

# **Relatório Final**

## **MI - Algoritmo I**

**Gustavo Henrique Bastos de Oliveira<sup>1</sup>**  
**ghboliveira@hotmail.com**

<sup>1</sup>Engenharia da Computação – Universidade Estadual de Feira de Santana (UEFS)  
Caixa Postal: 252-294 – CEP – 44.036.900 – Feira de Santana – BA – Brasil

### **1. Introdução**

A XIX Semana de Iniciação Científica da UEFS (SEMIC) faz parte das atividades da Semana Nacional de Ciência e Tecnologia. O tema do evento nesta edição de 2015 é "Luz, Ciência e Vida" e acontecerá no período de 19 à 22 de outubro de 2015. Podem participar do evento estudantes e professores e não há limite de vagas. No entanto é necessário que os interessados façam sua inscrição fornecendo alguns dados: Nome, CPF, E-mail, Matrícula, Data de Nascimento e Ocupação(professor ou estudante). Sabendo da competência dos estudantes do 1º semestre de computação, a Universidade Estadual de Feira de Santana(UEFS) resolveu conceder à esses alunos a oportunidade de desenvolver o sistema de inscrições para a SEMIC.

### **2. Desenvolvimento**

#### **2.1. Materiais e Métodos**

Como o problema exige que não deve haver limites de vagas, tornou-se necessário algum tipo de estrutura que não fosse alocadas estaticamente, que por sinal tornaria o projeto mais portátil e que suportasse uma grande e imprevisível quantidade de inscrições. Inicialmente os vetores iriam ser o tipo de estrutura que suportaria todos os dados exigidos e de fácil manipulação, mas ao decorrer do problema percebemos que com a utilização dessa estrutura estática não poderíamos ter uma quantidade ilimitada de inscrições.

Para desenvolvimento do programa tornou-se necessário um tipo de estrutura que fossem alocada dinamicamente, por não ter conhecimento da quantidade de cadastros. A alocação dinâmica foi escolhida para oferecer um meio pelo o qual o programa possa obter espaço para armazenamento em tempo real, como C inclui um subsistema de alocação dinâmica o armazenamento da informação é alocado na *heap*(memoria livre)[Schildt, 1996, p. 420], também foi aplicada por gerenciar o espaço adequado sem desperdiçar memoria alocada sem uso. Para o controle dos cadastros foram utilizados ponteiros, para a escrita, leitura e edição do arquivo de texto, que possibilita ao usuário o armazenamento de todo o conteúdo após a finalização do programa. O programa foi dividido em funções para não ter a que repetir varias vezes o mesmo código, por exemplo na escrita ou leitura dos arquivos gerados. Mas como alocar na *heap* todo aquele tipo de informações do inscrito?, seria necessário alocar vários tipos de dados(*int*, *char*), e depois como liga-los?.

#### **2.2. Struct**

Como a linguagem de programação C permite criar tipos de dados definíveis pelo usuário de diversas formas. Um deles é a *struct*, que é um agrupamento de variáveis sob um

nome e também é chamado de tipo de dado agregado, ou às vezes, conglomerado. Esse tipo de estrutura disponibiliza o conjunto de informações que conseqüentemente irá armazenar os dados dos usuários. Basicamente cria-se uma estrutura que pode armazenar ao mesmo tempo diversas informações(*int*, *int \**, *char*, *float* entre outros). Junto com a alocação dinâmica a *struct* traria uma infinidade de cadastros e uma maior portabilidade do projeto. Mas como iríamos acessar nos diversos espaços criados dinamicamente?, seria necessário ter ponteiros que indicassem onde cada cadastro estava guardado na *heap*. Mas como a *struct* suporta todos os tipos de dados definidos na linguagem C, além de armazenar os dados de cada usuário ela conseqüentemente armazenaria o local onde estaria o próximo cadastro. Mas como o ponteiro estaria apontando para uma estrutura até então desconhecida pelo projeto, para inicializa-lo devemos informar o tipo de estrutura que apontaremos, vamos tomar de exemplo que exista uma *struct cadastros*, o ponteiro deve ser iniciado por *struct cadastros \*ponteiro*. Para uma melhor manipulação da *struct*, invés de declararmos variáveis deste tipo o tempo de *struct cadastros valor*, definiríamos um novo nome para essa estrutura utilizando a palavra-chave *typedef*.

