

**UNIVERSIDADE DE SÃO PAULO (USP)**  
**INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO (ICMC)**  
**DEPARTAMENTO DE CIÊNCIAS DE COMPUTAÇÃO (SCC)**

Bacharelado em Ciências de Computação  
Disciplina de Organização de Arquivos (SCC0215)  
Prof<sup>a</sup>. Dra. Cristina Dutra de Aguiar Ciferri

# **Primeira Parte do Trabalho Prático (Parte I)**

## **ALUNOS**

David Souza Rodrigues (nº USP 4461180)  
Fernanda Tostes Marana (nº USP 4471070)  
Gabriel Toschi de Oliveira (nº USP 9763039)  
Marcelo de Moraes Carvalho da Silva (nº USP 9791048)

## **DATA DE ENTREGA**

09 de maio de 2017

# ÍNDICE

<b>ÍNDICE</b>	<b>1</b>
<b>ESPECIFICAÇÃO DO TRABALHO</b>	<b>2</b>
<b>DESCRIÇÃO E ORGANIZAÇÃO DOS CAMPOS</b>	<b>2</b>
<b>MODULARIZAÇÃO</b>	<b>4</b>
<b>ORGANIZAÇÃO DOS REGISTROS</b>	<b>5</b>
Registros com indicador de tamanho	5
Registros com delimitadores	6
Registros com número fixo de campos	6
<b>TRATAMENTO DE TEXTO</b>	<b>6</b>
Conversão entre diferentes codificações de caracteres	7
Comparação de strings com caracteres acentuados	8
<b>INTERFACE DO PROGRAMA</b>	<b>9</b>
<b>COMPILAÇÃO E INSTRUÇÕES DE USO</b>	<b>9</b>
<b>TELAS DA INTERFACE</b>	<b>10</b>
<b>BATERIAS DE TESTE</b>	<b>13</b>
Configuração 1: registros organizados com indicadores de tamanho	14
Configuração 2: registros organizados com delimitadores	14
Configuração 3: registros organizados com número fixo de campos	14
Bateria 1: mostrar todos os registros	14
Bateria 2: buscar registro por um campo	15
Bateria 3: buscar registro por RRN	16
Bateria 4: buscar campo por RRN	16

## ESPECIFICAÇÃO DO TRABALHO

O Trabalho Prático realizado tem, como principal objetivo, o armazenamento e tratamento de dados em arquivos, segundo certos métodos de organização destes dados em campos e registros. Os dados utilizados são relativos a domínios governamentais de internet, registrados no sistema do Registro.br (e disponíveis no endereço eletrônico <http://dados.gov.br/dataset/dominios-gov-br>).

Cada registro representa um domínio governamental de internet e seus dados são divididos em diferentes campos, de tamanho variável (domínio, nome, cidade e UF) e de tamanho fixo (documento, data e hora de cadastro, data e hora da última atualização e ticket de cadastro). Seguindo a especificação deste Trabalho Prático, os campos variáveis foram organizados usando indicadores de tamanho.

Os registros, por sua vez, foram manipulados usando três métodos diferentes de organização: indicadores de tamanho, delimitadores e número fixo de campos. É importante exaltar que a implementação de cada um desses métodos foi feita de forma separada e imiscível das outras, dando a opção ao usuário qual dos métodos escolher, antes do início da execução do programa.

Por fim, implementou-se também uma interface, em modo texto, para interagir com o usuário e permitir que as funções pudessem ser utilizadas. Foram definidas e implementadas as seguintes operações: (1) leitura de um arquivo de entrada, em modo texto, com vários registros e o posterior armazenamento deles de forma organizada em um arquivo binário; (2) recuperação de todos os registros do arquivo binário e impressão deles na saída padrão; (3) recuperação de registros que tenham um critério de busca, escolhido pelo usuário; (4) recuperação de todos os dados de um registro pelo seu RRN, definido pelo usuário; (5) recuperação de um campo de um registro específico pelo seu RRN, definido pelo usuário.

Vários aspectos já citados sobre a implementação desse Trabalho Prático serão comentados e explanados mais a fundo nas próximas seções desta documentação.

## DESCRIÇÃO E ORGANIZAÇÃO DOS CAMPOS

A especificação desse Trabalho Prático indica a utilização de um registro que contém todas as informações sobre um domínio governamental de internet, com oito campos, nos quais quatro deles tem tamanho fixo, e os outros quatro são de tamanho variável. Um arquivo de entrada foi disponibilizado juntamente com a especificação, com 200 registros, para ser lido pelo programa implementado. Os campos de tamanho fixo presentes no registro são (o caractere de fim de string — '\0' — também é armazenado com todos os campos de tamanho fixo, exceto pelo ticket):

- **documento:** o CNPJ do órgão/entidade a qual o domínio está vinculado; como o CNPJ segue sempre uma estrutura fixa ("###.###.###-###/##", sendo cada # um algarismo numérico), seu tamanho foi fixado em 20 bytes;
- **dataHoraCadastro:** a data e a hora de cadastro do domínio no sistema; essa marca temporal é apresentado sempre em uma estrutura fixa ("DD/MM/AAAA hh:mm:ss", sendo, os algarismos, D, para dia, M, para mês, A, para ano, h, para hora, m, para minuto e s, para segundo) e, portanto, seu tamanho foi fixado em 20 bytes;
- **dataHoraAtualiza:** a data e a hora da última atualização feita no registro; essa marca temporal é apresentada com a mesma estrutura fixa do campo dataHoraCadastro, e, logo, seu tamanho também foi fixado em 20 bytes;

- **ticket:** número de identificação vinculado ao registro do domínio no sistema do Registro.br; como é um campo numérico, ele foi armazenado como um número inteiro, e, portanto, tem tamanho fixo de 4 bytes.

