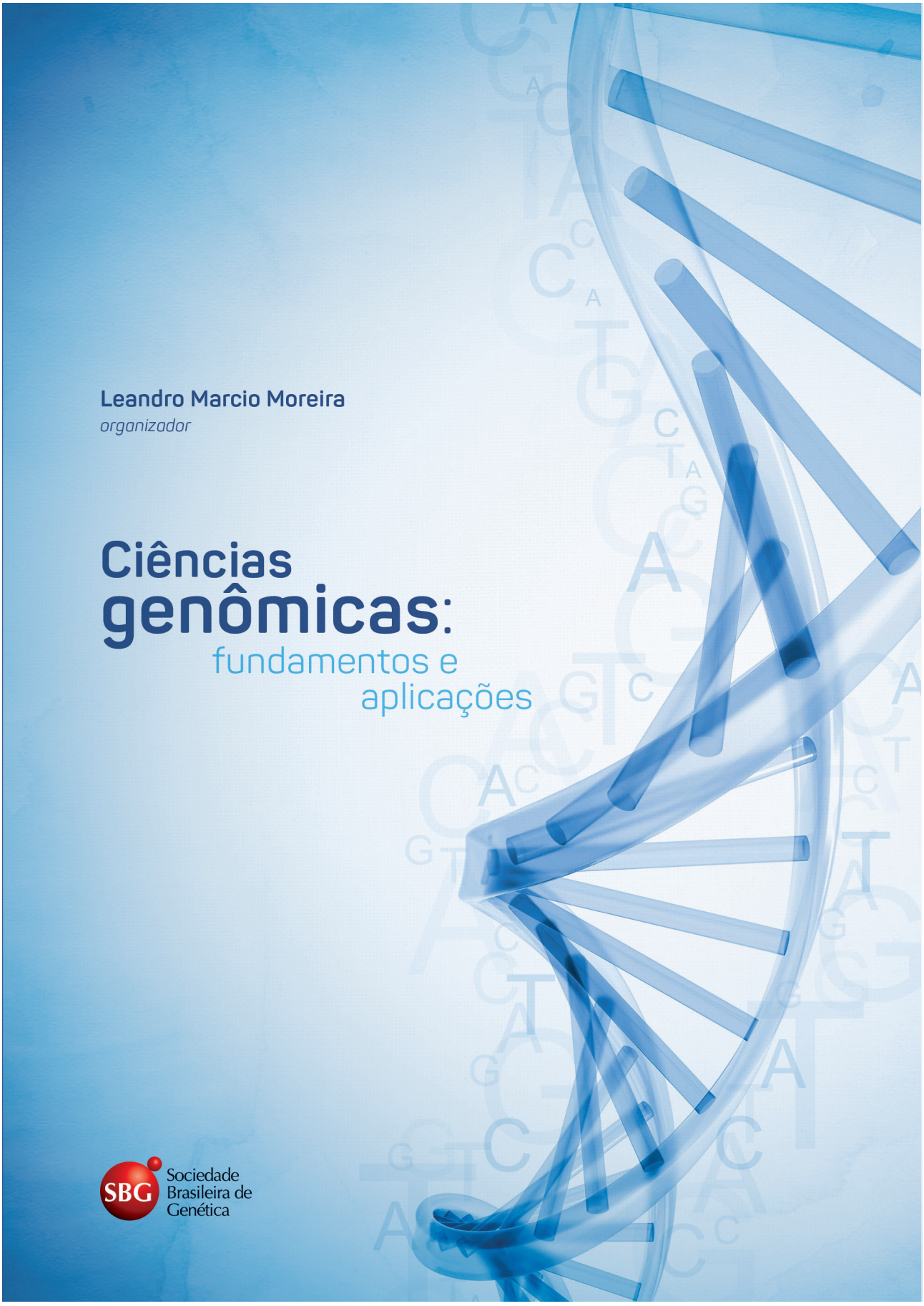
Leandro Marcio Moreira
organizador

Ciências genômicas:
fundamentos e aplicações



SBG Sociedade Brasileira de Genética

COMISSÃO EDITORIAL SOCIEDADE BRASILEIRA DE GENÉTICA

Editor

Élgion Lúcio Silva Loreto
Universidade Federal de Santa Maria

Comissão Editorial

Carlos Frederico Martins Menck
Universidade de São Paulo

Louis Bernard Klaczko
Universidade Estadual de Campinas

Marcio de Castro Silva-Filho
Universidade de São Paulo

Maria Cátira Bortolini
Universidade Federal do Rio Grande do Sul

Marcelo dos Santos Guerra Filho
Universidade Federal de Pernambuco

Pedro Manoel Galetti Junior
Universidade Federal de São Carlos

Sociedade Brasileira de Genética
Rua Capitão Adélmio Norberto da Silva, 736
Alto da Boa Vista | CEP 14025-670 | Ribeirão Preto, SP
Tels.: (16) 3621-8540 | contato@sbg.org.br | www.sbg.org.br

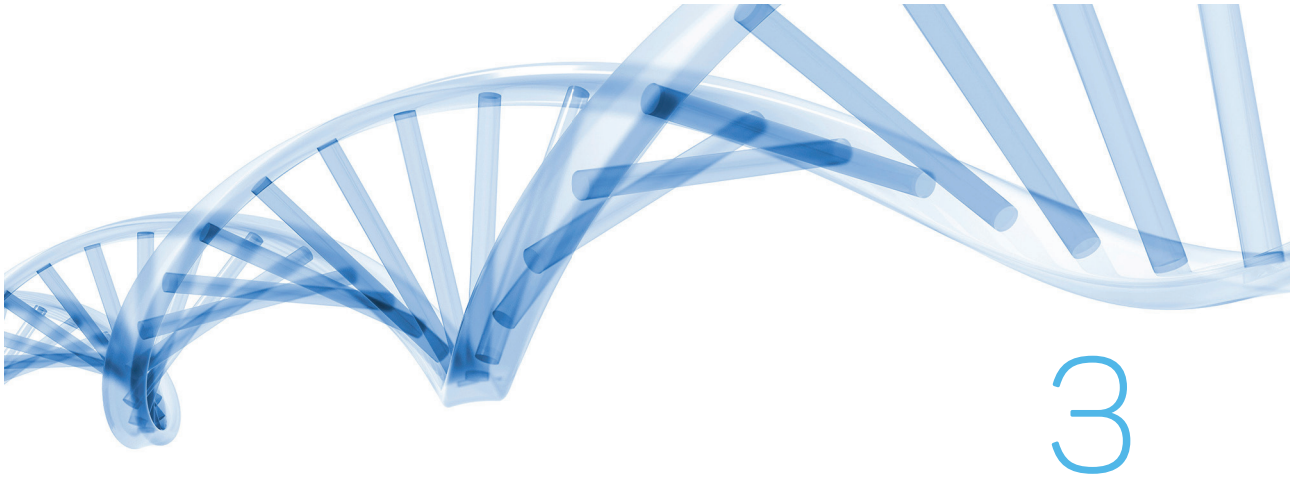
Ciências genômicas : fundamentos e aplicações / Leandro Marcio Moreira (organizador). – Ribeirão Preto: Sociedade Brasileira de Genética, 2015.
403 p: il