### 2.3. Typedef

C permite que possamos definir novos nomes aos tipos de dados, utilizando a palavra-chave *typedef*. Não estaríamos criando uma nova classe de dados, mas ao contrário, definindo um novo nome para um tipo já existente. Isso ajuda a tornar o programa um pouco mais portáteis. Mas deve-se ter noção que ao compilar o projeto em uma máquina com sistema diferente, apenas o comando *typedef* deve ser alterado para quando tornar-se necessário a compilação em um novo ambiente.

### 2.4. Lista Encadeada

Uma lista singularmente encadeada requer que cada item de informação contenha um elo com o próximo elemento da lista(que no nosso caso seria o ponteiro previamente definido na *struct*). Onde cada célula ou nó(cada bloco de informação alocada) iria ter a informação do próximo nó. Uma lista encadeada é basicamente o conjunto de nós, onde cada nó possui referência para o próximo.

### 2.5. Arquivos de texto

Como exigido no problema todos os dados cadastrados devem ser mantidos após o encerramento do programa e ao reexecutá-lo os dados devem ser resgatados. Para tal armazenamento, foram escolhidos os arquivos de texto. Apesar de ter conhecimento de arquivos binários, os arquivos de texto foram escolhidos por ser de mais fácil avaliação, facilitando a detecção de erros, que no binário não seriam perceptíveis. Para a manipulação do arquivo de texto foi utilizado ponteiros do tipo *FILE \**, que é um ponteiro para informações que definem várias coisas sobre obter arquivo, incluindo nome, status e a posição atual do arquivo. Basicamente identifica um arquivo específico em disco e é usado pela *stream*(sistema de arquivos de C que trabalha com uma ampla variedade de dispositivos, terminais, acionadores de disco entre outros[**Schildt, 1996, p. 220**]) associada para direcionar as operações das funções de E/S(funções de entrada/saída pré-estabelecidas na biblioteca *stdio.h*[**Schildt, 1996, p. 198**]). Nestes arquivos de textos serão guardadas as informações dos usuários cadastrados e data limite para inscrição, *Cadastros.txt* e *data.txt*, respectivamente.

### 3. Descrição da Solução

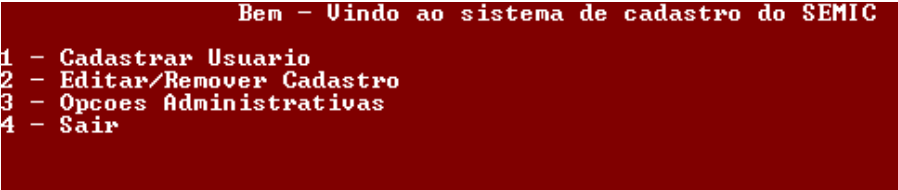
O projeto seguiu as orientações impostas, como boa modularização, fraco acoplamento entre funções e sem variáveis globais. Como imposto pelo problema o usuário além de poder se inscrever, ele deve também poder alterar ou remover o seu cadastro caso desista de participar do evento.

#### 3.1. Avaliação Inicial

Ao iniciar o programa o administrador deve informar a data limite para inscrições(função: `inserir_data_limite()`), que verifica a existência do arquivo `data.txt`, se existir a função não faz nada, se não irá criar um novo arquivo com as informações introduzidas pelo administrador, onde o programa vai ter como base o horário do sistema para avaliar o prazo. Para a implementação, foram usadas funções de horário do sistema como as estruturas de tempo: `struct tm`, `time`; e as funções: `localtime()`, `time()`; (funções pré-estabelecidas na biblioteca `time.h`). Com os dados obtidos a função `verificar_data()`, verifica se o prazo foi atingido, possibilitando ou não a inserção de novos cadastros.

#### 3.2. Menu Inicial

Após a avaliação inicial será exibido o menu inicial[Figura 1], que contém as seguintes opções: **1- Cadastrar Usuario, 2- Editar/Remover Cadastro, 3- Opcoes Administrativas, 4- Sair.**



```
Bem - Vindo ao sistema de cadastro do SEMIC
1 - Cadastrar Usuario
2 - Editar/Remover Cadastro
3 - Opcoes Administrativas
4 - Sair
```

Figure 1. Quadro de medalhas exibido durante a adição de medalhas.

