. You can have the main routine do whatever you want, because we will rename it to something harmless, never call it, and append our own main routine to your file. Our main routine will thoroughly test your functions. You'll probably want your main routine to do the same. If you wish, you may write functions in addition to those required here. We will not directly call any such additional functions.

As an example, here is a main routine that contains some tests of the `determineScore` function:

```
#include <iostream>
#include <cassert>
using namespace std;
...
int main()
{
    const int TEST1_NRULES = 3;
    char test1win[TEST1_NRULES][MAX_WORD_LENGTH+1] = {
        "family", "unhappy", "horse",
    };
    char test1wout[TEST1_NRULES][MAX_WORD_LENGTH+1] = {
        "",       "horse",   "",
    };
    assert(determineScore("Happy families are all alike; every unhappy family is unhappy in its own way.",
                test1win, test1wout, TEST1_NRULES) == 2);
    assert(determineScore("Happy horses are all alike; every unhappy horse is unhappy in its own way.",
                test1win, test1wout, TEST1_NRULES-1) == 0);
    assert(determineScore("Happy horses are all alike; every unhappy horse is unhappy in its own way.",
                test1win, test1wout, TEST1_NRULES) == 1);
    assert(determineScore("A horse!  A horse!  My kingdom for a horse!",
                test1win, test1wout, TEST1_NRULES) == 1);
    assert(determineScore("horse:stable ratio is 10:1",
                test1win, test1wout, TEST1_NRULES) == 0);
    assert(determineScore("**** 2020 ****",
                test1win, test1wout, TEST1_NRULES) == 0);
    cout << "All tests succeeded" << endl;
}
```

The program you turn in must build successfully under both g31 and either Visual C++ or clang++, and during execution, neither of the two required functions, nor any function that they call, may read anything from `cin` or write anything to `cout`. If you want to print things out for debugging purposes, write to `cerr` instead of `cout`. When we test your program, we will cause everything written to `cerr` to be discarded instead — we will never see that output, so you may leave those debugging output statements in your program if you wish.

When your functions are called with arguments that meet the specifications, your program must not do anything that depends on undefined program behavior. Your program could not, for example, assume anything about `t`'s value, or even whether or not the program crashes:

```
    ...
    char t[6];
    strcpy(t, "family");  // too long: 7 chars including '₩0'
    ...
```

Note

Although the program you turn in must use C strings and is forbidden from using C++ strings, you can experiment with ideas for doing this project without that restriction. For example, you could create an experimental project (that you will not turn in) and pretend the required functions are

```
int fixRules(string wordin[],
             string wordout[],
             int nRules)

int figureScore(const string document,
                const string wordin[],
                const string wordout[],
                int nRules)
```

You could work out a lot of what you need to do for this project using C++ strings without the distraction of having to wrestle with C strings. Use what you learn from the experimental project when writing the real project that uses only C strings. Warning: It may not be wise to try to completely finish the experimental C++ string version before even starting the real C string version; it might take you more time than you thought to figure out how to work with C strings, so you might not have anything working in the C string version that you must turn in. Instead, when you have just a few things working in a C++ string version, try implementing them and getting them to work in the C string version, so you'll know how much time it takes you to translate from using C++ strings to C strings. Get more things working in the C++ string version, then in the C string version. Once you're comfortable with C strings, you might abandon the experimental version and continue on with just the real C string version. (Or not; maybe you prefer to continue working out each new bit with C++ strings first before implementing it with C strings.)

Notes for Visual C++ users

Microsoft made a controversial decision to issue by default a warning in some cases when your code uses certain functions from the standard C and C++ libraries (e.g., `strcpy`). These warnings call those functions unsafe and recommend using a different function in their place; those other functions, though, are not Standard C++ function, so will cause a compilation failure when you try to build your program under g++ or clang++. Therefore, for this class, we want to use functions like `strcpy` without getting that warning from Visual C++; to eliminate the warning messages, put the following line in your program before any of your `#include`s:

```
#define _CRT_SECURE_NO_WARNINGS
```

It is OK and harmless to leave that line in when you build your program using g31 or clang++ and when you turn it in.

If the compiler issues a warning C6262: Function uses 'NNNNN' bytes of stack: exceeds /analyze:stacksize '16384'. Consider moving some data to heap., where NNNNN is some number, you can eliminate that harmless warning by adding this line at the top of your program (and you can leave it in when you build with g31 and when you turn it in):

```
#pragma warning(disable : 6262)
```

Alternatively, in Visual Studio, select Project / yourProjectName properties, then select Configuration Properties / Code Analysis / General, and then in Code Analysis's stacksize, modify 16384 to, say, 100000.

If your program dies under Visual C++ with a dialog box appearing saying "Debug Assertion Failed! ... File: ...₩src₩isctype.c ... expression: (unsigned)(c+1)<=256", then you called one of the functions defined by `<cctype>`, such as `isalpha` or `tolower`, with a character whose encoding is outside the range of 0 through 127. Since all the normal characters you would use (space, letters, punctuation, `'₩0'`, etc.) fall in that range, you're probably passing an uninitialized character to the function. Perhaps you're examining a character past the `'₩0'` marking the end of a C string, or perhaps you built what you thought was a C string but forgot to end it with a `'₩0'`.

What to turn in

You won't turn anything in through the CS 31 web site for Part 1; the zyBook system notes your successful completion of the PAs and CAs. For Part 2, turn in a zip file containing these two files and nothing more:

1. A text file named **match.cpp** that contains the source code for your C++ program. Your source code should have helpful comments that tell the purpose of your data structures and program segments, explain any non-obvious code.

2. A file named **report.docx** or **report.doc** (in Microsoft Word format), or **report.txt** (an ordinary text file) that contains:

a. A brief description of notable obstacles you overcame.
b. A description of the design of your program. You should use pseudocode in this description where it clarifies the presentation.
c. A list of the test data that you could use to thoroughly test your functions, along with the reason for each test. You must note which test cases your program does not handle correctly. (This could happen if you didn't have time to write a complete solution, or if you ran out of time while still debugging a supposedly complete solution.) Notice that most of this portion of your report can be written just after you read the requirements in this specification, before you even start designing your program.

By Sunday, November 22, there will be a link on the class webpage that will enable you to turn in your zip file electronically. Turn in the file by the due time above.