Idioma: Português
ISBN 978-85-89265-22-5

1. Genômica estrutural. 2. Genômica funcional. 3. Genômica comparativa. 4. Biologia sintética. 5. Biologia de sistemas. 6. DNA barcoding. 7. Metagenômica. 8. Epigenômica. 9. Metabolômica. 10. Filogenômica. I. Moreira, Leandro Marcio, org. II. Título.

Capa, projeto gráfico e diagramação

editora cubo
soluções para o universo acadêmico



3

Construindo bancos de dados biológicos

Luciano Digiampietri
Jerônimo Conceição Ruiz

Introdução

A bioinformática contemporânea engloba uma crescente e ampla gama de atividades relacionadas ao gerenciamento, análise e visualização da informação biotecnológica. Dentre a pletora de programas e abordagens de bioinformática aplicadas às analises de proteômica, genômica, microarranjos, metabolômica e biologia de sistemas, o desenvolvimento e utilização de banco de dados tem crescido em importância e mostrado sua fundamentalidade não só como essência dos grandes repositórios de dados biológicos como o DDBJ (DNA Data Bank of Japan), EMBL Nucleotide Sequence DB (European Molecular Biology Laboratory) e GenBank (National Center for Biotechnology Information), mas também no cotidiano científico dos laboratórios de pesquisa envolvidos em projetos que empreguem alguma abordagem de sequenciamento de nova geração.

Nosso objetivo nesse capítulo é trazer de uma maneira prática a essencialidade e aplicabilidade dos bancos de dados para o cotidiano das análises em bioinformática. Para tanto, a estratégia adotada integra a utilização de duas ferramentas de código aberto e independentes de plataforma: a linguagem de programação PERL (*Practical Extraction and Report Language*) e o Sistema de Gerenciamento de Banco de Dados (SGBD) MySQL. Pela sua consistência e segurança, o enfoque será dado em um tipo particular de banco de dados, o banco de dados relacional. Nos exemplos práticos utilizados, o MySQL representa o "Sistema Gerenciador de Banco de Dados Relacional" ou SGBDR que viabiliza a mineração das informações contidas nos bancos de dados de maneira extremamente eficiente e frequentemente através de comandos simples

criados através de uma linguagem de programação especial conhecida como SQL ou "*Structured Query Language*".

Tipos comuns de bancos de dados

Tendo como objetivo informar o leitor sobre as diferentes interpretações possíveis considerando o entendimento biológico e computacional de um DB, algumas considerações que levam em conta o *background* do leitor precisam ser feitas.

Assim sendo, no contexto dos leitores com *background* em bioinformática e área afins a definição de SGBC fica voltada para o entendimento de uma coleção de programas e recursos usados para armazenar, gerenciar e possibilitar a extração otimizada de informações em DBs relacionais. Cada SGBD pode gerenciar diversos Bancos de Dados (BD) locais ou distribuídos, sendo que em cada BD pode existir dezenas de centenas de tabelas (relações) extremamente estruturadas e relacionadas entre si (BDs relacionais).

Vale ainda destacar a existência SGBDs de distruição gratuita como MySQL (citado anteriormente) e Postgres e comericais como Oracle, SqlServer e DB2 e que portais como o NCBI, EMBL, DDBJ usam para gerenciar e disponibilizar suas informações aos usuários um tecnologia mista envolvendo SGBD e *flat-file*.

Arquivos texto (Flat text files)

Nesse tipo particular de banco de dados toda informação é armazenada em um esquema simples através de arquivos texto comumente denominada "flat files". Esses textos podem representar qualquer informação e são criados em um formato comum de leitura como os arquivos .txt criados por editores de texto como o TextPad (http://www.textpad.com/download/) ou Kate (http://kate-editor.org/). Um conjunto de documentos como esse no disco rígido do computador, representa um exemplo de banco de dados não realcional, contudo é de grande ajuda se algum tipo de estrutura de ordenamento é imposta à esses arquivos. Em termos de anotação genômica por exemplo, esses arquivos poderiam representar as sequencias de genes codificadores de proteínas, de RNAs (tRNAs e rRNAs) e outros elementos estruturais como repetições e regiões promotoras além é claro dos "contigs" ou "contig" (*Contig é o termo generico utilizado para descrever uma sequência consenso gerada computacionalmente a partir de fragmentos de sequências menores relacionados uns aos outros em função da sobreposição de suas sequências*) que representam a sequencia completa do genoma. A utilização desse tipo de estruturação ajuda o pesquisador que analisa os dados a rapidamente encontrar a informação de interesse e posteriormente aprofundar a pesquisa em uma subseção particular. Essa estruturação também ajuda o processo de extração de relatórios ou "parsing" dos dados. Se considerarmos um banco de dados texto formados apenas por sequencias no formato FASTA, várias informações extremamente úteis poderiam ser extraídas, vejamos o exemplo à seguir:

As duas sequências presentes na Figura 1, (114794895 e 212375028), são apresentadas no formato Fasta (Pearson) (http://www.ebi.ac.uk/help/formats.

```
>gi|114794895|pdb|2HAQ|A Chain A, Cyclophilin A From Leishmania Donovani
HHHHHHEPEVTAKVYFDVMIDSEPLGRITIGLFGKDAPLTTENFRQLCTGEHGFGYKDSI

>gi|212375028|pdb|3EOV|B Chain B, Cyclophilin From Leishmania Donovani
HHHHHHEPEVTAKVYFDVMIDSEPLGRITIGLFGKDAPLTTENFRQLCTGEHGFGYKDSI
```

Figura 1. Exemplos de sequências no formato FASTA.

html#fasta). Esse formato é amplamente utilizado em bionformática e possui várias características que podem ser exploradas.