##### 3.2.1. Cadastrar Usuario

Ao escolher essa opção o usuário deve informar todos os dados exigidos. Se em algum momento o usuário tentar introduzir um participante que já esteja cadastrado uma mensagem de erro é exibida. Para determinar se existe um cadastro a função verifica o CPF. Ao inserir todos os dados pela primeira vez, a função cria a lista encadeada. Com a inserção de novos participantes a lista vai inserindo ordenando em ordem alfabética. Foi necessário inserir ordenando pela necessidade de depois listar todos os participantes sem a necessidade de uma nova função[Figura 2].

```

!--> Cadastro do Usuario <--!
Digite o seu CPF <XXX.XXX.XXX-XX>
000.000.000-00
Digite seu nome:
Ml - Algoritmos I
Digite a sua matricula:
00000000
Digite a sua data de nascimento: DD/MM/AAAA
00/00/0000
Digite seu E-mail:
mialgoritmosi@uefs.com
Qual a sua ocupacao?
P - Professor  A - Aluno
P

```

Figure 2. Informações exigidas ao cadastrar o usuário.

### 3.2.2. Editar/Remover Cadastro

Nessa função o usuário deve inserir o CPF que ele deseja editar ou remover. Se não existir uma mensagem de erro é exibida. Se existir serão mostrados as informações armazenadas e a opção para editar, remover ou não fazer nada. No caso de editar, o usuário deve inserir novamente todas as informações corretamente, para seguir a ordenação da lista o cadastro é removido e inserido novamente.

### 3.2.3. Opções Administrativas

Nesta seção apenas o administrador irá ter acesso, pois exige senha, que é "admin". Após inserir a senha, o administrador será levado para um segundo menu, onde irá ter as seguintes opções[Figura 3]: **1- Listar Cadastros, 2- Remover/Editar dados cadastrados, 3- Alterar data do evento, 4- Sair**. Se o administrador escolher a opção **1- Listar Cadastros**, ele poderá escolher entre mostrar todos, somente alunos ou somente professores. Para mostrar todas essas informações foi necessário apenas uma função(*void p\_lista(char)*) que recebe como parâmetro um caractere, se escolher mostrar todos os participantes irá listar todos os cadastros onde a ocupação seja diferente do caractere 'T', se escolher listar todos os professores irá listar todos os cadastros com ocupação diferente de 'A' e se escolher listar os alunos irá listar todos os cadastros com ocupação diferente de 'P'.

Na opção **2- Remover/Editar dados cadastrados**. É a mesma função disponibilizada para o usuário.

Na opção **3- Alterar data do evento** o administrador pode manipular a data limite. Onde deve informar dia, mês e ano do seguinte formato **DD/MM/AAAA**(Dia/Mês/Ano).

Na opção **4- Sair**, o administrador volta para o menu principal.

```

Bem - Vindo!
Escolha uma das opcoes :
1- Listar cadastros
2- Editar/Remover dados cadastrados
3- Alterar data do evento
4- Sair

```

Figure 3. Menu do administrador.

### 3.2.4. Sair

Ao selecionar a opção que encerra o programa, o usuário permite que todos os dados sejam armazenados[Figura 4] e dá free(função responsável por desalocar) na lista.

```
000.000.000-00
MI - Algoritmos I
00000000
00/00/0000
mialgoritmosi@uefs.com
P
```

Figure 4. Dados gravados na saída.

## 4. Conclusão

Para a solução do problema foi necessário busca de conhecimento externo, por meios de livros e apostilas. Durante a execução do programa todos os testes denotaram funcionamento de acordo com o exigido. Algumas funções podem ser aprimoradas para um melhor funcionamento. Todas as palavras do programa estão sem a acentuação correta, por haver pouco tempo para a entrega. Para que o relatório não excedesse o tamanho máximo permitido as imagens foram cortadas, abaixado consequentemente sua qualidade, além que alguns aspectos não mencionados, mas que não seriam necessários para manipular o programa. Alguns bugs foram detectados mas não interferiram na execução do programas, relacionados a IDE utilizada. Em alguns momentos a lista perdia uma parte das informações, depois retornavam. O *if(s)* da quarta opção do menu principal não funciona da forma pretendida, mas não interferiu nos testes. O relatório segue o exemplo disponível no site da disciplina e seguindo as orientações das aulas de Produção de Textos Teóricos e Acadêmicos(PTTA). Para obtenção dos mesmos dados destacados no relatório deve-se considerar usuário perfeito. Para a construção do problema foi utilizado a IDE *Code-Blocks 13.12* e compilador *GCC*, baseado no sistema Windows 10.

## 5. Referências

Introdução ao  $\text{\LaTeX}$ , <<http://latexbr.blogspot.com.br/2010/04/introducao-ao-latex.html>>. Acesso em: 05/10/2015 as 20:10.  
Tec503-1. <<http://sites.ecomp.uefs.br/tec503-1/>>. Acesso em: 05/10/2015 as 19:14.  
Schiltdt, H. C COMPLETO E TOTAL. 3º edição. São Paulo. 1996.