UNIVERSIDADE DE SÃO PAULO - USP

Instituto de Ciências Matemáticas e de Computação - ICMC

Bacharelado em Ciências de Computação

Disciplina Organização de Arquivos

Profa. Dra. Cristina Dutra de Aguiar Ciferri

Bruno Mendes da Costa (9779433)

Daniel Sivaldi Feres (9912275)

Leonardo Moreira Kobe (9778623)

Matheus Henrique Junqueira Saldanha (9763234)

1º TRABALHO PRÁTICO

PARTE 1

SÃO CARLOS

8 de Maio de 2017

SUMÁRIO

1 Representação Interna dos Campos

Pode-se considerar que existem 3 tipos de dados sendo armazenados: CNPJ, data e texto. De forma abstrata, cada um desses 3 tipos possui 4 atributos:

a) representação textual, que é o formato textual pelo qual os valores contidos em um arquivo CSV são processados, e é o formato pelo qual exibir o dado em forma de texto para o usuário;

b) representação binária, que é a maneira pela qual cada tipo é armazenado no disco;

c) valor binário nulo, que é a representação binária que representa o valor nulo;

d) valor textual nulo, que é igual para todos os tipos de dado. Uma *string* vazia e a *string* "*null*" são ambas consideradas representações textuais de um valor nulo, sendo que a exibição para o usuário ocorre com a representação "*null"*.

A escolha da representação binária de cada tipo de dado teve como prioridade minimizar o espaço ocupado em disco, mas também foi levado em consideração a facilidade de manipulação dessas representações.

1.1 CNPJ

A representação textual é de 14 dígitos separados por símbolos, como em "01.851.771/0001-55". A sequência de dígitos e símbolos é exata e não pode ser alterada.

A representação binária é obtida retirando todos os símbolos, concatenando todos os dígitos e gerando um grande número *long int*. Ou seja, o CNPJ textual "01.851.771/0001-55" será convertido para o *long int* 1851771000155 para ser armazenado no disco. Portanto, o tamanho desse campo no disco é de 8 *bytes*.

O valor binário nulo é o valor -1.

1.2 Data

A representação textual é de 6 dígitos, e cada par de dígitos é separado por um caractere '/' (barra inclinada para frente), como em "30/05/97".

A representação binária é obtida convertendo cada par de dígitos em um número de 1 *byte* (tipo *char*). Com isso, tem-se um conjunto de 3 *bytes* representando o dia, mês e ano de uma data. O tamanho desse campo em disco é, então, de 3 *bytes*.

O valor binário nulo equivale à representação binária quando o dia da data é -1. Isto é, o primeiro *byte* da representação binária possui valor -1.

1.3 Texto

A representação textual equivale à cadeia de caracteres do texto.

A representação binária é a cadeia de caracteres desprovida do terminador '\0' (barra zero). Como o delimitador de campos é o caractere '\0', é possível ter a impressão de que ele faz parte da representação binária, o que não é verdade. Porém, escolher esse delimitador foi conveniente justamente por facilitar a leitura de texto do arquivo diretamente para a sua representação textual.

O valor binário nulo é uma cadeia vazia de caracteres; ou seja, contém 0 *byte* de tamanho. No arquivo, o campo que contém texto nulo possui apenas o delimitador.

O tamanho desse campo no arquivo é variável, como requerido nas especificações. O tamanho em *bytes* equivale ao número de caracteres no texto sendo armazenado, mais o delimitador de campo variável que ocupa 1 *byte*.

2 Organização dos Registros

Conforme foi pedido nas especificações do trabalho, os campos são separados por delimitadores. Decidiu-se por não separar os campos de tamanho fixo com delimitadores, porque o tamanho sendo sempre o mesmo torna dispensável um método que indique o seu final. Os campos de tamanho variável, por serem todos do tipo texto, são delimitados pelo *byte* de valor zero, que é conveniente por dois motivos: é um caractere inválido, portanto não existe ambiguidade entre delimitador e um dado; e é o *byte* necessário para terminar quaisquer *strings* na linguagem C, então após ler o valor do campo mais o delimitador, tem-se uma *string* pronta para uso.

Também de acordo com as especificações, são oferecidas ao usuário do programa as 3 organizações de registros dadas, seguindo a restrição de usar cada organização de forma independente, sem mesclá-las.

2.1 Número Fixo de Campos

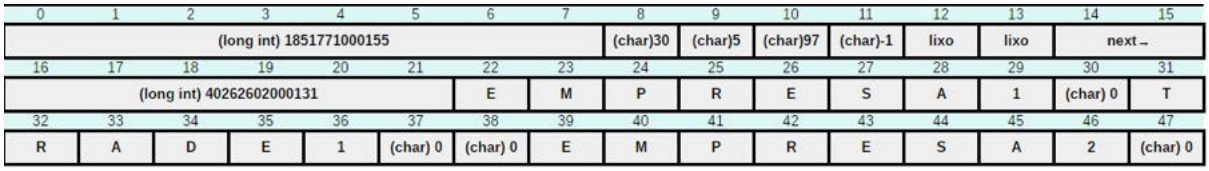
Nessa organização, todos os campos fixos são armazenados no início do registro, para permitir o avanço do ponteiro de arquivo através deles com apenas uma chamada à função *fseek*. Por isso, os campos são armazenados na ordem em que são apresentados na Tabela 1.