Com um olhar criterioso poderíamos identificar vários padrões estruturais do texto que descrevem as duas sequencias que poderiam ser utilizados na extração de informações. São eles:

a. O formato Fasta contem uma única linha de cabeçalho para cada sequencia descrita no arquivo que se inicia com o símbolo “>” (maior);
b. Logo após o símbolo de maior (>) vários blocos de informação são separados pelo sinal “|”(pipe). O primeiro bloco de informação representa o gi, ou número de identificação da proteína no NCBI (http://www.ncbi.nlm.nih.gov/Sitemap/sequenceIDs.html), o segundo bloco de informação representa um identificador único do banco de dados PDB (http://www.rcsb.org/pdb/staticHelp.do?p=help/advancedsearch/pdbIDs.html), o terceiro e último bloco representa uma breve descrição da proteína anotada e o organismo de origem seguido pela inserção de uma nova linha. Apesar de serem apresentados apenas três blocos informativos, teoricamente não existe um limite para a quantidade de blocos, contudo apenas uma linha pode ser utilizada. A limitação fica relacionada a dificuldade humana de leitura de uma linha muito longa; e
c. As linhas restantes representam a sequencia propriamente dita. Rotineiramente são representados 60 caracteres por linha mas esse número pode variar até 80 caracteres por linha.

Apesar de o exemplo apresentar apenas um trecho da sequencia de proteína, o padrão identificado é consistente para um arquivo multi-FASTA e assim um dos pontos centrais nesse arquivo é que dentro dele os dados são separados pelo emprego de uma estrutura definida. Nenhuma informação sobre a convenção de nomes utilizada na nomenclatura das proteínas, sua descrição ou sequência é necessária para atribuição dos dados nesses três padrões. Além disso, cada pedaço de informação do arquivo está explicitamente relacionado pela sua localização no arquivo.

Tendo em mente essa abordagem, vários formatos muito mais elaborados como o formato EMBL (http://www.ebi.ac.uk/embl/Documentation/User_manual/usrman.html) e o formato GenBank (http://www.ncbi.nlm.nih.gov/Sitemap/samplerecord.html) são utilizados em muitos aplicativos de bioinformática.

Arquivos XML (Extensible Markup Language)

O formato XML representa um formato bastante empregado em vários aplicativos de bioinformática e adiciona o conceito de sintaxe a um texto estruturado. Similarmente ao que acontece na linguagem escrita (língua portuguesa por exemplo) onde o emprego de uma pontuação e gramática correta ajudam o entendimento da mensagem escrita, a sintaxe define a ordem dos elementos de linguagem a serem utilizados no texto.

A descrição do arquivo Fasta descrito anteriormente poderia ser feita pela utilização da linguagem XML, contudo a idealização dessa formatação de arquivo foi feita primariamente para ser "machine readable", ou seja, feita para ser interpretada pelo computador, o que torna difícil o entendimento e leitura humana desse tipo de arquivo.

Apesar disso, a sintaxe XML básica é composta por um pequeno número de elementos estruturais. O primeiro elemento básico de um arquivo XML é a declaração que define a versão de XML utilizada e o tipo codificação. Veja o exemplo abaixo:

<? XML version="1.0" enconding="UTF-8"?>

Adicionalmente, vários outros elementos estruturais podem ser utilizados tais como: *names* (nomes), *attributes* (atributos) e *values* (valores). A informação textual pode ser integrada em cada um desses elementos e genericamente poderiam ser escritos da seguinte forma:

```
<nome atributo_1="valor_1"
                atributo_2="valor_2"

                atributo_n="valor_n">
                Algum texto o elemento nome
</nome>
```

No contexto da bioinformática que utiliza uma grande e heterogênea variedade de formatos de dados tais como fasta, embl, genbank, csv, resultados de alinhamento (.aln), filogenia, etc, o formato XML é muito interessante uma vez que formatos pré-definidos facilitam a validação e consistência dos dados bem como sua análise. Várias propostas de modelos estão disponíveis para bioinformática entre eles podemos citar: AGAVE (Architecture for Genomica Annotation, Visualization and Exchange | http://www.agavexml.org/), BIOML (BIOpolymer Markup Language | http://www.thegpm.org/bioml/), BSML (Bioinformatic Sequence Markup Language | http://xml.coverpages.org/bsml.html), HSAML (Multiple sequence alignment format with guide tree data anda quality scores | http://www.ebi.ac.uk/goldman-srv/hsaml/) e NCBI (http://www.ncbi.nlm.nih.gov/IEB/ToolBox/XML/).

Como exemplo utilizaremos a proteina com número de acesso GI:7387828 que foi salva do NCBI no formato XML (http://www.ncbi.nlm.nih.gov/protein/7387828) (Figura 2). Devido a extensão do arquivo, e para efeito de maior clareza das informações, apenas parte das informações sobre esta proteína foram apresentada.

```
<!DOCTYPE Bioseq-set SYSTEM "http://www.ncbi.nlm.nih.gov/dtd/NCBI_Seqset.dtd" PUBLIC "-//NCBI//NCBI Seqset/EN">
<Bioseq-set>
  <Bioseq-set_seq-set>
    <Seq-entry>
      <Seq-entry_seq>
        <Bioseq>
          <Bioseq_id>
            <Seq-id>
              <Seq-id_swissprot>
                <Textseq-id>
                  <Textseq-id_name>LEA3_WHEAT</Textseq-id_name>
                  <Textseq-id_accession>Q03968</Textseq-id_accession>
                  <Textseq-id_release>reviewed</Textseq-id_release>
                  <Textseq-id_version>1</Textseq-id_version>
                </Textseq-id>
              </Seq-id_swissprot>
            </Seq-id>
            <Seq-id>
              <Seq-id_gi>7387828</Seq-id_gi>
            </Seq-id>
          </Bioseq_id>
          <Bioseq_descr>
            <Seq-descr>
              <Seqdesc>
                <Seqdesc_title>RecName: Full=Late embryogenesis abundant protein, group 3; Short=LEA; AltName:
Full=PMA2005</Seqdesc_title>
              </Seqdesc>
              . . .
    <Title>
                <Title_E>
                  <Title_E_name>Sequence and regulation of a late embryogenesis abundant group 3 protein of maize.</Title_E_name>
                </Title_E>
              </Title>
              . . .
        <Bioseq>
      <Seq-entry_seq>
    <Seq-entry>
  <Bioseq-set_seq-set>
<Bioseq-set>
```

Figura 2. Proteína com número de acesso GI:7387828 salva a partir do NCBI no formato XML.

