Nome do Projeto: GleamEx
Equipe: Alan Teixeira da Costa, Johann Jakob Schmitz Bastos, Ricardo Magalhães Filho

Propósito do sistema

As ferramentas de Regex (expressões regulares) criam “molduras” e tentam achar partes do texto que se encaixam nelas. Uma vez achada, pode-se escolher o que se deseja fazer com essa informação: validar se um texto está seguindo o padrão esperado, achar informações que seguem um padrão no texto (ex.: Telefone), ou que seguem uma disposição (ex.: nome - CPF; ano de nascimento).

Gleam é um projeto aberto de uma linguagem de programação que vem ganhando relevância nos últimos meses. Ela se propõe a ser uma linguagem tipada amigável para a construção de sistemas, focando em ser muito paralelizável, consequentemente apresenta alta escalabilidade e é muito boa para web services. Por ser uma linguagem relativamente recente, a Gleam ainda carece de alguns recursos de linguagens mais maduras, sendo um dos principais o suporte à Regex.

Tendo em vista essa deficiência da linguagem Gleam, e tendo o intuito de contribuir com esse projeto aberto para que ele possa amadurecer e assumir um papel de relevância no meio em que se propõe, iremos com esse trabalho desenvolver uma biblioteca para oferecer as principais funcionalidades que um desenvolvedor pode querer quando trabalhar com Regex na linguagem Gleam.

Perfil dos usuários
Por conta da especificidade da biblioteca que será desenvolvida, podemos considerar um tipo de usuário direto, descrito a seguir: Validador de dados: Perfil: Desenvolvedor que precisa verificar se determinado dado está conforme o esperado, Objetivo: Usar expressões regulares para analisar se o dado segue o padrão esperado.

Principais funcionalidades

Linguagem a ser tratada: Por questões de escopo do curso, apenas será tratado um subconjunto das linguagens regulares. Do contrário, teríamos que gastar ainda mais tempo nos preocupando com ReDOS (Regular Expression Denial of Service), que ocorre quando o usuário digita, muitas vezes propositalmente, expressões regulares muito complicadas para que o processador gaste muitos recursos tratando-as, negando serviço a outras aplicações. Esse subconjunto será definido por todas as linguagens geradas por expressões regulares que contenha apenas os operadores de concatenação, OR (identificado pelo símbolo “|”) e por quantificadores (fecho de Kleene “*”, fecho positivo de Kleene “+” e quantificadores de números específicos). Posteriormente, pode ser possível adicionar o suporte aos operadores “?” e “^’. Seguem alguns exemplos de expressões regulares que serão tratadas inicialmente:

(ab) - Concatenação do caractere “a” com o caractere “b”.
(a | b) - Escolha entre caractere “a” ou “b”.
(a)* - Zero ou mais ocorrências do caractere “a”.

$(a)^n$ - n ocorrências do caractere “a”.
$a(a \mid b)^+$ - Concatenação entre o “a” e uma ou mais ocorrências de “a” ou “b”.

Principais funcionalidades: A principal funcionalidade oferecida pela biblioteca será a de determinar se uma dada string foi gerada por uma, também dada, expressão regular. Essa funcionalidade é chamada de “match”. Partindo disso, será construída outra funcionalidade fundamental: a de determinar se uma dada faixa de índices de uma string foi gerada pela expressão regular, chamada de “search”. Essas funcionalidades serão fornecidas através de funções da biblioteca, que terão as seguintes assinaturas:

bool regex_match(regex Regex, char *str);

bool regex_search(regex Regex, char *str, int first, int last);

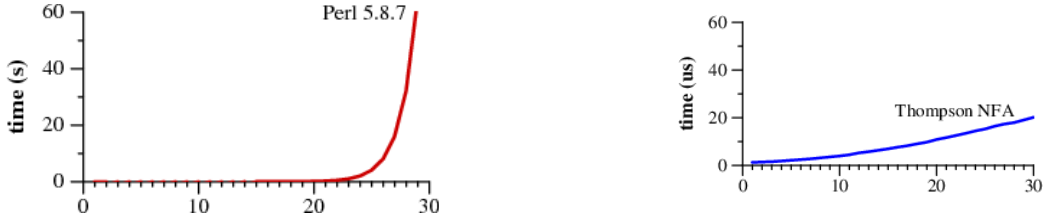
Funcionalidades opcionais: As funcionalidades tratadas aqui seguem as funcionalidades anteriores, e portanto podem ser implementadas a depender do andamento do projeto. A primeira delas é a de iterar sobre todos os matches individuais de uma dada expressão regular dentro da string alvo, chamada de “iterator”. A segunda é a de buscar, em uma dada string, todos os matches individuais de uma dada expressão regular e em seguida substituir esses matches por um texto da escolha do usuário. Como feito anteriormente, as assinaturas das funções serão:

match regex_iterator(regex Regex, char *str);

char * regex_replace(regex Regex, char* str, char* replace_text);

(Onde o tipo **match** é um tipo que consiste de um par ordenado, indicando índices de começo e final de uma string).

Tecnologias empregadas
Pelo projeto ser uma biblioteca para uma linguagem de programação o seu front-end é por linha de comando. O que será usado no desenvolvimento: Linguagens: C++(para fins de desempenho) e gleam; Bibliotecas: C++ Standard Template Library (STL); Automação de build: Cmake. Controle de Versão: git com o auxílio do GitHub.
Pelo motivo de desempenho, a abordagem para realizar o matching da expressão regular será diferente do que é implementado normalmente nas linguagens. No nosso caso, usaremos a implementação Thompson NFA, que possui algumas restrições em relação ao método usual, como não cobrir expressões regulares com backreferences, mas em questão de complexidade é muito melhor, como pode ser visto a seguir:



Time to match $a?^n a^n$ against a^n

Link do Figma

https://www.figma.com/file/zBsH43cJulvbKlfDs2bGtN/Projeto-Integrado?type=design&node-id=0%3A1&mode=design&t=XuE3Ge5alvqW9FNK-1