No armazenamento de registro, todos os campos são armazenados, mesmo que sejam nulos. Quando algum campo é nulo, a representação nula para o seu tipo de dado é armazenada, como dito na Seção 1.

Tabela 1: Exemplo de registro.

CNPJ	01.851.771/0001-55
Data de Registro	30/05/97
Data de Cancelamento	*null*
CNPJ do Auditor	40.262.602/0001-31
Nome Social	EMPRESA1
Nome Fantasia	TRADE1
Motivo do Cancelamento	*null*
Nome do Auditor	EMPRESA2

Figura 1: Armazenamento seguindo a organização “número fixo de campos”.



Na Figura 1 é mostrada a maneira pela qual o registro exibido na Tabela 1 seria armazenado em disco seguindo a organização em questão.

No *byte* 11, como o campo “Data de Cancelamento” é nulo, colocou-se um “(*char*) -1” como representando o dia dessa data, que é a representação nula definida para esse tipo de dado. Nada é escrito nos *bytes* que representam o mês e o ano, por isso contém lixo.

No *byte* 38, onde se inicia o campo “Motivo do Cancelamento”, colocou-se apenas o delimitador de campo “(*char*) 0”, que equivale a uma *string* vazia, a qual é a representação nula definida para o tipo texto.

2.2 Registros Separados por Delimitador

Para essa organização de registro, assumiu-se obrigatório o uso de número variável de campos, porque as especificações ditam que não é permitido misturar as diferentes organizações de registros dadas, e número fixo de campos é uma delas. Por conta disso, valores nulos não são armazenados no arquivo, o que trouxe a necessidade de algum método que identifique o campo que está sendo lido.

Para resolver o problema de identificação dos campos, um número de 1 *byte*, que pode ser considerado um *tag*, é armazenado antes de todos os campos que são escritos no arquivo. Esse *byte* contém o número do campo escrito, que pode assumir valores entre 0 e 7.

A escolha do delimitador de registros foi influenciada pela maneira como os registros são percorridos, que é por meio da repetição desses dois passos: ler um *byte*, que identifica o campo; e depois ler o campo. No primeiro passo, percebeu-se que seria possível detectar o fim do registro se o *byte* identificador de campo contivesse um valor inválido. Portanto, o delimitador escolhido para separar registros foi um *byte* contendo o número 8.

Figura 2: Armazenamento seguindo a organização “registros separados por delimitador”.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
(char)0	(long int) 1851771000155									(char) 1	(char) 30	(char) 5	(char) 97	(char)3	next ...
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
(long int) 40262602000131							(char)4	E	M	P	R	E	S	A	1
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
(char)0	(char)5	T	R	A	D	E	1	(char)0	(char)7	E	M	P	R	E	S
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
A	2	(char)0	(char)8												

A Figura 2 mostra a maneira pela qual o registro exibido na Tabela 1 seria armazenado em disco seguindo a organização de registros em questão.

No *byte* 0, “(*char*) 0” é a primeira *tag*, indicando que o primeiro campo, “CNPJ da Empresa”, será lido. Nos *bytes* 10, 14, 23, 33, 41 também há *tags*. Nota-se que os campos que têm valor nulo não estão presentes no registro.

O delimitador de registro é representado por “(*char*) 8” no *byte* 51, e os delimitadores dos campos de tamanho variável estão na tabela como “(*char*) 0”.

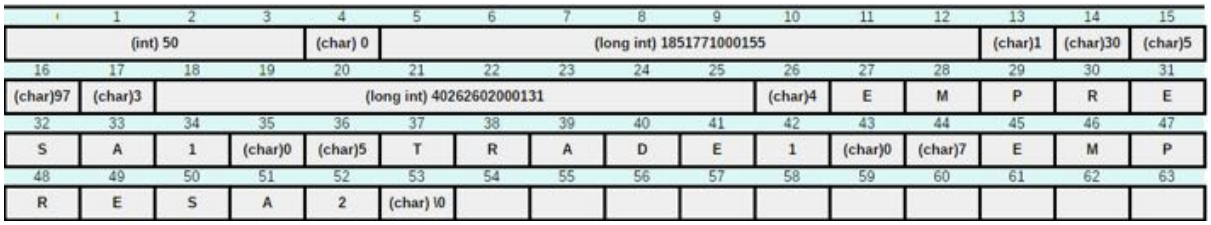
2.3 Registros com Indicador de Tamanho

Assim como nos registros separados por delimitadores, considerou-se imperativo o uso de número variável de campos, e utilizou-se um *byte* contendo números de 0 a 7, semelhantes a *tags*, para identificar qual campo que está naquela posição do registro.

O percurso do registro ocorre da mesma forma que nos registros separados por delimitadores, seguindo os passos: ler *tag* e depois ler campo. Por isso, não há muita diferença entre essas duas organizações de registro. A principal diferença é que, nesta

organização, os registros não são terminados por um *byte* “(char) 8”, mas são iniciados por um número inteiro de 4 *bytes* que contém o tamanho do registro.