Por fins de otimização, os tamanhos fixos destes campos foram considerados constantes na implementação do programa e não foi preciso armazenar tais valores no arquivo. Por sua vez, os campos de tamanho variável presentes no registro são (o caractere de fim de string — '\0' — também é armazenado com todos os campos de tamanho variável):

- **dominio:** a URL cadastrada no sistema do Registro.br;
- **nome:** nome do órgão/entidade a qual o domínio está vinculado;
- **cidade:** cidade onde está localizado o órgão/entidade a qual o domínio está vinculado;
- **UF:** Unidade Federativa onde está localizado o órgão/entidade a qual o domínio está vinculado;

Para que seja possível a recuperação dos campos de tamanho variável e seguindo a especificação deste Trabalho Prático, foi usado o método de indicadores de tamanho para organizá-los em um registro. Essa técnica consiste em armazenar, à frente do campo de tamanho variável, a quantidade de bytes ocupada pelo campo, na forma de um número inteiro. Assim, é possível recuperar todos esses campos corretamente.

No arquivo de entrada disponibilizado, os campos estão dispostos na ordem: dominio, documento, nome, UF, cidade, dataHoraCadastro, dataHoraAtualiza, ticket; cada campo está separado por pontos e vírgulas (;).

Entretanto, na hora da criação de um registro para ser escrito no arquivo binário, a ordem dos campos é alterada por fins de otimização: todos os campos de tamanho fixo são colocados no início do registro, e os campos de tamanho variável ficam ao final do registro. Tal processo é feito para permitir a busca sequencial de forma mais rápida, já que é possível “pular” todos os campos de tamanho fixo apenas atualizando o ponteiro do arquivo, ao invés de acessar campo a campo.

A ordem final dos campos dentro de um registro é: documento, dataHoraCadastro, dataHoraAtualiza, ticket, dominio, nome, cidade, UF. A Tabela 1 mostra a estrutura de um registro, a partir de uma abstração de seus dados principais (a leitura deve ser feita da esquerda para a direita, de cima para baixo). Cada quadrado da tabela representa 1 byte e a quantidade de bytes de cada campo de tamanho variável tem apenas fins ilustrativos.

**Tabela 1: representação de um registro único, com os campos de tamanho variável, seus indicadores de tamanho e os campos de tamanho fixo**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
#	#	#	.	#	#	#	.	#	#	#	-	#	#	#	#	/	#	#	\0

20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
D	D	/	M	M	/	A	A	A	A		h	h	:	m	m	:	s	s	\0

40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

D	D	/	M	M	/	A	A	A	A		h	h	:	m	m	:	s	s	\0
---	---	---	---	---	---	---	---	---	---	--	---	---	---	---	---	---	---	---	----

60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	
ticket				tam. domínio				domínio (tamanho de exemplo)								\0	tam. nome			

80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99
nome (tamanho de exemplo)							\0	tam. cidade				cidade (tamanho de exemplo)							\0

100	101	102	103	104	105	106	107	108	109	110	111	
tam. UF						UF (tamanho de exemplo)						\0

Na Tabela 1, os campos estão sendo representados nos intervalos de bytes:

- **0 a 19:** documento, com o caractere de fim de string;
- **20 a 39:** dataHoraCadastro, com o caractere de fim de string;
- **40 a 59:** dataHoraAtualiza, com o caractere de fim de string;
- **60 a 63:** ticket, em formato de número inteiro;
- **64 a 67:** indicador de tamanho do campo domínio;
- **68 a 75:** domínio, com o caractere de fim de string;
- **76 a 79:** indicador de tamanho do campo nome;
- **80 a 87:** nome, com o caractere de fim de string;
- **88 a 91:** indicador de tamanho do campo cidade;
- **92 a 99:** cidade, com o caractere de fim de string;
- **100 a 103:** indicador de tamanho do campo UF;
- **104 a 111:** UF, com o caractere de fim de string.

## MODULARIZAÇÃO

De acordo com a natureza da especificação deste Trabalho Prático, foi possível colocar a modularização do código como um dos pilares principais de desenvolvimento. Ao todo, são sete bibliotecas diferentes que funcionam juntas durante a execução do programa.

O principal motivo da modularização foi separar as rotinas que eram usadas a um único registro isolado e as que eram usadas em cada uma dos métodos de organização de vários registros em um arquivo binário. Como o registro segue a mesma estrutura independentemente se são usados indicadores de tamanho, registradores ou número fixo de campos para organizá-los no arquivo, uma biblioteca especial foi criada para tratar deles.

