Informações gerais

O presente laboratório tem como objetivo avaliar conceitos elementares da linguagem de programação C, com foco na manipulação de cadeias de caracteres (strings), o que inclui o tipo `char` e o uso da tabela ASCII. Para isso, é proposto um problema clássico em Bioinformática, a busca de um padrão (“motif”) de nucleotídeos em uma sequência genômica.

Importante 1: Neste laboratório, será permitido o uso apenas da biblioteca `stdio.h`. Em particular, não será permitido o uso da biblioteca `string.h`. Além disso, para compilar o código, use o compilador `gcc` com as seguintes flags:

php-template

CopiarEditar

```
gcc -Wall -Werror -std=c99 <programa>.c -o <programa>
```

Importante 2: Todas as instâncias que envolvem vetores de caracteres não excederão 1000 caracteres. Recomenda-se que usem a diretiva `#define` para definir o tamanho máximo da sequência (e.g., `#define MAX_SEQ_SIZE 1000`).

Importante 3: Deve-se considerar entradas contendo letras maiúsculas e minúsculas (i.e., `"TTC"` equivale a `"TtC"`), sendo aconselhável criar uma função que converta as letras para minúsculas. Não é necessário verificar se a entrada contém caracteres que não sejam letras ou quebras de linha (`'\n'`).

Sequências genômicas e expressões regulares

Motivos (*motifs*, em inglês) são pequenos padrões de sequências que se repetem em sequências maiores de nucleotídeos ou de aminoácidos. Em sequências genômicas, motivos podem estar associados a diversas funções bioquímicas. Genomas são constituídos de cadeias de bases nitrogenadas (ou simplesmente bases): adenina (A), citosina (C), guanina (G) e timina (T).

Podemos buscar tais motivos utilizando expressões regulares (*regexp*), que são uma ferramenta poderosa para busca de strings, incluindo sequências genômicas. Para este laboratório, utilizaremos apenas curingas e quantificadores.

Curinga (*Wildcard*)

Trata-se de um símbolo (`.`) que significa qualquer caractere. Por exemplo, `"a.b"` é uma expressão regular que casa com qualquer cadeia que comece por `"a"`, seguida de qualquer caractere, e então `"b"`.

Quantificadores

São modificadores que especificam quantas vezes o caractere que o precede deve repetir. Os modificadores que utilizaremos neste laboratório são:

- `?`: Indica zero ou uma ocorrência do caractere que o precede. Por exemplo, `colou?r` casa com `"colour"` e também com `"color"`.

- `*`: Indica zero ou mais ocorrências do caractere que o precede. Por exemplo, `ab*c` casa com `"ac"`, com `"abc"`, com `"abbc"`, com `"abbbc"`, etc.
- `+`: Indica uma ou mais ocorrências do caractere que o precede. Por exemplo, `ab+c` casa com `"abc"`, com `"abbc"`, com `"abbbc"`, etc.

Busca de motivos em texto usando *regexp*

Quando utilizamos uma *regexp* para procurar um motivo no texto, a busca é "gulosa"; ou seja, se eu avaliar a string `"abbbcd"` com `ab*c`, será registrado o casamento somente para `"abbbc"`.

Enunciado

Escreva uma função `busca_motivo`, prototipada como:

```
unsigned int busca_motivo(char s[], char m[]);
```

Essa função recebe dois vetores de caracteres:

- `s` (sequência de nucleotídeos de até 1000 bases);
- `m` (motivo em *regexp*).

Ela deve devolver um inteiro não negativo contando o número de ocorrências de `m` em `s`. O seu programa deve supor que a cadeia de caracteres em `s` tem tamanho igual ou maior do que a contida em `m`.

Exemplo:

Se `s` for:

```
TCAAAAAAGGGCGGGGATGAAAAT
CCA
```

E `m` for:

```
T.?C*A+
```

Então a sua função deverá devolver **3**.

Avaliação

A implementação do quantificador `+` é obrigatória, e o seu código deverá passar por todos os testes que envolvam o seu uso.

Já as implementações dos demais quantificadores (`?` e `*`) e do coringa (`.`) são opcionais. Implementações corretas desses itens valem meio ponto cada (ou seja, se implementar todos eles corretamente, então você ganha **1,5 ponto de bônus** na nota deste laboratório).