Assim como os bancos de dados gerenciados por um SGBD, os arquivos XML possuem estrutura extremamente flexível. Isto é, não existem regras rigorosas para determinar quais campos e informações farão parte destes arquivos e assim a padronização final fica dependente do problema a ser resolvido, do conhecimento e criatividade de quem esta gerando tais arquivos. Arquivos em formatos XML podem ser complexos para a interpretação humana, porém, são facilmente interpretados por inúmeros programas de computador, inclusive alguns SGBDs, navegadores WEB e bibliotecas de diversas linguagens de programação como PERL (http://search.cpan.org/~msergeant/XML-Parser-2.36/Parser.pm). Por este motivo XML é um formato muito utilizado para migrar (transferir) dados de um determinado sistema informatizado para outro.

Relevância da linguagem Perl

A linguagem de programação Perl (*Practical Extraction and Report Language | http://www.perl.com*) foi originalmente concebida para processar textos de maneira fácil e otimizada. Esta é uma das principais características que popularizou esta linguagem nas ferramentas de bioinformática nas quais, comumente, é necessário processar textos (sequências de caracteres) utilizando poderosos mecanismos de expressões regulares.

Há três tipos principais de variáveis na linguagem Perl: escalares (simples), arranjos (ou vetores) e hashes. As variáveis não precisam ser declaradas e o tipo do conteúdo de uma variável é atribuído dinamicamente, ou seja, uma variável pode receber valores numéricos e efetuar cálculos sobre esses valores e em seguida concatenar uma sequência de caracteres a esta mesma variável. A Figura 3 apresenta um código simples escrito em Perl que utiliza os três principais tipos de variáveis da linguagem. Na linha 2 uma variável escalar chamada *$simples* recebe o valor 5, que é multiplicado por 4 na linha 3 e o seu resultado é impresso na linha 4. Na linha 5 essa variável tem um texto concatenado ao seu valor e o novo valor da variável *$simples* é impresso na linha 6. Na linha 7 um arranjo chamado *@arranjo* recebe três valores. O laço das linhas 8 a 10 imprime cada um dos valores do arranjo. A expressão *$#arranjo* indica o índice do último elemento na variável *@arranjo* e neste caso *$#arranjo* vale 2, pois há três elementos (das posições 0 até 2). Na linha 11 uma variável do tipo hash, chamada *%hash* é declarada nas linhas seguintes duas chaves e seus respectivos valores são inseridos

```
1  #!/usr/bin/perl -w
2  $simples = 5;
3  $simples *= 4;
4  print "$simples\n";
5  $simples = $simples . ", este foi o resultado da multiplicação.";
6  print "$simples\n";
7  @arranjo = ('a', 'b', 'c');
8  for ($i=0;$i<=$#arranjo;$i++){
9    print "$arranjo[$i]\n";
10 }
11 my %hash;
12 $hash{'h1'} = "elemento h1";
13 $hash{'h2'} = "elemento h2";
14 verificaChave('h1');
15 verificaChave('h3');
16 exit 0;
17 sub verificaChave{
18   $chave = shift;
19     if (defined($hash{$chave})){
20       print "Chave $chave definida: $hash{$chave}\n";
21     }else{
22       print "Chave $chave não definida.\n";
23   }
24 }
25
```

Figura 3. Exemplo de Variáveis em Perl.

neste hash. Uma função (ou subrotina) é criada da linha 18 a 25 para verificar se uma dada chave existe ou não no hash. Note que tipos escalares, arranjos e hashs, quando declarados, utilizam respectivamente os seguintes caracteres especiais $, @ e % (linhas 2, 7 e 11). Porém, para acessar o conteúdo individual de um escalar de um arranho ou hash utilliza-se o caractere $ (linhas 9, 12, 13, 20 e 21 do código).

O resultado da execução do programa anterior (Figura 3) pode ser visto na Figura 4.

Um pequeno exemplo de um programa escrito em Perl para processar uma sequência de caracteres é apresentado na Figura 5. Neste exemplo deseja-se analisar a sequência de aminoácidos de uma proteína alfa-amilase da *Haloarcula hispanica* (sequência retirada do GenBank). Na primeira linha do código há uma indicação de como esse programa deve ser processado: pelo programa perl, localizado em /usr/bin e com o parâmetro -w indicando que os avisos (*warnings*) devem ser exibidos caso ocorram operações ilegais. Na segunda linha é declarada uma variável chamada $sequencia que recebe como (valor) parâmetro as informações da alfa-amilase (tanto

```
20
20, este foi o resultado da multiplicacao.
a
b
c
Chave h1 definida: elemento h1
Chave h3 não definida.
```

Figura 4. Resultado da Execução do Programa na Figura 3 do Capítulo 2.

```
1  #!/usr/bin/perl -w
2  my $sequencia = ''
3  >CAI64586 alpha-amylase [Haloarcula hispanica].
4  MNRPRITGSKQASRRTVLKGIGVLGAAVFGTAASVGSSAAVGDSAVYQYYHTDWTEITAT
5  LETVAQQGYDAIQVPPAQRSRLDRSHQNGVTDPPLGYQPVDLTDFNSVFGTEDEYEAMVQ
6  EAHNQDLDVVADAVINHMAANDDFRDAPGITFADLPRFSERDFHPKDDINYDNPESVEDD
7  WLVGLKDLKQESAYVRGELQAYVQKYADLGVDGIRWDAAKHVPESFFADYANQWADDLDL
8  WTVGEVLDGDIGVCQGYADTGMSVTDYPLYYTMKEEAFHSDGNMQALDGAGMVNQSPFQA
9  FTFVSNHDSGPPDYEKLAYAYILTYEGYPRVYSNRISVDDDDIRNLLWIRNNLASGGSQT
10 RHVDQDLYVYEREGNLLVGLNRASGQRSKWVPTSWTNQTLNDYSGNAGNISTNGDSWVQI
11 TVPATSWVCYAPE'';
12 $sequencia =~s/.+\.//;
13 $sequencia =~s/\n//g;
14 if ($sequencia =~ /DDD(...)/){
15   print "$1\n";
16 }
17 exit 0;
```