Figura 3: Armazenamento seguindo a organização “indicador de tamanho”.



A Figura 3 mostra a maneira pela qual o registro exibido na Tabela 1 seria armazenado em disco seguindo a organização de registros em questão.

Nos *bytes* de 0 a 3 encontra-se o tamanho do registro, que é de 50 *bytes*. Não há delimitadores de registros. O restante é igual ao que foi descrito na Seção 2.2.

3 Otimizações Voltadas ao Usuário

Existem alguns aspectos do projeto que foram feitas com intuito de melhorar a interação do usuário com o programa, e que merecem ser destacadas.

3.1 Busca de CNPJ e Data por Meio de Expressões Regulares

Inicialmente, para buscar quaisquer CNPJ ou datas, o usuário era forçado a digitar os valores buscados de forma exata, seguindo o formato das representações textuais de CNPJ e data. Ou seja, para buscar o CNPJ 01.851.771/0001-55, o usuário não poderia omitir nenhum dos símbolos. No caso das datas, números de um dígito teriam que ser digitados com um 0 à esquerda, para formar 2 dígitos como estão nos arquivos CSV.

Esse comportamento do programa provavelmente é aceito em várias situações reais; e talvez seja até desejável, se for levado em consideração que tornar esse comportamento mais amigável ao usuário impactaria em algum nível o desempenho das buscas realizadas no programa. No entanto, um programa que aceite chaves de busca de forma mais leniente também parece ter o seu mérito, por tornar mais intuitivo o seu uso.

Tendo em vista essas ponderações, e também considerando que não foram explicitadas nas especificações o comportamento desejado, foi decidido que seriam aceitos vários formatos de CNPJ e data, por meio do uso da biblioteca padrão *regex.h*.

Alguns formatos possíveis de CNPJ são: “01.851.771/0001-55”, “01851771000155” ou “01 851 771 / 0001 55”.

Para datas, alguns formatos possíveis são: “09/01/07”, “9/1/7”, “09 01 7”.

3.2 Busca de Texto *Case-Insensitive* e sem Acentos

Seguindo as mesmas ponderações feitas na Seção 3.1, foi decidido que a busca de texto seria realizada desconsiderando quaisquer acentos e diferença na caixa das letras (*case-insensitive)*.

Portanto, se o usuário desejar buscar o texto “Elisão por Incorporação”, pode digitar como chave da busca quaisquer um dos seguintes valores: “Elisão por Incorporação”, “Elisao por Incorporacao” ou “elisao por incorporacao”.

4 Otimizações de Desempenho

Durante todo o desenvolvimento do trabalho, teve-se como prioridades tanto a modularização do projeto como o desempenho do programa; que são duas prioridades conflitantes entre si.

Por exemplo, cogitou-se tirar as funções pequenas dos arquivos *.c* para definí-las no arquivo *.h* como *inline*, o que permite ao compilador realizar melhores otimizações. Porém, isso tornaria o código mais difícil de entender.

Como outro exemplo, considerou-se a possibilidade de criar uma biblioteca que manipulasse arquivos com auxílio de *buffers*, com uma região de memória que armazena previamente algumas páginas de disco do arquivo que se está manipulando. Porém, após estudar o assunto, descobriu-se que a biblioteca padrão C já faz isso, e permite a alteração do buffer por meio da função *setvbuf*. Fez-se alguns poucos *benchmarks* alterando o tamanho do buffer, mas o desempenho para manipulação de arquivos grandes nunca chegou a ser relevante.

Por fim, só houve uma otimização que merece estar nessa seção, que melhorou o desempenho do uso de expressões regulares.

4.1 Precompilação de Expressões Regulares

Após implementar a busca de registros com uso de expressões regulares, percebeu-se que o uso da função *regcomp* era muito custoso e estava sendo chamada a cada comparação. Porém, dado que o usuário escolheu fazer uma busca pelo critério, por exemplo, “CNPJ da Empresa”, é sabido que o único padrão de expressão regular que será utilizado para comparações com todos os registros será o padrão de detecção de CNPJ; este poderia, então, ser compilado somente uma vez.

Com isso, implementou-se novas rotinas de expressões regulares, que permitem a pré-compilação e liberação de memória alocada para esses padrões pré-compilados. O ganho de desempenho na comparação por critério foi significativo, compensando a perda de simplicidade no código.

5 Tratamento de Acentuação

Para desenvolver um projeto que possua funções de processamento de texto, é preciso ter conhecimento do domínio do dados textuais que são dados como entrada ao programa. Conhecendo o domínio, é possível dizer quais *encodings* de texto o programa deve suportar.

