Strings and Functions

Assigned: Sunday, October 4
Due: Monday, October 12, before midnight
Value: 20 points (for successfully submitting a functionally correct program to your Assembla repo before the deadline). Late submissions will receive a 0.

Executive Summary

In this lab you will develop a recursive function for performing a primitive type of regular expression matching.

Deliverables

You will be provided with a `main.c` file, and a file `assignment-4.c` defining the API (application program interface) for your code. This file contains stub functions with short descriptions. You will deliver a completed program source file assignment-4.c implementing the functionality described below.

Simple String Matching Algorithm

Let $S = \{c,g,a,t\}$ and S^* be the set of all strings consisting of characters from S, including the length-0 string "".

Define a matcher expression (ME) as follows:

- x, where $x \in S$ is a ME;
  that is, c, g,a and t are MEs. So is '.'
- x^* is an ME, where $x \in S$ is a ME;
  that is c^*, g^*, a^* and t^* are MEs. So is '.*'
- m_1m_2 is an ME, where m_1 and m_2 are MEs;
  for example cag, ca^*g, c^*ag^*, $.^*ac^*g^*ttt^*ag$ are all MEs

Define a string $s \in S$ to match an ME m iff

1. s is of length one, and m is either $s[0]$ or '.', or
2. m is $.^*n$ and n matches some suffix of s, or

3. m is x*n and s can be decomposed into a string s_1 followed by a string s_2 such that s_1 is 0 or more occurrences of x and s_2 matches n, or
4. m can be written as m_1m_2 and s as s_1s_2 such that m_1 matches s_1 and m_2 matches s_2 based on one of the three conditions above.

For example, s = "aaggaaacctcga" matches m = "aa.*ac*t.*", as follows:

1. s[0] matches m[0] (Rule 1),
2. s[1] matches m[1] (Rule 1),
3. s[2..6] = ggaaa matches .*a (Rule 2),
4. s[7..8] = cc matches c* (Rule 3),
5. s[9] = t matches t (Rule 1),
6. s[10..12] = cga matches .* (Rule 2)
7. Thus m matches s, by combining 1-6 using Rule 4.

Design an algorithm that takes a string s and ME m and returns whether s matches m. The `main.c` we provide contains a number of tests, but you should write some of your own test cases, in the spirit of the tests in our `main.c`. Make sure you consider corner cases.

Requirements

Setup

1. You need to create a Project named `Lab4` in Visual Studio for this lab
2. From the Solution Explorer, add the files `main.c` and `assignment-4.c` to your `Lab4` project, so your file structure will be `<repo>/Lab4/assignment-4.c`

Input format

Inputs for your functions will come from tests cases written in `main.c` file. You can assume that input format will always be correct. That is, you don't need to check the validity of input in your functions.

Output Format

Your match function should return TRUE or FALSE

Grading

Your score will be solely based on your program's functionality

Lab Specific FAQ

- Where can I read about Regular Expressions?

  You can read about regular expressions commonly used in unix environment here: http://www.grymoire.com/Unix/Regular.html

- What is s?

  s is the string that must match a matcher expression m. s consists of a combination of the characters c, g, a, and t.
  s, as well as m, is represented as a null-terminated array of characters, i.e., char[ ].

- What does it mean for a string to be 'null-terminated'?

  This means that the last element in a C style string is the null character, '\0'. This null character is used to indicate the end of the string when the size of the string is not provided. For example, the if s = "hey", then s[0] has the character 'h', s[1] = 'e', s[2] = 'y', and s[3] = '\0'.

- Can I use library functions for string manipulation?

  No. You are not allowed to use strlen, strcmp or any other C library string function.

- Is the empty string s = "" a valid input?

  Yes, the empty string will match m if m = ".*" or m = "x*" where $x \epsilon S$