LUCAS FONSECA DOS SANTOS, MATHEUS HENRIQUE R. DE SOUZA, MAYRA CRISTIANE S. ASSIS

MARVEL DATA SYSTEM:

DATA STRUCT PRATICAL PROJECT

1^o Stage - Access to secondary memory

LAVRAS – MG

2017

LUCAS FONSECA DOS SANTOS, MATHEUS HENRIQUE R. DE SOUZA, MAYRA CRISTIANE S. ASSIS

MARVEL DATA SYSTEM:

DATA STRUCT PRATICAL PROJECT

1^{o} Stage - Access to secondary memory

Pratical project to data structure discipline. This project is an information recording system, based on a subject chosen by the teacher

Prof. Juliana Galvani

Professora

Prof. Joaquim Uchoa

Professor

LAVRAS – MG

2017

RESUMO

O presente documento descreve de modo detalhado o desenvolvimento de um sistema de registros, de tema apresentado pela professora. O projeto prático está divido em 4 etapas de desdenvolvimento, onde a primeira etapa consiste no desenvolvimento de um mecanismo de acesso à memória secundária, para o presente caso, acesso e armazenamento em um arquivo binário, seguido de métodos de leitura, escrita, procura e listagem dos registros armazenados nos arquivos de dados.

Palavras-chave: Data struct, Memory Access, Binary File, List, Marvel, Registers

ABSTRACT

This document describes the development of a system of registers from a topic presented by the teacher. The pratical project is divited into 4 stages. The first stage is to develop a memory access mechanism, for this case, using binary files for register the data and methods for load, write and search register in data files. The theme worked by team, is Marvel Hero records.

Keywords: Data struct, Memory Access, Binary File, List, Marvel, Registers

FIGURES LIST

TABLES LIST

SUMÁRIO

1 INTRODUÇÃO

O presente documento tem como objetivo descrever de forma técnica e teórica o desenvolvimento do projeto prático da disciplina de Estrutura de Dados (GCC126) da Universidade Federal de Lavras. O projeto é dividido em 4 etapas de desenvolvimento, onde faz-se presente a primeira etapa. Tal projeto, quando finalizado em 4 etapas, terá como objetivo a aplicação direta dos conhecimento angariados ao decorrer letivo da discplina. Todo o código é disponibilizado pelo equipe de trabalho sobre licença `GNU/GPL` para a quaisquer possiveis reusos para aprendizagem e outros fins.

Na presente etapa (primeira etapa), o objetivo da equipe tange-se no desenvolvimento de um mecanismo de acesso à memória secundária, por via da leitura/escrita de um arquivo. Para este caso, gozou-se de um arquivo binário de dados, onde através de métodos de leitura, escrita, procura, listagem e ordenação, os registros foram organizados e manipulados para fins do usuário final.

Segundo conscenso da equipe, não possui interface gráfica, a interação humano-computador concede-se por meio de opções selecionadas pelo usuário através de linhas de comando em um terminal linux. Desenvolvido em linguagem de programação C++, sobre paradigma de programação orientada a objeto, apresenta outros recursos em sua composição, tais como Listas e mecanismos de manipulação de dados. A presente equipe objetiva futuramente ao decorrer do desenvolvimento das outras etapas do projeto, a implementação de funcionalidades além do requisitado pelo escopo do projeto, além de outras estruturas de dados, trabalhadas ao decorrer letivo da disciplina afim de facilitar-se a manipulação dos registros armazenados.

2 ESTRUTURA DO SISTEMA

À presente etapa de desenvolvimento do sistema (primeira etapa), o sistema conta apenas com um mecanismo de acesso à memória secundária por via de um arquivo binário de dados (`.dat`), localizado no diretório de dados `/data/data.dat`. Os dados armazenados em tal arquivo, são codificados e seu conteudo é composto das posições de memória dosm objetos ou estruturas armazenadas em seu conteúdo por via dos registros. Sua interpretação e manipulação faz-se necessário por intermédio do software aqui explanado.

Tabela 2.1 – Diretórios do Sistema

Diretório	Função no Sistema
`data`	*Armazena arquivos de registro de dados e futuramente configurações do sistema.*
`include`	Responsável por agrupar os arquivos header, cabeçalhos que descrevem classes e tipos de dados.
`src`	Agrupa as implementações das classes descritas e armazenadas no diretório `include` além da implementação de classes test.

Fonte: Diretórios do sistema

Trabalhando-se sobre o paradigma da programação orientada à objetos, projetou-se o sistema de tal forma a aumentar a coesão das classes, facilitando a manutenciabilidade que será um aspecto altamente requisitado à tal projeto, haja visto do mesmo ser dividido em 4 etapas de desenvolvimento. Dessa forma, como supracitado, aumenta-se a facilidade da manutenção e as possibilidades de evolução do software.