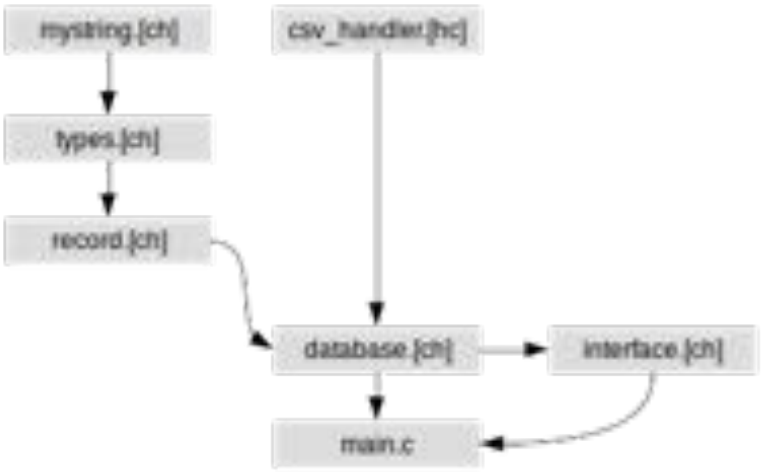
Nas especificações desse trabalho, foi dito o *website* específico de onde os dados seriam retirados, e os dados de lá vêm todos em *encoding latin1*. O arquivo CSV fornecido para testar o programa também veio nesse mesmo *encoding*. Portanto, considerou-se, baseado nesse domínio de dados textuais, que o *encoding* dos dados de entrada é somente o *latin1*.

Para trabalhar com esse *encoding*, apenas converteu-se todos os caracteres para o *encoding* UTF8, que é o padrão utilizado no Linux, sistema operacional no qual, de acordo com as especificações, os avaliadores executarão este projeto.

Como dito na Seção 3.2, a busca por registros ignora acentuação. Para realizar isso, tratou-se apenas os caracteres UTF8 que representam caracteres acentuados; estes são, por meio de uma tabela de conversão, convertidos para seus equivalentes ASCII antes de realizar qualquer comparação entre valores textuais.

6 Organização Interna do Projeto

Figura 4: Hierarquia dos arquivos de código-fonte.



A Figura 4 mostra as dependências entre os arquivos de código-fonte que constituem o projeto.

Tentou-se organizar o código-fonte de forma modularizada e de forma a reduzir o número de dependências de cada arquivo.

Um breve resumo da função de cada arquivo:

a) *mystring.c* contém rotinas de manipulação de "*strings*", incluindo a manipulação de caracteres em UTF-8;

b) *csv_handler.c* contém rotinas de leitura de valores em arquivos CSV;

c) *types.c* contém rotinas de manipulação dos tipos de dados que cada campo pode assumir: CNPJ, data e texto;

d) *record.c* contém rotinas de manipulação de registros, como leitura e escrita;

e) *database.c* contém rotinas de manipulação do banco de registros que fornecem as funcionalidades principais do programa;

f) *interface.c* contém rotinas de impressão da interface com o usuário;

g) *main.c* contém a função *main*, dando início ao programa;

h) *utils.h* contém funções que podem ser úteis em qualquer parte do projeto; todos os arquivos têm dependência do *utils.h*, motivo pelo qual foi omitido da Figura 4.

7 Detalhes Sobre as Funcionalidades

O projeto desenvolvido pelo grupo apresenta as seguintes funcionalidades:

a) leitura de um arquivo CSV;

b) impressão de todos os registros;

c) recuperação de registro por critério;

d) recuperação de registro por número;

e) recuperação de campo específico em registro recuperado por número.

7.1 Leitura de um Arquivo CSV

Essa funcionalidade permite adicionar ao arquivo principal todos os registros de um arquivo CSV especificado pelo usuário. Pelos motivos apresentados na Seção 5, é esperado que o arquivo CSV contenha texto em *encoding latin1*, que será convertido para UTF-8 antes de ser armazenado no arquivo.

7.2 Impressão de Todos os Registros

A funcionalidade de recuperação de todos os registros a partir do arquivo permite ao usuário realizar a impressão de todos os registros armazenados no arquivo, na ordem em que foram inseridos.

O processo de recuperação de registros é realizado por sucessivas leituras de registros, até que seja atingido o fim do arquivo. Após cada leitura, o registro é convertido para sua representação textual e então impresso.

No caso de não haver registros no arquivo, uma mensagem apropriada é impressa.

7.3 Recuperação de Registro por Critério

A funcionalidade de recuperação de registro por critério permite ao usuário buscar no arquivo um ou mais registros cujo campo selecionado seja equivalente à chave de busca fornecida.

O processo de busca é realizado pela leitura sucessiva de registros, comparando, após cada leitura, o valor contido no campo selecionado com o valor da chave de busca. Os critérios de comparação para cada tipo de dado são os descritos na Seção 3; e as comparações são realizadas nas representações textuais dos campos, e não nas representações binárias, o que possibilita o uso de expressões regulares.

A forma de percorrer o arquivo depende do tipo de organização de registros; e, além disso, os campos dos registros com valor nulo são exibidos como *null*.

No caso de não haver registros no arquivo, uma mensagem apropriada é impressa.

7.4 Recuperação de Registro por Número

A funcionalidade de recuperação de registro por número permite ao usuário buscar no arquivo um registro com um dado índice, que deve ser um número maior ou igual a 0.

A busca nesse caso tenta ser o mais eficiente possível, dentro das limitações de cada organização de registros. Por exemplo, na organização de registros “registros com indicador de tamanho”, o ponteiro de arquivo é avançado através de um registro inteiro com apenas 2 chamadas à função *fseek*, uma para ler o tamanho do registro, e outra para avançar sobre ele.