Figura 5. Exemplo de Uso de Expressão Regular em Perl.

```
1  create database livro;
2  use livro;
3  create table `ResultadosBlast` (
4    `QueryAccession` varchar(63) NOT NULL,
5    `Accession` varchar(63) NOT NULL,
6    `Description` varchar(1023) NOT NULL,
7    `Genus` varchar(511) NOT NULL,
8    `MaxScore` int(11) NOT NULL,
9    `TotalScore` int(11) NOT NULL,
0    `QueryCoverage` int(11) NOT NULL,
10   `EValue` double NOT NULL,
11   primary key (`QueryAccession`,`Accession`)
12 );
13
```

Figura 6. Comandos MySQL para criação de banco de dados e tabela.

o cabeçalho quanto a sequência propriamente dita). As linhas 12, 13 e 14 possuem diferentes expressões regulares. A expressão da linha 12 faz com que todo conteúdo (.+) anterior ao ponto final (\.) seja substituído por nada (ou seja, removido), com esta expressão o cabeçalho da sequência é removido. A expressão regular da linha 13 faz que todos os "finais de linha" (**\n**) sejam removidos, ou seja, as diversas linhas da sequência serão concatenadas em uma única linha. Por fim, a expressão regular da linha 14, que está dentro de um desvio condicional (**if**), verifica se existe nesta sequência uma subsequência iniciada por três aminoácidos D seguidos por três aminoácidos quaisquer: **DDD(...).** Os parênteses nessa expressão servem para indicar que o conteúdo entre eles deverá ser armazenado automaticamente na variável especial do perl ($1) para posterior utilização. Na linha 12 é solicitado que o conteúdo de $1 seja impresso seguido de uma instrução de "fim de linha" (**\n**). O resultado da execução desse programa será a impressão na tela dos aminoácidos **DIR**, pois, como visto na linha 9 do código, os três primeiros aminoácidos posteriores ao padrão de três letras D são os aminoácidos **DIR**. Vale a pena lembrar que dependendo da base de dados utilizada, os aminoácidos podem ser apresentados por letras minúsculas ou maiúsculas, como é o caso das sequências do GenBank onde são representadas por letras maiúsculas.

O SGBD MySQL

De acordo com a Seção 2 deste capítulo, MySQL é um dos mais populares sistemas gerenciadores de bancos de dados (SGBD) gratuitos.

Nesta seção será criado um banco de dados extremamente simples dentro do MySQL, que será utilizado nos exemplos do uso de Perl juntamente com MySQL.

A Figura 6 apresenta um conjunto de comandos para a criação de um banco de dados e de uma tabela. Esses comandos devem ser executados de dentro do MySQL.

```
use livros;
show tables;
+-----------------+
| Tables_in_livro |
+-----------------+
| ResultadosBlast |
+-----------------+

desc ResultadosBlast ;
+----------------+---------------+------+-----+---------+-------+
| Field          | Type          | Null | Key | Default | Extra |
+----------------+---------------+------+-----+---------+-------+
| QueryAccession | varchar(63)   | NO   | PRI | NULL    |       |
| Accession      | varchar(63)   | NO   | PRI | NULL    |       |
| Description    | varchar(1023) | NO   |     | NULL    |       |
| Genus          | varchar(511)  | NO   |     | NULL    |       |
| MaxScore       | int(11)       | NO   |     | NULL    |       |
| TotalScore     | int(11)       | NO   |     | NULL    |       |
| QueryCoverage  | int(11)       | NO   |     | NULL    |       |
| EValue         | double        | NO   |     | NULL    |       |
+----------------+---------------+------+-----+---------+-------+
```

Figura 7. Comandos MySQL: show tables e desc.

Na linha número 1 (um) desta figura é possível observar a criação de um novo banco de dados chamado livro (dentro de um SGBD vários bancos de dados podem ser criados). Na segunda linha a instrução é que utilizaremos agora este banco para a criação de uma nova tabela. As linhas de 4 a 13 descrevem a estrutura da tabela que será criada (ResultadosBlast). Esta tabela deverá albergar um resumo dos resultados de busca por similaridade de sequências produzidas pela ferramenta BLAST (http://blast.ncbi.nlm.nih.gov/Blast.cgi). As linhas de 5 a 11 descrevem os campos da tabela e a linha 12 informa que a chave primária da tabela será composta pelos campos QueryAccession e Accession.

A Figura 7 apresenta dois comandos usados para consultar informações gerais de um banco de dados. Os resultados destes comandos, quando utilizados dentro do nosso banco de dados "livro" são: **show tables;** que exibe a lista de tabelas dos banco de dados e **desc <nome_da_tabela>;** que exibe a descrição de uma tabela, isto é, nome dos campos, tipos, etc.

Acessando banco de dados relacionais utilizando Perl

Em Perl, assim como na maioria das linguagens de programação, existem bibliotecas (ou módulos) para facilitar o acesso aos bancos de dados. Neste capítulo será utilizada a biblioteca DBI[2] (*Database Interface* | http://dbi.perl.org/) para a comunicação com o SGBD MySQL. Para utilizar (ou importar) bibliotecas em Perl, basta tê-las instaladas em seu sistema e utilizar o comando *"use"* seguido do nome da biblioteca.