A divisão por diretórios sobre cada responsabilidade dentro da funcionalidade do sistema, aumenta-se a flexibilidade da modularização, outro aspecto fundamental no paradigma da orientação à objeto. Dessa forma, a legibilidade mais superficial do sistema, a nível de arquivos e código em alguns aspectos, torna-se muito mais facíl ao usuário não familiarizado com o sistema, como o professor que está a corrigir o presente trabalho e não obteve um acesso antecipado ao código. Dessa forma, a evolução do software torna-se muito mais facilitada também,

Tabela 2.2 – Classes do Sistema

Classe	Função no Sistema
`Teste`	*Classe para teste das funcionalidades do sistema.*
`System`	Classe que descreve o objeto sistema e suas funcionalidades, toda execução.
`Object`	Classe para definição das estruturas de dados dos registros trabalhados e manipulados no sistema.
`Screen`	Classe de comunicação com usuário final e GUI do sistema.
`List`	Classe que implementa a estrutura de dados Lista para manipulação dos registros.

Fonte: Classes do sistema

podendo-se inclusive expandir-se à implementação mais facil de uma interface gráfica, devido à uma separação (neste caso superficial) do meu `controller`.

O projeto até o presente momento contém uma estrutura de dados Lista, de especificação lista duplamente encadeada, escrita estritamente para o projeto em questão. Contém apenas os métodos necessários às necessidades em questão, de forma NÃO ADEQUANDO-SE A REUTILIZAÇÃO EM OUTROS PROJETOS, haja visto à estrutura especifica para a solução de problemas do projeto em questão.

```
class Node {
   friend class List;
   private:
      Node* next;
      Node* previus;
      object hero;
   public:
      Node(object hero);
      ~Node();
};

class List {
   private:
      Node* firstElement;
```

```
        Node* lastElement;
        int size;
    public:
        List();
        ~List();
        void insertNewElement(object hero);
        void removeElement(int id);
        inline bool isEmpty();
        string printList();
        object returnData();
};
```

Fonte: Estrutura do objeto nó e lista.

Como não denota-se objetivo do presente trabalho o estudo primário das estruturas de dados anteriormente estudadas na disciplina, o presente documento apresentará abaixo como exemplo somente os métodos especificos para o projeto em questão, conforme discorrido no paragráfo anterior.

```
object List::returnData() {
    if(isEmpty()) return object{-99,0,0,0};
    Node* tmpNode = this->firstElement;
    object tmpObj = tmpNode->hero;
    if(tmpNode->next == NULL) {
        delete tmpNode;
        this->firstElement = NULL;
        this->size--;
        return tmpObj;
    }else {
        tmpNode = tmpNode->next;
        delete tmpNode->previus;
        tmpNode->previus = NULL;
        this->firstElement = tmpNode;
```

```
        this->size--;
        return tmpObj;
    }
}
```

Fonte: Métodos especificos da estrutura de dados Lista.

O método supracitado, retorna um elemento da lista por vez, a cada interação na unidade chamadora do método, onde a cada retorno, efetua uma remoção do elemento em questão, de tal forma a ir esvaziando a lista.

O método de remoção da lista também forá modificado para tal necessidade, efetuando a remoção de um elemento com base no ID informado via parametro.

```
Node* tmp = this->firstElement;
    while(tmp->hero.id != idHero) {
        tmp = tmp->next;
    }
    if(tmp->hero.id == idHero) {
            Implementacao do metodo...
    }
```

Fonte: Métodos especificos da estrutura de dados Lista.

Como supracitado no presente documento, o mesmo apresenta a descrição a respeito do desenvolvimento da primeira de quatro etapas subdivididas no escopo do trabalho. Implementado nessa fase, encontra-se o mecanismo de comunicação com memória secundária por via de armazenamento de registros em arquivo binário. Para tal, gozou-se de bibliotecas próprias da linguagem de programação C++, em questão a biblioteca `fstream`.

Declarado um objeto stream de dados, para tipo binário, executou-se por via dos algoritimos no código implementados, a manipulação dos registros no software.

```
/*
```