Caso um registro seja encontrado, as mesmas conversões textuais citadas no item anterior são realizadas, após isso a impressão é feita. Se não houver o registro pedido, uma mensagem apropriada é impressa.

7.5 Recuperação de Campo Específico em Registro Específico

A funcionalidade de recuperação de campo específico em determinado registro permite ao usuário imprimir na tela um campo específico do registro com o índice fornecido. Cada campo é identificado por um número entre 0 e 7, que o usuário deve fornecer ao

programa quando requisitado pela interface gráfica. A númeração de campos e registros é feita a partir do número 0.

O processo de busca é semelhante ao item anterior, incluindo o esforço de percorrer o arquivo de forma eficiente. A diferença aqui está na impressão, na qual apenas o campo especificado pelo usuário é impresso, em vez de o registro inteiro.

8 Ambientes de Execução Utilizados

Os ambientes utilizados para o desenvolvimento do projeto foram os expostos nas Tabelas 2, 3 e 4 abaixo.

Tabela 2: Ambiente de execução 1.

Sistema Operacional	Linux
Versão do Kernel	4.4.0-75-generic
Distribuição	Ubuntu GNOME 16.04.2 LTS (Xenial Xerus)
Compilador	gcc 5.4.0
Flags de Compilação	-Wall -O1 -Wno-unused-result -g

Tabela 3: Ambiente de execução 2.

Sistema Operacional	Mac
Versão do Kernel	MacOS Sierra (versão 10.12.4)
Compilador	gcc 4.2.1
Flags de Compilação	-Wall -O1 -Wno-unused-result -g

Tabela 4: Ambiente de execução 3.

Sistema Operacional	Linux
Versão do Kernel	4.4.0-38-generic
Distribuição	Linux Mint 18 Sarah
Compilador	gcc 5.4.0
Flags de Compilação	-Wall -O1 -Wno-unused-result -g

Em todos os 3 ambientes de execução, foi utilizado o mesmo arquivo *Makefile*. O comando de compilação foi:
gcc src/main.c src/csv_handler.c src/record.c src/types.c src/database.c src/interface.c src/mystring.c -Wall -O1 -Wno-unused-result -I include/ -g -o database

9 Capturas de Tela da Interface com o Usuário

A interface foi elaborada para ser a mais autoexplicativa possível.

Como mostrado na Figura 5, inicialmente a interface solicita ao usuário que informe 2 tipos de dados, os quais deverão estar dispostos um em cada linha. Primeiro deve-se informar o nome para o arquivo onde serão armazenados os registros. Após isso, deve-se escolher entre uma das opções oferecidas pelo menu como forma de organização dos dados para este arquivo, as quais se limitam às opções: número fixo de campos, delimitadores entre registros ou indicadores de tamanho de registro.

Figura 5: Tela inicial do programa.

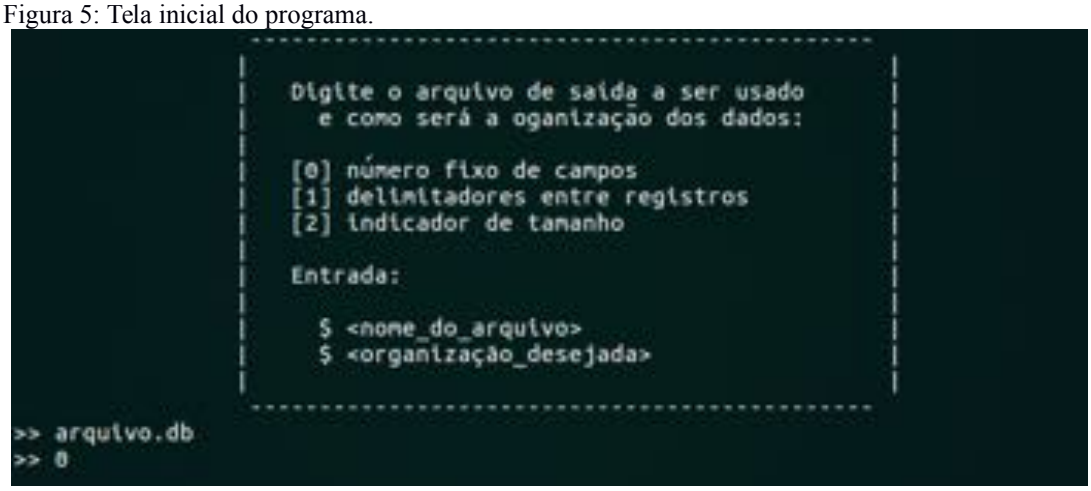
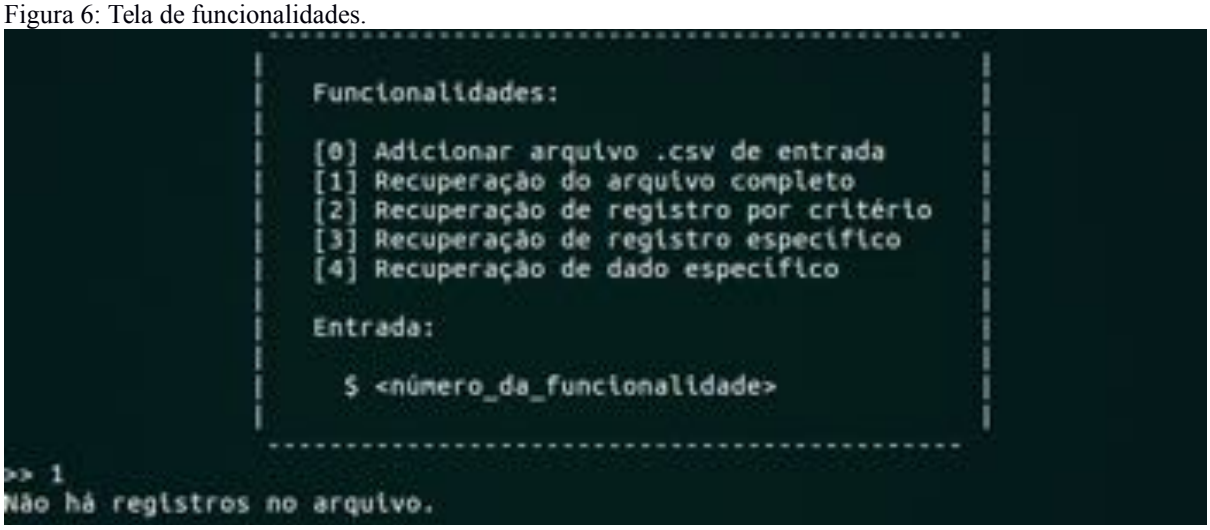