A Figura 8 apresenta um programa Perl que abre um arquivo do tipo CSV (*Comma Separated Values*) que contém a tabela dos *hits* de alinhamentos produzidos pelo programa BLAST e salvo previamente no computador (arquivo Alignment-HitTable.

```
1  #!/usr/bin/perl -w
2  use DBI;
3  if ($#ARGV<1){
4    die "Modo de uso: perl -w parserBlastTabela.pl <nome do arquivo blast><query accession id>\nExemplo: perl -w parserBlastTabela.pl Alignment-HitTable.csv CAI64586.1";
5    }
6  my $arquivo = $ARGV[0];
7  my $queryAccession = $ARGV[1];
8  my $DATABASE = 'livro';
9  my $TABLE = 'ResultadosBlast';
10 my $CONFIG  = ";mysql_read_default_file=/home/usuario/.my.cnf";
11 my $dbh = DBI->connect("DBI:mysql:$DATABASE" . $CONFIG, , ) or die "Não foi possivel conectar no bando de dados '$DATABASE'\n";
12 my $sth;
13 open(IN, "< $arquivo") or die "Nao foi possivel abrir arquivo de entrada: '$arquivo'\n";
14 my @campos;
15 my $SQL_INSERT;
16 my $genero;
17 while (<IN>){
18   chomp;
19   @campos = split("\t", $_);
20   $campos[4]=~s/\%//;
21   if ($campos[1] =~ /\[/){
22                         $genero = $campos[1];
23                         $genero =~s/\].*//;
24                         $genero =~s/.*\[//;
25                         $genero =~s/ .*//;
26                         $SQL_INSERT = "INSERT INTO $DATABASE.$TABLE VALUES ('$queryAccession', '$campos[0]', '$campos[1]', '$genero', $campos[2], $campos[3], $campos[4],
27 $campos[5]);";
28                         $sth = $dbh->prepare($SQL_INSERT);
29                         if ($sth->execute){
30                           print "Inserção realizada com sucesso.\n";
31                         }else{
32                           print "Não foi possivel inserir o registro.\n";
33                         }
34   }
35 }
36 close IN;
37 exit 0;
```

Figura 8. Programa para Ler Resultados do BLAST e Armazená-los no Banco de Dados.

csv). Para gerar este arquivo, foi feito um alinhamento de aminoácidos da sequência da alfa-amilase CAI64586.1 contra o banco *nr (non redundante database)* do GenBank. Na Figura 3.6, observamos na linha 2 o uso da biblioteca DBI. A condição da linha 3 verifica se menos que dois parâmetros foram passados como entrada para o programa, em caso afirmativo a execução será cancelada, pois o programa espera receber dois parâmetros: o nome do arquivo de entrada e a identificação da sequência que foi usada como sequência de consulta do alinhamento. Nas linhas 6 e 7 as variáveis do programa recebem o valor dos parâmetros de entrada. Das linhas 8 a 10 há a definição de algumas variáveis para acesso ao banco de dados, como o nome do banco (livro), o nome da tabela (ResultadosBlast) e por fim parâmetros de configuração que indicam como a autenticação será feita no banco de dados. Neste caso utilizamos o arquivo *.my.cnf (O arquivo .my.cnf contém as informações necessários para o estabelecimento de uma conexão com o banco de dados, por exemplo, nome do usuário e senha. Usando este arquivo o nome do usuário e a senha não ficam expostos no código do programa)* que foi criado no diretório exemplo */home/usuario*. Na linha 11 há existe conexão com o banco de dados e é importante observar que um dos parâmetros indica que o SGBD será o MySQL. Na linha 13 o arquivo de entrada é aberto para leitura. O laço que se estende da linha 17 até a linha 35 será executado enquanto houver linhas a serem lidas no arquivo de entrada e cada uma dessas linhas do arquivo será jogada numa variável padrão chamada *$_*. Das linhas 19 a 26 alguns processamentos são feitos para separar cada linha de entrada em campos e para tentar identificar o gênero de cada uma das descrições dos resultados do BLAST. Na linha 27 foi definida uma variável com o comando SQL para inserção dos resultados BLAST no banco de dados. Nessa linha

é necessário dizer em que tabela será feita a inserção e definir os valores que serão inseridos em cada campo da tabela, separados por vírgula. Na linha 28 a variável $sth recebe o comando pré-processado pela biblioteca DBI e na linha 29 esse comando será executado. Se não houver erro na execução do comando a mensagem "Inserção realizada com sucesso." será exibida na tela. Na linha 36 o arquivo de entrada é fechado.

BioPerl

BioPerl (http://www.bioperl.org) consiste de uma biblioteca desenvolvida para facilitar o processamento de dados biológicos. Esta biblioteca apresenta diversas funcionalidades relacionadas a bioinformática, ferramentas para copiar sequências e outras informações de bancos de dados importantes, como o GenBank, e tratamento de diferentes tipos de arquivos, como FASTA ou resultados da ferramenta BLAST.

O pacote chamado **bioperl** encontra-se disponível no repositório de pacotes para Linux, podendo ser facilmente instalado (http://www.bioperl.org/wiki/Installing_BioPerl). Para utilizar o pacote BioPerl em um programa Perl basta inserir a linha "use Bio::Perl;" no início do código e utilizar os métodos e funções desejadas. A Figura 9 apresenta um código que utiliza funções da biblioteca bioperl para baixar sequências do GenBank no formato FASTA, tendo como parâmetro o *accession* da sequência. A linha 2 deste código informa que o código utilizará a biblioteca BioPerl. Na linha 3 é declarada a variável *database* com o valor *genbank*. A linha 4 contém um arranjo com a identificação das sequências que serão baixadas, neste exemplo há apenas duas sequências no arranjo. Na linha 5 é definida uma variável *format* que indica que o formato utilizado será o FASTA. Das linhas 6 a 9 há um laço iterativo que para cada um dos identificadores de sequência do arranjo *accessions* fará o seguinte: utilizará a função *get_sequence* da biblioteca BioPerl para receber e guardar na variável *sequence* a sequência desejada (linha 7) e utilizará a função *write_sequence* para imprimir essa sequência no formato FASTA.