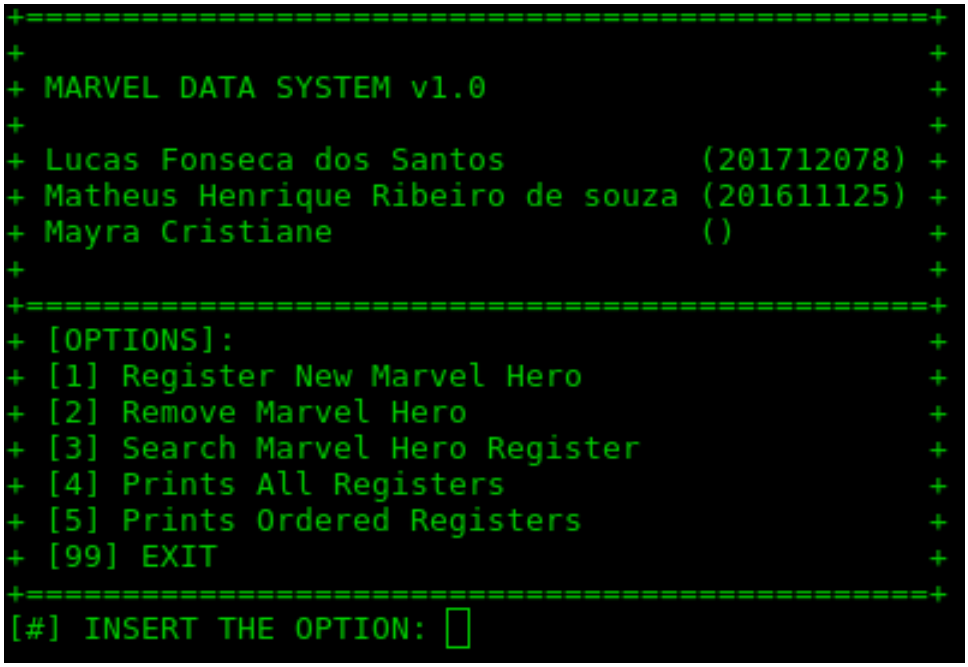
```
 * Stream de saida de dados;
 */
ofstream dataFile("../data/data.dat",
ios_base::in|ios_base::out|ios_base::binary|ios_base::app);

/*
 * Stream de entrada de dados;
 */
ifstream dataFileIn("../data/data.dat", ios_base::in|ios_base::out|ios_base::bina

/*
 * Metodo de leitura dos dados;
 */
dataFileIn.read((char*) &hero, sizeof(object));

/*
 * Metodo de escrita dos dados;
 */
dataFile.write((char*) &tmpHero, sizeof(object));
```

Fonte: Stream de dados do arquivo binário.

3 INTERFACE E FUNCIONALIDADE

Optou-se em consenso na equipe o não desenvolvimento de uma interface gráfica. Sendo assim, o projeto apresenta uma interface em linhas de comando, mas com usuabilidade amigável.

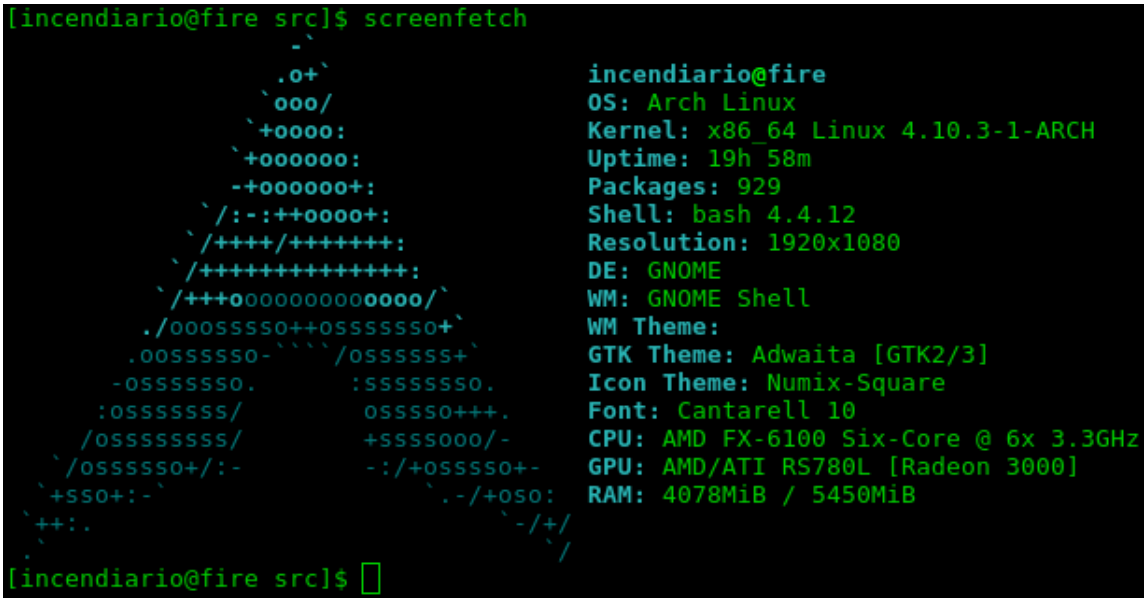
Figure 3.1 – Uma Figura de Exemplo



Fonte: Interface do Sistema

O sistema foi desenvolvido em uma máquina munida da distribuição GNU-LINUX - Arch Linux, com kernel x86 64 Linux 4.10.3-1-ARCH, com processador AMD FX-6100 Six-Core @ 6x 3.3GHz.

Figure 3.2 – Uma Figura de Exemplo



Fonte: S.O.