Figura 6: Tela de funcionalidades.



Após as informações iniciais serem obtidas, o menu com o restante das funcionalidades é apresentado e nele o usuário pode escolher uma funcionalidade do programa. Esse comportamento é mostrado na Figura 6.

Ao selecionar a opção 0, relativa à adição de registros a partir de um arquivo CSV, o usuário deverá informar o nome do arquivo CSV para leitura, como exibido na Figura 7. Após isso, os dados serão gravados no arquivo, seguindo a organização de registros selecionada inicialmente.

Após a execução dessa funcionalidade, assim como a de todas as outras, uma interface de saída é exibida perguntando se o usuário deseja continuar ou sair da aplicação, para previnir que a interface de seleção de funcionalidade impeça a visualização dos resultados gerados pela última funcionalidade executada. Dessa forma, o espaço ocupado após a execução de uma funcionalidade é mínimo, permitindo melhor visualização dos resultados.

Figura 7: Tela de inserção de arquivo CSV.

Figura 8: Tela da impressão de registros.

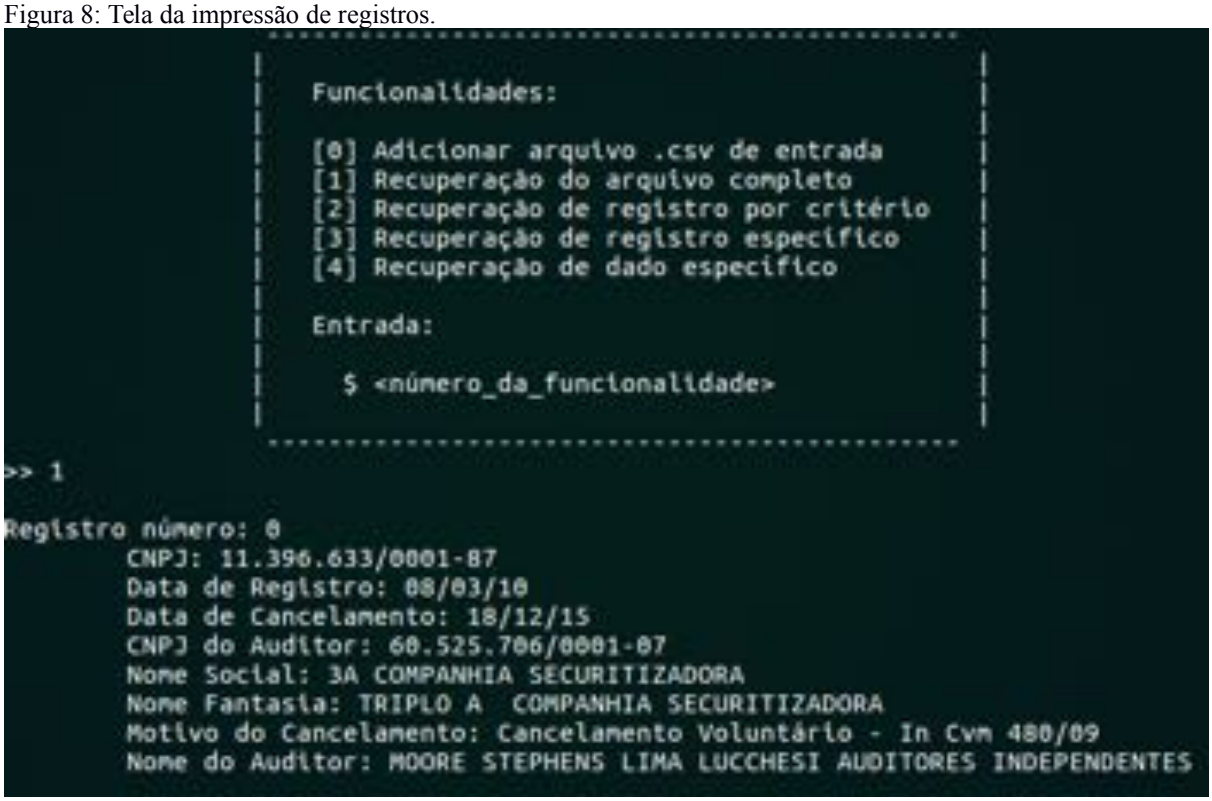


Figura 9: Final da impressão de registros.

A opção de recuperação de todos os registros simplesmente imprime na tela todos os registros armazenados no arquivo previamente gerado, como mostrado nas Figuras 8 e 9.

A Figura 10 exibe o modo de uso da funcionalidade de recuperação por critério. Essa funcionalidade solicita ao usuário por qual critério ele deseja realizar a busca, em seguida deve-se informar o dado a ser buscado. Por fim, todos os registros encontrados são exibidos na tela do programa.

Figura 10: Imagem da busca por critério.

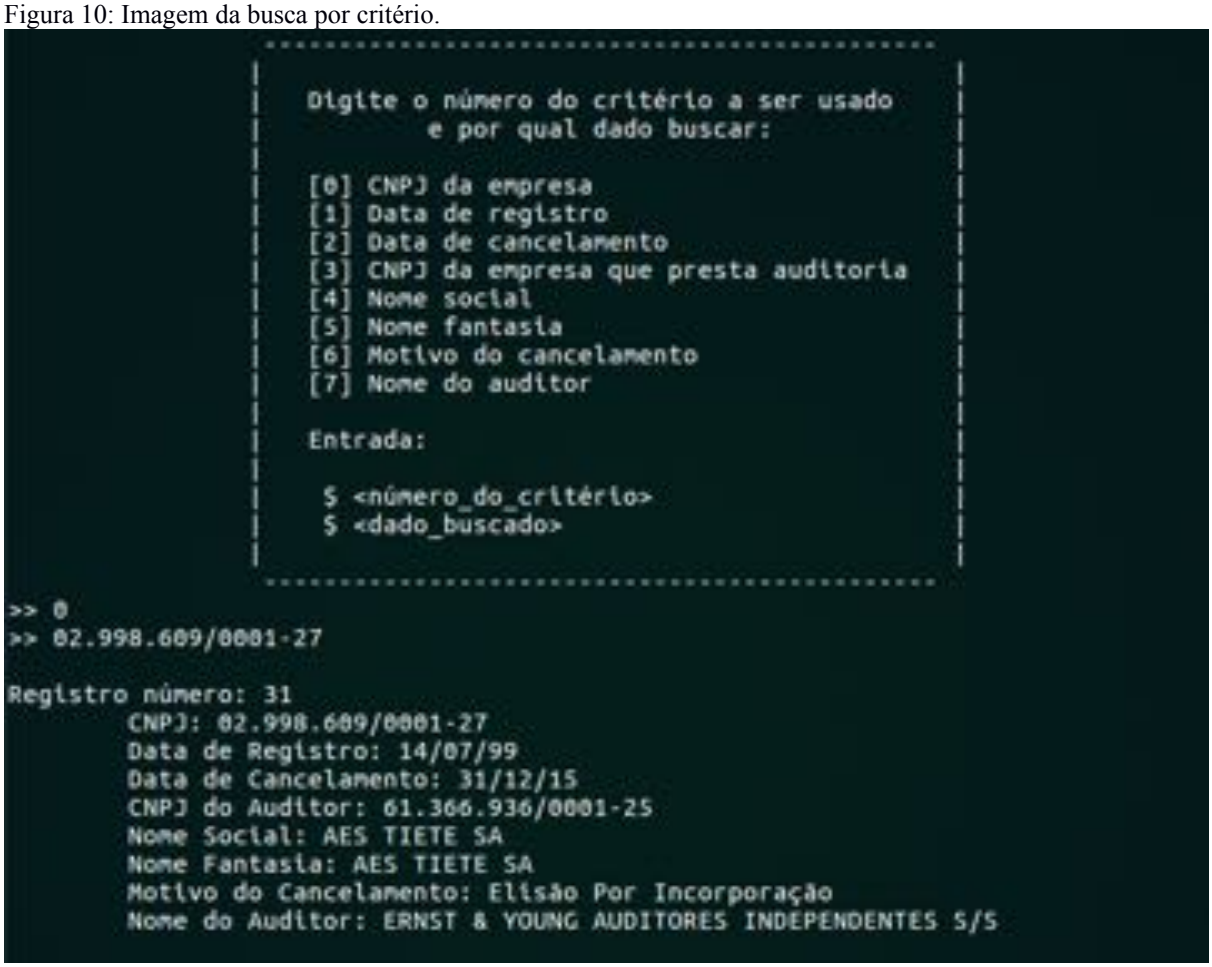


Figura 11: Tela da busca por índice de registro.