As funções desta biblioteca, a `registro.h`, tem funções para criar um registro a partir de dados de entrada, mapear onde cada campo começa em um registro, imprimir um registro na saída padrão, imprimir um campo de um registro na saída padrão e comparar um campo de um registro com uma chave de busca. Todas elas trabalham com um vetor de bytes e não acessam diretamente nenhum arquivo, de entrada ou de saída.

Para manipular os arquivos de saída, foram criadas outras três bibliotecas, `indicador.h`, `delimitador.h` e `numeroFixo.h`, respectivamente, para registros organizados por indicadores de

tamanho, delimitadores e número fixo de campos. Todas elas tem funções para inserir um registro (previamente criado usando a outra biblioteca) em um arquivo binário, buscar o próximo registro, a partir da posição atual de um ponteiro de arquivo, e buscar um registro por meio de seu RRN no arquivo de saída.

A integração entre essas quatro bibliotecas se dá por meio da própria interface do programa, condensada na biblioteca `interface.h`. Ela é a responsável por receber os dados de entrada, criar o registro usando uma biblioteca e inserí-la no arquivo de saída usando outra biblioteca/receber um registro lido no arquivo de saída usando uma biblioteca e imprimí-la na saída padrão com outra biblioteca. Essa modularização permite que novas organizações de registros sejam implementadas com poucas alterações no código atual.

Outras bibliotecas auxiliares também estão presentes no código-fonte do programa. A `CSV.h` é capaz de ler dados do arquivo de entrada e liberar a memória alocada para tal. Por sua vez, a biblioteca `caractere.h` tem funções relacionadas à entrada, saída e tratamento de texto, como limpar o retentor do teclado, ler uma string da entrada padrão, converter letras acentuadas para letras acentuadas maiúsculas ou, ainda, para outra codificação de caracteres.

## ORGANIZAÇÃO DOS REGISTROS

Seguindo a especificação do Trabalho Prático, foram considerados três métodos de organização de registros: (a) registros com indicador de tamanho; (b) registros com delimitadores; e (c) registros com número fixo de campos. Apenas um método é utilizado durante uma execução do programa e deve ser configurado pelo usuário ao abrir o software, por meio de uma opção mostrada na interface textual. As seguintes subseções explanam sobre a implementação de cada um dos métodos e de algoritmos de inserção/busca relacionados e eles.

### Registros com indicador de tamanho

Neste método, os registros de tamanhos variável são tratados com indicadores de tamanho antes de cada registro, ou seja, no começo de cada registro, é posto um valor inteiro que indica o tamanho total do registro, ou seja, o comprimento total de todos os campos fixos e variáveis do registro mais o tamanho ocupado pelos indicadores de tamanho dos campos fixos.

Foram implementadas três funções específicas para este método: uma função de inserção no arquivo, e outras duas utilizadas na busca de um registro.

A função de inserção (`insereRegistro_Indicador`) calcula o tamanho total do registro, escreve o valor obtido no arquivo como indicador de tamanho e, logo em seguida, escreve o registro no arquivo. Quanto às funções de busca, uma delas retorna um registro a partir da posição atual do ponteiro de arquivo (`buscaRegistro_Indicador`), enquanto a outra percorre desde o começo do arquivo até encontrar o RRN desejado (`buscaRRN_Indicador`).

Nota-se que a busca de um registro pelo seu RRN, usando indicadores de tamanho, é muito mais eficiente do que usando os outros métodos, uma vez que, apesar da busca ainda ser sequencial, não é necessário percorrer todos os bytes do registro que não é desejado. A partir do RRN, basta percorrer desde o início do arquivo, armazenar o indicador de tamanho do registro em uma variável e, caso ainda não esteja no registro desejado, levar o ponteiro do arquivo para o próximo indicador de tamanho, somando sua posição atual com o valor salvo na variável.

### Registros com delimitadores

Nesse método, os registros de tamanho variável são tratados com delimitadores entre registros, ou seja, ao final de cada registro, é posto um delimitador, que indica que o registro acabou. O delimitador utilizado foi o caractere de valor numérico -1, escolhido após testes, em que

foi constatado que outros valores, quando percorria-se o arquivo binário, apresentavam conflito com os números inteiros utilizados para identificar os tamanhos dos campos de tamanho variável.

Foram implementadas três funções específicas de registros com delimitadores, uma função de inserção no arquivo, e outras duas utilizadas na busca de um registro.

A função de inserção (`insereRegistro_Delimitador`) escreve o registro, previamente montado por outra rotina, no arquivo e, depois, escreve o delimitador. Quanto às funções de busca, uma delas retorna um registro a partir da posição atual do arquivo (`buscaRegistro_Delimitador`), enquanto a outra (`buscaRRN_Delimitador`) chama essa primeira função por vezes consecutivas até que se alcance o RRN desejado, ou até que se constate que o RRN desejado é maior do que o número de registros do arquivo; neste caso, a função retorna o valor NULL.