O resultado da execução desse código pode ser visto na Figura 10.

```
1 #!/usr/bin/perl -w
2 use Bio::Perl;
3 $database="genbank";
4 @accessions = ("CAI64586.1","BAD06002.1");
5 $format="fasta";
6 for (my $i=0;$i<=$#accessions;$i++){
7   $sequence = get_sequence($database, $accessions[$i]);
8   write_sequence(">-", $format, $sequence);
9 }
```

Figura 9. Exemplo de Código Utilizando BioPerl.

```
>CAI64586 alpha-amylase [Haloarcula hispanica].
MNRPRITGSKQASRRTVLKGIGVLGAAVFGTAASVGSSAAVGDSAVYQYYHTDWTEITAT
LETVAQQGYDAIQVPPAQRSRLDRSHQNGVTDPPLGYQPVDLTDFNSVFGTEDEYEAMVQ
EAHNQDLDVVADAVINHMAANDDFRDAPGITFADLPRFSERDFHPKDDINYDNPESVEDD
WLVGLKDLKQESAYVRGELQAYVQKYADLGVDGIRWDAAKHVPESFFADYANQWADDLDL
WTVGEVLDGDIGVCQGYADTGMSVTDYPLYYTMKEEAFHSDGNMQALDGAGMVNQSPFQA
FTFVSNHDSGPPDYEKLAYAYILTYEGYPRVYSNRISVDDDDIRNLLWIRNNLASGGSQT
RHVDQDLYVYEREGNLLVGLNRASGQRSKWVPTSWTNQTLNDYSGNAGNISTNGDSWVQI
TVPATSWVCYAPE
>BAD06002 alpha-amylase [Aspergillus awamori].
MMVAWWSLFLYGLQVAAPALAATPADWRSQSIYFLLTDRFARTDGSTTATCNTADQKYCG
GTWQGIIDKLDYIQGMGFTAIWITPVTAQLPQTTAYGDAYHGYWQQDIYSLNENYGTADD
LKALSSALHERGMYLMVDVVANHMGYDGAGSSVDYSVFKPFSSQDYFHPFCFIQNYEDQT
QVEDCWLGDNTVSLPDLDTTKDVVKNEWFDWVGSLVSNYSIDGLRIDTVKHVQKDFWPGY
NKAAGVYCIGEVLDGDPAYTCPYQNVMDGVLNYPIYYPLLNAFKSTSGSMDDLYNMINTV
KSDCPDSTLLGTFVENHDNPRFASYTNDIALAKNVAAFIILNDGIPIIYAGQEQHYAGGN
DPANREATWLSGYPTDSELYKLIASANAIRNYAISKDTGFVTYKNWPIYKDDTTIAMRKG
TDGSQIVTILSNKGASGDSYTLSLSGAGYTAGQQLTEVIGCTTVTVGSDGNVPVPMAGGL
PRVLYPTEKLAGSKICSSS
```

Figura 10. Saída Produzida pelo Código da Figura 9.

Interação na Web

Enquanto grande parte do processamento relacionado a bioinformática é feito em servidores potentes, a consulta, a anotação e o refinamento dos resultados costumam ser tarefas feitas por usuários que interagem com os dados produzidos via navegador Web.

Conceitos básicos de HTML

A maioria dos sites na Internet está escrita em HTML (*HyperText Markup Language*), uma linguagem de marcação que define os elementos básicos de um site, por exemplo, título, tabelas e figuras. A linguagem HTML é interpretada pelos navegadores Web permitindo uma fácil visualização de textos, imagens, sons e vídeos, além de possibilitar a navegação entre diferentes sites. Uma página (ou site) HTML é um conjunto de marcações (tags) e conteúdos. A Figura 11 representa uma página HTML extremamente simples. Nesta figura podemos observar algumas marcações como o cabeçalho (head) que possui um título (title) cujo conteúdo é "Olá mundo" e o corpo da página HTML delimitada pelas marcações body. Cada marcação é iniciada pelo seu nome dentro de sinais de menor e maior (<html>) e encerrada pela marcação com o mesmo nome porém com um barra (</html>), alternativamente, marcações que não possuam marcações internas podem ser iniciadas e encerradas com uma única marcação: <nome_da_marcação ... />. A marcação META tem objetivo, nesta página, de informar qual será a codificação e conteúdo da página. Neste caso o conteúdo é texto/html e a codificação dos caracteres é a UTF-8 (codificação que permite que acentos sejam utilizados sem a necessidade de marcações especiais).

```
<html>
 <head>
  <META http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
  <title>Olá mundo!</title>
 </head>
 <body>
  <h2>Olá mundo!</h2>
  Uma página HTML extremamente simples.
 </body>
</html>
```

Figura 11. Página HTML simple.

Diversas tecnologias e linguagens de programação foram desenvolvidas para complementar a experiência na navegação Web possibilitando um nível de interação muito maior com o usuário, bem como permitir que diferentes tipos de processamento sejam feitos no navegador Web ou no servidor. Por exemplo, javascript, php e CGI.

Uso de CGI

CGI (Common Gateway Interface | http://tools.ietf.org/html/rfc3875) é um padrão que permite aos servidores Web compartilhar a responsabilidade da geração de páginas Web com scripts CGI. Em outras palavras, o servidor Web, ao atender uma requisição de um usuário para a visualização de uma página CGI, solicita que o script correspondente seja executado e a execução deste programa produzirá a página solicitada pelo usuário.

Para se utilizar CGI é necessário configurar seu servidor Web de forma que ele identifique alguns diretórios do computador como autorizados a executar scripts CGI. Um diretório tipicamente utilizado para este fim é o /var/www/cgi-bin.

CGI permite que diferentes linguagens de programação sejam usadas para a produção de páginas Web e interação com o usuário. Neste capítulo descreveremos apenas o uso de CGI juntamente com a linguagem Perl.

Scripts CGI escritos em Perl

Para se fazer um script CGI escrito em Perl, basta desenvolver o programa Perl de forma a imprimir todo o conteúdo HTML desejado, com uma única restrição: a primeira linha a ser impressa deverá informar o tipo de conteúdo que está sendo produzido, conforme código da Figura 12. Este código apresenta um script perl para gerar uma página HTML simples. A primeira linha do código informa que este código é um script perl e deverá ser interpretado pelo comando /usr/bin/perl; a segunda linha informa o tipo de conteúdo que está sendo produzido, sem esta linha ocorrerá o seguinte erro no servidor Web: “Premature end of script headers: ola_mundo.cgi”, considerando que nosso script chama-se ola_mundo.cgi.

```
#!/usr/bin/perl
print "Content-Type: text/html; charset=UTF-8\r\n\r\n";
print '<html>';
print '<head>';
print '<title>Olá mundo!</title>';
print '</head>';
print '<body>';
print '<h2>Olá mundo!</h2>';
print 'Um programa CGI extremamente simples.';
print '</body>';
print '</html>';
```

Figura 12. Script Perl para a geração de HTML simples.

```
#!/usr/bin/perl
use CGI;
my $meuCGI = new CGI;
print $meuCGI->header(-charset=>"UTF-8");
print $meuCGI->start_html(-title=>"Olá mundo!");
print '<h2>Olá mundo!</h2>';
print 'Um programa CGI extremamente simples que usa a biblioteca CGI.';
print $meuCGI->end_html;
```

Figura 13. Script Perl para a geração de HTML usando a biblioteca CGI.