A opção de recuperação de registro por número solicita ao usuário o número do registro a ser buscado e o imprime na tela caso encontrado, como mostrado na Figura 11.

Figura 12: Tela da recuperação de campo em registro.

A Figura 12 mostra a recuperação de um campo de um registro. Essa funcionalidade permite que o usuário informe o número de um registro e o índice de um campo que deseja imprimir, caso a combinação desses dois critérios seja válida, o campo será impresso na saída do programa.

10 Bateria de Testes

Figura 13: Escolha do nome do arquivo e organização.

As Figuras de números 14 em diante são feitas com a organização de registros

“registros com indicador de tamanho”.

Figura 14: Teste da funcionalidade 1 em arquivo sem registros.

Figura 15: Teste da funcionalidade 0, adição de CSV.

Figura 16: Teste da funcionalidade 2, recuperação por critério. Data de registro como critério.

Figura 17: Continuação do teste da funcionalidade 2, recuperação por critério. Data de registro como critério.

Figura 18: Teste da funcionalidade 2, recuperação por critério. Nome fantasia como critério. Chave da busca em letras minúsculas.

Figura 19: Abertura de arquivo em outra organização de registros, e leitura do arquivo CSV.

As Figuras de número 20 em diante mostram testes realizados em um arquivo com organização de registros “número fixo de campos”.

Figura 20: Teste da funcionalidade 1.

```
 ---------------------------------------------
|                                             |
|   Funcionalidades:                          |
|                                             |
|   [0] Adicionar arquivo .csv de entrada     |
|   [1] Recuperação do arquivo completo       |
|   [2] Recuperação de registro por critério  |
|   [3] Recuperação de registro especifico    |
|   [4] Recuperação de dado especifico        |
|                                             |
|   Entrada:                                  |
|                                             |
|     $ <número_da_funcionalidade>            |
|                                             |
 ---------------------------------------------
>> 1

Registro número: 0
        CNPJ: 11.396.633/0001-87
        Data de Registro: 08/03/10
        Data de Cancelamento: 18/12/15
        CNPJ do Auditor: 60.525.706/0001-07
        Nome Social: 3A COMPANHIA SECURITIZADORA
        Nome Fantasia: TRIPLO A  COMPANHIA SECURITIZADORA
        Motivo do Cancelamento: Cancelamento Voluntário - In Cvm 480/09
        Nome do Auditor: MOORE STEPHENS LIMA LUCCHESI AUDITORES INDEPENDENTES


Registro número: 1
        CNPJ: 01.547.749/0001-16
        Data de Registro: 11/07/97
        Data de Cancelamento: null
        CNPJ do Auditor: 10.830.108/0001-65
        Nome Social: 521 PARTICIPAÇOES S.A. - EM LIQUIDAÇÃO EXTRAJUDICIAL
        Nome Fantasia: 521 PARTICIPAÇÕES S/A
        Motivo do Cancelamento: null
        Nome do Auditor: GRANT THORNTON AUDITORES INDEPENDENTES


Registro número: 2
        CNPJ: 01.851.771/0001-55
        Data de Registro: 30/05/97
        Data de Cancelamento: null
        CNPJ do Auditor: 40.262.602/0001-31
        Nome Social: 524 PARTICIPAÇOES SA
        Nome Fantasia: 524 PARTICIPACOES SA
```

Figura 21: Continuação do teste da funcionalidade 1.

Figura 22: Teste da funcionalidade 4, busca de campo específico.

Figura 23: Teste da funcionalidade 2. Critério “CNPJ da empresa”. Chave da busca como um CNPJ escrito sem símbolos.

```
>> 2

 ------------------------------------------------
|                                                |
|   Digite o número do critério a ser usado      |
|           e por qual dado buscar:              |
|                                                |
|   [0] CNPJ da empresa                          |
|   [1] Data de registro                         |
|   [2] Data de cancelamento                     |
|   [3] CNPJ da empresa que presta auditoria     |
|   [4] Nome social                              |
|   [5] Nome fantasia                            |
|   [6] Motivo do cancelamento                   |
|   [7] Nome do auditor                          |
|                                                |
|   Entrada:                                     |
|                                                |
|    $ <número_do_critério>                      |
|    $ <dado_buscado>                            |
|                                                |
 ------------------------------------------------
>> 0
>> 04 913 711 0001 08

Registro número: 197
        CNPJ: 04.913.711/0001-08
        Data de Registro: 20/07/77
        Data de Cancelamento: null
        CNPJ do Auditor: 52.803.244/0001-06
        Nome Social: BANCO DO ESTADO DO PARÁ S/A.
        Nome Fantasia: BANPARÁ S.A.
        Motivo do Cancelamento: null
        Nome do Auditor: KPMG AUDITORES ASSOCIADOS


 ------------------------------------------------
|                                                |
|   [0] Executar outra funcionalidade            |
|   [1] Sair                                     |
|                                                |
 ------------------------------------------------
```

11 Referências Bibliográficas

FOLK, M.; ZOELLICK, B., File Structures, Second Edition. Addison-Wesley, 1992.