Para percorrer um registro no arquivo, a função de busca segue a seguinte sequência de passos: (1) pula os campos de tamanho fixo; (2) obtém o tamanho do próximo campo de tamanho variável; (3) pula o campo de tamanho variável; (4) confere o próximo caracter; (5) se o caracter for o delimitador de final de registro, acabou o registro e sai da função, caso contrário, volta para o passo (2).

## Registros com número fixo de campos

Na metodologia de número fixo de campos, nenhum dado extra é adicionado ao arquivo; já que o número de campos é fixo, a identificação do que é um registro, no arquivo binário, se dá após a leitura de todos os campos.

Foram implementadas três funções específicas de registros com número fixo de campos, uma função de inserção no arquivo, e outras duas utilizadas na busca de um registro.

A inserção de um registro no arquivo (`insereRegistro_NumeroFixo`) é bem simples e resume-se a escrever o registro diretamente, sem algum tipo de processamento prévio.

Durante a função de busca de um registro (`buscaRegistro_NumeroFixo`), a partir da posição atual do ponteiro de arquivo, primeiro é determinada a posição, no arquivo, do último campo de tamanho fixo do registro e, após, já tendo em vista o montante fixo de campos contidos no registro, utilizou-se um contador de campos até esse se igualar à quantidade de campos de tamanho variado, determinando assim o final do registro bem como seu tamanho, em bytes. Assim, é alocado uma sequência de bytes para conter o registro por completo.

A busca por RRN (`buscaRRN_NumeroFixo`) faz uso da função de busca já descrita para determinar o registro solicitado. Com o uso de um contador até o valor do RRN e enquanto não for o final do arquivo, a função de busca é chamada; caso seja o RRN requisitado, o registro é retornado ao fluxo do programa.

## TRATAMENTO DE TEXTO

Algoritmos de tratamento de texto foram necessários durante a implementação deste Trabalho Prático, por dificuldades em relação à presença de acentuação gráfica nos dados de entrada (símbolos, estes, não contemplados pela tabela ASCII padrão) e às diferentes codificações de caracteres utilizadas nos dados de entrada e no programa implementado. As seguintes subseções explanam sobre as implementações feitas para resolver estes problemas relacionados ao tratamento de texto.

### Conversão entre diferentes codificações de caracteres

Durante o desenvolvimento do programa, foram observadas incompatibilidades entre a codificação de caracteres utilizada pelo sistema operacional no qual o software estava sendo

implementado e a codificação de caracteres usado no arquivo de entrada de dados. A codificação de caracteres padrão utilizada pelo sistema operacional Ubuntu é a UTF-8, enquanto algumas tentativas mostraram que a codificação de caracteres utilizada pelo arquivo de entrada era a Windows-1252.

Por mais que a maioria dos caracteres (letras sem acentuação, números e sinais gráficos, como ponto e hífen) tivessem representação idêntica em ambas as codificações, os caracteres acentuados, tanto maiúsculos como minúsculos, eram representados de forma diferente entre eles. Dessa forma, tanto a impressão destes valores na entrada padrão quanto a comparação entre dados digitados pelo usuário e dados presentes originalmente no arquivo de entrada apresentavam problemas.

Como forma de resolver o problema, foram estudadas todas as representações de caracteres acentuados (além daqueles com outros sinais gráficos, como til e cedilha) presentes na língua portuguesa, nas duas codificações. A Tabela 2 apresenta os valores dos bytes de cada codificação, para cada caractere acentuado *ch*; cada valor é apresentado na tabela como mostrados por meio de um comando `printf("%d\n", valor)`, ou seja, como a representação padrão de um `signed char` na linguagem C. No caso da codificação de caracteres UTF-8, dois bytes são usados para representar cada caractere acentuado e estão separados por vírgulas na tabela.

**Tabela 2: representação dos caracteres acentuados da língua portuguesa nas codificações de caracteres UTF-8 e Windows-1252, no formato padrão de `signed char`, da linguagem C**

<i>ch</i>	UTF-8	Windows-1252	<i>ch</i>	UTF-8	Windows-1252
á	-61, -95	-31	Á	-61, -127	-63
é	-61, -87	-23	É	-61, -119	-55
í	-61, -83	-19	Í	-61, -115	-51
ó	-61, -77	-13	Ó	-61, -109	-45
ú	-61, -70	-6	Ú	-61, -102	-38
à	-61, -96	-32	À	-61, -128	-64
â	-61, -94	-30	Â	-61, -126	-62
ê	-61, -86	-22	Ê	-61, -118	-54
ô	-61, -76	-12	Ô	-61, -108	-44
ã	-61, -93	-29	Ã	-61, -125	-61
õ	-61, -75	-11	Õ	-61, -107	-43
ç	-61, -89	-25	Ç	-61, -121	-57

É possível ver na Tabela X que, para estes caracteres, a codificação UTF-8 tem o mesmo valor para o primeiro byte (-61), tornando a correspondência entre as duas codificações de um pra um, ou seja, considerando -61 como padrão, há um único valor numérico em UTF-8 para cada caractere em Windows-1252, também representado por um único valor numérico.