A linguagem de programação Perl, assim como diversas outras linguagens, possui uma biblioteca para facilitar o desenvolvimento de scripts CGI. Com esta biblioteca o usuário não precisa se lembrar dos comandos ou tags específicas para a construção de uma página HTML e, mesmo usando a biblioteca, ainda poderá imprimir comandos HTML da mesma forma que faria sem o uso desta biblioteca. A Figura 13 apresenta o código equivalente ao da Figura FF2, porém utilizando A biblioteca CGI.

Recebendo e retornando informações básicas

Há diversos tipos de campos HTML feitos para o envio/recebimento de informações. Nesta seção será apresentado um pequeno exemplo que utiliza a maioria desses campos. Neste exemplo, o usuário será convidado a preencher um conjunto de campos de um formulário Web gerado e cujos dados serão processados por um script CGI. A Figura BB1 apresenta este script. A biblioteca CGI é usada para produzir o

conteúdo HTML dos diferentes campos do formulário (*textfield*, *textarea*, *checkbox*, *radio_group*, e *submit*), bem como para recuperar o valor desses campos. A condição da linha 7 do código verifica se o campo chamado *botao* não está definido. Se não estiver definido, o código das linhas 8 a 13 produzirá o formulário HTML, caso já esteja definido, o código das linhas 15 a 25 produzirá uma página HTML com o resultado do processamento das informações passadas pelo usuário. A Figura 14 apresenta o formulário a ser preenchido pelo usuário e a Figura 15 apresenta o resultado do processamento desse formulário.

No material suplementar deste livro você encontrará um script CGI Perl que produz um formulário HTML. Esse formulário será capaz de receber um arquivo contendo uma lista de identificadores de sequência, copiar esse arquivo localmente, buscar no GenBank cada uma das sequências e imprimir seus dados no formato FASTA.

Permitindo interação Web com um banco de dados

É possível utilizar informações recebidas por scripts CGI Perl para consultar ou atualizar o banco de dados. Para isto, basta combinarmos o recebimento de informações de formulários HTML com o uso de bancos de dados dentro de um script Perl.

No material suplementar deste livro você encontrará um script CGI Perl que produz um formulário HTML que permite ao usuário escolher qual termo procurar no banco de dados.

```
1  #!/usr/bin/perl
2  use CGI;
3  my $meuCGI = new CGI;
4  print $meuCGI->header(-charset=>"UTF-8");
5  print $meuCGI->start_html(-title=>"Formulágio Perl CGI");
6  print $meuCGI->start_multipart_form;
7  if (!defined($meuCGI->param('botao'))){
8   print '<h2>Olá, favor preencher o formulário e pressionar "Ok".</h2>';
9   print "Nome: ", $meuCGI->textfield(-name => 'nome', -size => 25, -maxlength => 25);
10  print "<br>Endereço: ", $meuCGI->textarea(-name => 'endereco', -value => 'Preencha aqui seu endereço.', -cols => 50, -rows => 4);
11  print "<br>Você é professor: ", $meuCGI->checkbox(-name => 'ehProfessor', -label => 'sim', -checked => 0);
12  print "<br>Sexo: ", $meuCGI->radio_group(-name => 'sexo', -values => ['feminino', 'masculino'], -default => 'feminino', -columns => 1, -rows => 2);
13  print "<center>", $meuCGI->submit(-name => 'botao', -value => 'Ok');
14 }else{
15  my $textoProfessor = "";
16  if ($meuCGI->param('ehProfessor') eq 'on'){;
17    if ($meuCGI->param('sexo') eq 'feminino'){
18                    $textoProfessor = "professora ";
19    }else{
20      $textoProfessor = "professor ";
21    }
22  }
23  print '<h2>Formulário preenchido corretamente.</h2>';
24  print "Obrigado $textoProfessor", $meuCGI->param('nome'), ".";
25  print "<br>Endereco: ", $meuCGI->param('endereco');
26 }
27 print $meuCGI->end_form;
28 print $meuCGI->end_html;
```

Figura 14. Script Perl para a geração e processamento de um formulário HTML.

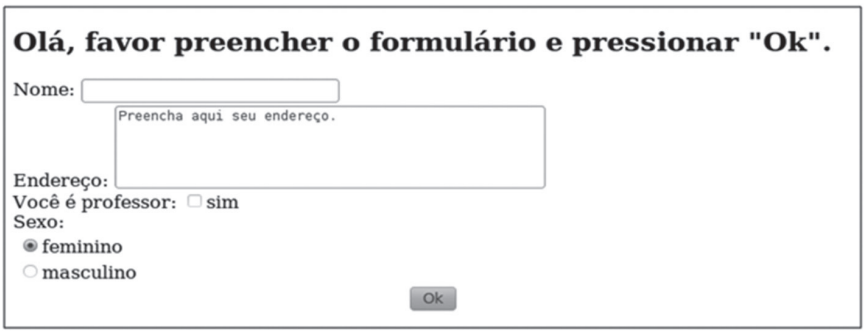


Figura 15. Formulário HTML produzido pelo script da Figura 14.

Figura 16. Página HTML com o resultado do processamento dos dados.

Bibliografias

ALTSCHUL, S; GISH, W; MILLER, W; MYERS, E; LIPMAN, D. *Basic local alignment search tool*. Journal of Molecular Biology. 1990. 215 (3): 403–410.

CHROMATIC. *Modern Perl*. Onyx Neon Press, pp. 2004, 2012.

BIOPERL. QUICK START, 2013. Disponível em http://www.bioperl.org/wiki/Quick_start . Acessado em 24/02/2013

MUSCIANO, C.; KENNEDY, B. *HTML & XHTML: The Definitive Guide, 6th Edition*. O'Reilly Media, pp. 680, 2009.

MySQL. MySQL 5.6 Reference Manual, 2013. Disponível em http://dev.mysql.com/doc/refman/5.6/en/ . Acessado em 24/02/2013

POE, C.. *Beginning Perl*. Wrox, pp. 744, 2012.

WEINMAN , W.E.. *The CGI Book*. New Riders Pub, pp. 304, 1996.