Universidade Federal de Minas Gerais

Exercício I – Programação e Desenvolvimento de Software II

Daniel Matoso de Oliveira Cruz | 2017118014
Lucas Thiago Dias | 2018072158
Robson Matheus Fernandes de Oliveira | 2017074297

28 de Abril de 2019

1 Introdução

Neste trabalho desenvolvemos uma aplicação para a simulação de uma agenda virtual. Essa agenda é capaz de mapear compromissos por mês, dia, hora e minuto. Para o desenvolvimento, utilizamos duas estruturas de dados: vetores de tamanho fixo (*array*) e listas encadeadas. Os detalhes de implementação serão abordados nas próximas seções.

2 Estruturas de Dados Utilizadas

A agenda possui uma lista encadeada de meses, como requisitado na especificação. Cada mês possui um vetor de tamanho fixo compondo os compromissos para cada dia, onde seu tamanho é o número de dias naquele mês, o que pode variar de um mês para outro. Além disso, as listas de compromissos em cada mês são listas encadeadas, tendo em vista que podem assumir tamanhos variáveis.

3 Modularização e Organização

O trabalho foi desenvolvido de forma modular, possibilitando uma melhor manutenabilidade e legibilidade do código. A organização adotada é apresentada abaixo.

- **Agenda**: estrutura principal da agenda. Trata-se de uma lista encadeada de meses.
- **Mês**: estrutura responsável pela definição de um mês. Seus atributos são um identificador para o mês em questão, variando de 1 a 12 (janeiro, fevereiro, ..., dezembro), um inteiro representando o número de dias naquele mês e um um vetor alocado dinamicamente compondo a lista de compromissos para cada dia do mês.
- **Dia**: estrutura responsável pela representação da lista de compromissos de um dia, a qual é implementada através de listas encadeadas. Cada célula da lista é um compromisso de tipo próprio, de forma a possibilitar uma melhor organização do código.
- **Compromisso**: estrutura responsável pela representação de um compromisso, compondo o campo de dados das células da lista encadeada de compromissisos de um dia. Seus atributos são dois inteiros (dia e hora) e uma *string* (descrição do compromisso).

4 Compilação & Execução

De forma a facilitar a compilação e execução do trabalho desenvolvido, foi implementado um *Makefile*, localizado no diretório raíz do projeto. Abaixo estão os comandos necessários para compilar e executar o trabalho.

```
$ make
$ make run
```

5 Funções Implementadas

As funções implementadas para a agenda são aquelas descritas na especificação desse trabalho. Listamos abaixo a descrição de cada uma dessas funções.

- **Abrir agenda**: a primeira função básica é abrir a agenda. Nesse trabalho, os dados cadastrados são armazenados em um arquivo, de forma a permitir futura alterações com maior facilidade, bastando importar os compromissos já cadastrados. Vale ressaltar, entretanto, que a agenda só pode ser manipulada após ela ser aberta.
- **Inserir Compromisso**: insere um compromisso na agenda, levando em consideração o mês, dia e horário. Os compromissos sempre são armazenados no fim da lista.
- **Remover Compromisso**: remove um compromisso da agenda, utilizando como parâmetro para a busca o mês, dia, hora e minuto do compromisso a ser removido.
- **Listar Compromissos**: lista todos os compromissos cadastrados para um dia de um mês.
- **Checar Compromisso**: verifica se um dado compromisso existe na agenda, utilizando como parâmetro da busca o mês, dia, hora e minuto.
- **Fechar Agenda**: salva o estado atual da agenda em disco, de forma a possibilitar futuras edições após o fechamento do programa. Cada linha do arquivo criado contém a descrição de um compromisso, agrupando as informações referentes ao *dia*, *mês*, *hora*, *minuto* e *descrição* através de um separador (ponto e vírgula).
- **Sair**: fecha o programa, descartando todas as alterações realizadas.

6 Linguagem e Plataforma

A linguagem utilizada nesse trabalho foi *C++ 11* e o compilador foi o *g++*. Além disso, o trabalho foi desenvolvido e testado em ambiente Linux. De forma a tornar o código modular, utilizamos cabeçalhos *hpp* para a definição das estruturas e assinaturas de funções e arquivos *cpp* para a efetiva implementação, desacoplando a declaração da implementação.

7 Conclusão

Esse trabalho possibilitou uma melhor compreensão acerca dos tópicos estudados em sala. O conteúdo de listas encadeadas e, principalmente, a motivação para seu uso, foi bem trabalhado e contribuiu para o aprendizado do grupo. Além disso, destaca-se também a experiência positiva de desenvolver um trabalho modular em equipe.