A partir destes dados, foi possível planejar uma simples rotina de substituição de valores que, implementada em conjunto com o algoritmo de leitura do arquivo de entrada, fosse capaz de converter os caracteres acentuados de uma codificação para outra. Entre as duas conversões possíveis, foi escolhido passar os caracteres acentuados do arquivo de entrada, codificados em Windows-1252, para a codificação UTF-8. Por mais que a UTF-8 use um byte a mais para representar cada caractere, o fato dele ser o padrão do sistema operacional usado durante a implementação e, além disso, ser padrão na maioria dos sistemas operacionais baseados em Linux — uma das restrições da especificação do Trabalho Prático era que o programa fosse compilado em Linux —, traria mais facilidade e comodidade ao usuário esperado e, portanto, foi escolhido como padrão.

Essa conversão é realizada a partir de uma subrotina aplicada no algoritmo de leitura do arquivo de entrada (`leCSV`, na biblioteca `CSV.h`) e de uma rotina auxiliar (`acentowin1252_UTF8`, na biblioteca `caractere.h`), para retornar o valor em UTF-8 de um caractere acentuado em Windows-1252. A cada caractere lido do arquivo de entrada, ele passa pela rotina auxiliar que tenta encontrar o correspondente UTF-8 dele; caso encontre, um espaço extra é alocado na memória para colocar o primeiro byte padrão e, depois dele, o novo valor encontrado para o caractere.

Dessa forma, todos os dados de entrada são mantidos no mesmo padrão dos dados inseridos pelo usuário, mantendo a consistência do sistema e tornando dispensáveis configurações extras na máquina do usuário, por exemplo.

## Comparação de strings com caracteres acentuados

Em uma das funções do programa desenvolvido, o usuário pode digitar um critério de busca, escolher um dos campos do registro e o programa imprime, na saída padrão, todos os registros no qual aquele campo seja igual ao critério de busca digitada. Letras maiúsculas e minúsculas, nesse escopo, podem mostrar-se como um problema; nesta implementação, decidiu-se que dois termos como “BaHla” e “bAHlA”, por exemplo, deviam ser equivalentes durante a operação de busca. Para tal, antes de qualquer comparação entre strings neste escopo, todas as letras deviam ser colocadas em caixa alta, a partir da função `toupper`, da biblioteca `cctype.h`. No exemplo anteriormente mostrado, ambos os termos ficariam no formato “BAHIA” e, portanto, seriam considerados iguais.

Entretanto, essa função padrão da linguagem C não é capaz de tratar caracteres acentuados. Como decisão de implementação, foi decidido que os caracteres acentuados seriam considerados diferentes das letras equivalentes sem acentos — termos como “São Paulo” e “Sao Paulo” são considerados diferentes, segundo esse método. Desta forma, a questão se resume a colocar os caracteres acentuados também em caixa alta, para fazer as comparações.

Tornando a codificação UTF-8 como padrão e utilizando os dados presentes na Tabela X, foi possível ver que a diferença numérica entre os valores de uma letra acentuada minúscula e maiúscula é sempre -32. Logo, para colocar um caractere acentuado em caixa alta, basta somar -32 ao seu valor numérico. Tal rotina é realizada para todo um termo pela função `stringMaiusculaAcentos`, da biblioteca `caractere.h`. Assim, a comparação se faz como pretendido, mesmo com letras acentuadas; “bRasÍLIA” e “BrASÍLia” seriam consideradas iguais, já que, após o processamento, ambas ficariam como “BRASÍLIA”.

## INTERFACE DO PROGRAMA

O programa deste Trabalho Prático consiste, além das rotinas para manipular os arquivos com registros e campos, uma interface simples para que o usuário acesse as funções

implementadas, em modo texto (para ser acessada dentro de um terminal ou prompt de comando). Além disso, ela permite que o usuário escolha o método de organização de registros e escolha o arquivo que servirá como entrada de dados para o programa. Ambas as duas configurações são feitas no começo da execução e são obrigatórias, ou seja, não é possível executar o software sem que sejam realizadas.

Ao escolher o método de organização de registros, a configuração é finalizada ao abrir um arquivo para armazenar os dados na forma binária, além de selecionar as funções corretas para manipular os registros, por meio de atribuições à ponteiros de função. Desta forma, o código torna-se menos repetitivo, mais elegante e mais compreensível.

Após as configurações, o menu do programa é aberto e o usuário pode escolher qual função realizar, dentre as possíveis: (a) “Mostrar todos os registros”; (b) “Buscar um registro por um campo”; (c) “Buscar um registro por identificação numerica”; (d) “Buscar um campo por identificação numerica”; ou (e) “Sair do programa”.

Ao escolher a opção (a), todos os registros do programa serão listados, um a um, de uma forma organizada e compreensível. O usuário deve apertar Enter para passar para o próximo registro, até que eles acabem.

A opção (b), primeiramente, pergunta qual campo será usado na busca e, em seguida, pede que o usuário digite a chave de busca que ele deseja. Os resultados da busca também serão listados, um a um, e o usuário deve apertar Enter para passar para o próximo registro. Caso nenhum registro seja encontrado durante a busca, uma mensagem é mostrada ao usuário. Marcas de tempo não podem ser usadas como chave de busca, por decisão de projeto.

A opção (c) pede uma identificação numérica, ou seja, o RRN do registro a ser buscado e, caso aquele RRN seja válido — ou seja, um número inteiro positivo e menor que o maior RRN disponível no arquivo —, o registro é mostrado na tela. Ao contrário, uma mensagem de erro é mostrada ao usuário. A opção (d) funciona de forma semelhante mas, antes da busca, pergunta qual campo deve ser mostrado na tela, já que imprime apenas o campo escolhido pelo usuário e não todo o registro.

Para todas as opções acima e para as rotinas de configuração, nos momentos em que se pede uma entrada direta de dados do usuário, a validade dos dados são checados. Assim, digitar um número que não seja uma opção em um menu, digitar uma letra ao invés de um número ou digitar um nome de arquivo inexistente desencadeiam mensagens de erro e pedem uma nova entrada de dados.

A opção (e) termina o programa e desencadeia uma rotina de desalocar memórias auxiliares e fechamento de arquivos. Os dados no arquivo binário são apagados e reescritos a cada execução do programa.

## **COMPILAÇÃO E INSTRUÇÕES DE USO**

Esse Trabalho Prático foi implementado e testado no Ubuntu, sistema operacional baseado em Linux. Nenhum ambiente de desenvolvimento integrado (IDE) foi utilizado durante a implementação, usando apenas editores de textos voltados à programação, como VIM e Sublime Text.

Por isso, a compilação do programa foi feita usando diretamente o compilador GCC, na versão 5.4.0, a mesma versão recomendada para compilar o programa antes do uso. Um arquivo Makefile foi criado para facilitar a compilação do programa em sistemas operacionais baseados em UNIX e pode ser acessado a partir do comando “make”, no diretório onde o código-fonte foi descompactado. Caso não tenha compatibilidade com esse recurso, ele pode ser compilado com

o comando "gcc interface.c CSV.c registro.c indicador.c delimitador.c numeroFixo.c caractere.c -o dominios -I./".

A interface, por ser feita em modo texto, precisa ser rodado diretamente de um terminal ou um prompt de comando do sistema operacional do usuário, que esteja configurado para aceitar a codificação de caracteres UTF-8. A maioria dos sistemas operacionais Linux tem essa configuração como padrão. Caso o programa apresente caracteres estranhos no lugar das letras acentuadas, a configuração deve ser feita.

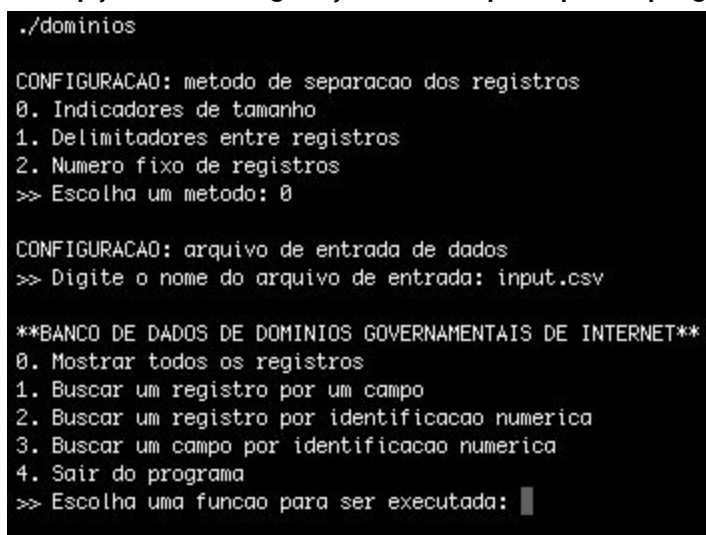
O programa pode ser executado, em sistemas operacionais baseados em UNIX, também com ajuda do Makefile, a partir do comando "make run". Caso o sistema não tenha compatibilidade com o recurso, o software é executado ao abrir o executável de nome "dominios".

Após o início do programa, a interface guiará por alguns passos de configuração e, após isso, o menu de funções estará disponível. É imprescindível que o arquivo de entrada esteja no mesmo diretório que o executável ou a configuração não poderá ser completada e o programa não será executado.

## TELAS DA INTERFACE

Esta seção apresenta capturas de tela da interface do programa, em diferentes situações. As entradas feitas pelo usuário são apenas para fins ilustrativos.

Figura 1: opções de configuração e menu principal do programa



```
./dominios

CONFIGURACAO: metodo de separacao dos registros
0. Indicadores de tamanho
1. Delimitadores entre registros
2. Numero fixo de registros
>> Escolha um metodo: 0

CONFIGURACAO: arquivo de entrada de dados
>> Digite o nome do arquivo de entrada: input.csv

**BANCO DE DADOS DE DOMINIOS GOVERNAMENTAIS DE INTERNET**
0. Mostrar todos os registros
1. Buscar um registro por um campo
2. Buscar um registro por identificacao numerica
3. Buscar um campo por identificacao numerica
4. Sair do programa
>> Escolha uma funcao para ser executada: █
```

Figura 2: função para mostrar todos os registros do arquivo

```
>> Escolha uma funcao para ser executada: 0

DOMINIO: eletrobras.gov.br (ticket 5023)
Cadastro feito em 09/01/1996 12:00:00
Ultima atualizacao em 02/07/2014 19:56:36

ORGAO/ENTIDADE:
CENTRAIS ELETRICAS BRASILEIRAS SA
Rio de Janeiro (Rio de Janeiro)
CNPJ: 000.001.180/0002-07

Aperte ENTER para mostrar o proximo registro...

DOMINIO: premioconservacaoenergia.gov.br (ticket 5379)
Cadastro feito em 24/03/2009 20:07:14
Ultima atualizacao em 15/04/2013 21:36:17

ORGAO/ENTIDADE:
CENTRAIS ELETRICAS BRASILEIRAS SA
Rio de Janeiro (Rio de Janeiro)
CNPJ: 000.001.180/0002-07

Aperte ENTER para mostrar o proximo registro...

DOMINIO: procel.gov.br (ticket 453438)
Cadastro feito em 06/11/2000 18:41:03
Ultima atualizacao em 05/10/2009 18:54:36

ORGAO/ENTIDADE:
CENTRAIS ELETRICAS BRASILEIRAS SA
Rio de Janeiro (Rio de Janeiro)
CNPJ: 000.001.180/0002-07

Aperte ENTER para mostrar o proximo registro...

DOMINIO: proinfra.gov.br (ticket 784297)
Cadastro feito em 18/04/2002 14:55:25
Ultima atualizacao em 15/04/2013 21:36:41

ORGAO/ENTIDADE:
CENTRAIS ELETRICAS BRASILEIRAS SA
Rio de Janeiro (Rio de Janeiro)
CNPJ: 000.001.180/0002-07
```

Figura 3: busca de registro por chave de busca

```
**BANCO DE DADOS DE DOMINIOS GOVERNAMENTAIS DE INTERNET**
0. Mostrar todos os registros
1. Buscar um registro por um campo
2. Buscar um registro por identificacao numerica
3. Buscar um campo por identificacao numerica
4. Sair do programa
>> Escolha uma funcao para ser executada: 1

0. Numero do documento
1. Ticket de cadastro
2. Dominio
3. Nome do(a) orgao/entidade
4. Cidade
5. UF
>> Escolha que campo vai ser usado como critério de busca: 4

>> Digite o que deve ser buscado: Resende
DOMINIO: inb.gov.br (ticket 89158)
Cadastro feito em 17/03/1998 19:26:24
Ultima atualizacao em 27/01/2017 14:24:11

ORGAO/ENTIDADE:
Industrias Nucleares do Brasil S/A
Resende (Rio de Janeiro)
CNPJ: 000.322.818/0020-93

Aperte ENTER para mostrar o proximo resultado da busca...
```

Figura 4: busca de registro por RRN e busca de campo por RRN

```
**BANCO DE DADOS DE DOMINIOS GOVERNAMENTAIS DE INTERNET**
0. Mostrar todos os registros
1. Buscar um registro por um campo
2. Buscar um registro por identificacao numerica
3. Buscar um campo por identificacao numerica
4. Sair do programa
>> Escolha uma funcao para ser executada: 2
>> Digite a identificacao numerica do registro: 0

DOMINIO: eletrobras.gov.br (ticket 5023)
Cadastro feito em 09/01/1996 12:00:00
Ultima atualizacao em 02/07/2014 19:56:36

ORGAO/ENTIDADE:
CENTRAIS ELETRICAS BRASILEIRAS SA
Rio de Janeiro (Rio de Janeiro)
CNPJ: 000.001.180/0002-07

**BANCO DE DADOS DE DOMINIOS GOVERNAMENTAIS DE INTERNET**
0. Mostrar todos os registros
1. Buscar um registro por um campo
2. Buscar um registro por identificacao numerica
3. Buscar um campo por identificacao numerica
4. Sair do programa
>> Escolha uma funcao para ser executada: 3
>> Digite a identificacao numerica do registro: 0

0. Numero do documento
1. Data e hora do cadastro
2. Data e hora da ultima atualizacao
3. Ticket de cadastro
4. Dominio
5. Nome do(a) orgao/entidade
6. Cidade
7. UF
>> Escolha um campo a ser buscado: 1

Cadastro feito em 09/01/1996 12:00:00
```

## BATERIAS DE TESTE

As próximas subseções apresentam possíveis configurações para o programa e baterias de teste, com entradas e saídas esperadas. Devido aos três métodos de organização de registros, são três configurações possíveis, que mudam o funcionamento interno do programa, mas não alteram os resultados mostrados pela interface. Assim, é possível usar qualquer uma das três configurações em qualquer bateria de teste.

Todas as entradas e saídas descritas nessa seção não mostram o texto mostrado pela interface, para evitar redundâncias e prolixidade. Frases inseridas entre os símbolos '<' e '>' indicam comentários nas baterias para o usuário, como funções que devem ser repetidas várias vezes ou explicações sobre as saídas esperadas.

Em todas as configurações, é usado o arquivo "input.csv" como entrada dos dados. Ele corresponde ao arquivo disponibilizado juntamente com a especificação deste Trabalho Prático e, no contexto das baterias de teste, está no mesmo diretório do executável.

## Configuração 1: registros organizados com indicadores de tamanho

### Entrada

```
0
input.csv
```

## Configuração 2: registros organizados com delimitadores

### Entrada

```
1
input.csv
```

## Configuração 3: registros organizados com número fixo de campos

### Entrada

```
2
input.csv
```

## Bateria 1: mostrar todos os registros

### Entrada

```
0
<aperte Enter até os registros acabarem>
4
```

### Saída esperada

```
<primeiro registro>
DOMINIO: eletrobras.gov.br (ticket 5023)
Cadastro feito em 09/01/1996 12:00:00
Ultima atualizacao em 02/07/2014 19:56:36

ORGAO/ENTIDADE:
CENTRAIS ELETRICAS BRASILEIRAS SA
Rio de Janeiro (Rio de Janeiro)
CNPJ: 000.001.180/0002-07

<...>

<último registro>
DOMINIO: serac4.gov.br (ticket 2228294)
Cadastro feito em 12/07/2005 16:00:47
Ultima atualizacao em 30/11/2006 13:18:46

ORGAO/ENTIDADE:
MAER QUARTO SERVIÇO REGIONAL DE AVIAÇÃO CIVIL
```

São Paulo (São Paulo)  
CNPJ: 000.394.429/0086-08

## Bateria 2: buscar registro por um campo

### Entrada

1  
<pesquisa por número de documento>  
0  
000.119.784/0001-71

### Entrada

1  
<pesquisa por ticket>  
1  
7629594

### Entrada

1  
<pesquisa por dominio>  
2  
cfmv.gov.br

### Entrada

1  
<pesquisa por nome do órgão/entidade>  
3  
CONSELHO FEDERAL DE MEDICINA VETERINARIA

### Saída esperada

DOMINIO: cfmv.gov.br (ticket 7629594)  
Cadastro feito em 29/12/2010 12:54:29  
Ultima atualizacao em 06/01/2017 01:06:21  
  
ORGAO/ENTIDADE:  
CONSELHO FEDERAL DE MEDICINA VETERINARIA  
Brasília (Distrito Federal)  
CNPJ: 000.119.784/0001-71



## Bateria 3: buscar registro por RRN

### Entrada

<opção por RRN>  
2  
<valor do RRN>  
3

### Saída esperada

DOMINIO: proinfa.gov.br (ticket 784297)  
Cadastro feito em 18/04/2002 14:55:25  
Ultima atualizacao em 15/04/2013 21:36:41  
  
ORGAO/ENTIDADE:  
CENTRAIS ELETRICAS BRASILEIRAS SA  
Rio de Janeiro (Rio de Janeiro)  
CNPJ: 000.001.180/0002-07

### Entrada

<opção por RRN>  
2  
<valor do RRN>  
30

### Saída esperada

DOMINIO: cfmv.gov.br (ticket 7629594)  
Cadastro feito em 29/12/2010 12:54:29  
Ultima atualizacao em 06/01/2017 01:06:21  
  
ORGAO/ENTIDADE:  
CONSELHO FEDERAL DE MEDICINA VETERINARIA  
Brasília (Distrito Federal)  
CNPJ: 000.119.784/0001-71

## Bateria 4: buscar campo por RRN

### Entrada

<opção campo por RRN>  
3  
<valor do RRN>  
3  
<campo pesquisado: documento>

0

### Saída esperada

Documento: 000.001.180/0002-07

### Entrada

<opção campo por RRN>  
3  
<valor do RRN>  
3  
<campo pesquisado: data e hora do cadastro>  
1

### Saída esperada

Cadastro feito em 18/04/2002 14:55:25

### Entrada

<opção campo por RRN>  
3  
<valor do RRN>  
3  
<campo pesquisado: data e hora da ultima atualização>  
2

### Saída esperada

Ultima atualizacao em 15/04/2013 21:36:41

### Entrada

<opção campo por RRN>  
3  
<valor do RRN>  
3  
<campo pesquisado: ticket>  
3

### Saída esperada

Ticket: 784297

### Entrada

<opção campo por RRN>  
3

<valor do RRN>  
3  
<campo pesquisado: dominio>  
4

#### **Saída esperada**

Dominio: proinfa.gov.br

#### **Entrada**

<opção campo por RRN>  
3  
<valor do RRN>  
3  
<campo pesquisado: nome do órgão/entidade>  
5

#### **Saída esperada**

Orgao/Entidade: CENTRAIS ELETRICAS BRASILEIRAS SA

#### **Entrada**

<opção campo por RRN>  
3  
<valor do RRN>  
3  
<campo pesquisado: cidade>  
6

#### **Saída esperada**

Cidade: Rio de Janeiro

#### **Entrada**

<opção campo por RRN>  
3  
<valor do RRN>  
3  
<campo pesquisado: UF>  
7

#### **Saída esperada**

UF: Rio de Janeiro