Universidade Federal de Viçosa
Campus Florestal
CCF 211 – Algoritmos e Estrutura de Dados

Trabalho Prático 1
SISTEMA DE GERENCIAMENTO DE TEMPO

Ícaro Gabriel dos Santos Moreira – EF3856
Paula Teresa Mota Gibrim – EF4234
Samuel Aparecido Delfino Rodrigues – EF3476

Florestal 2021

Sumário

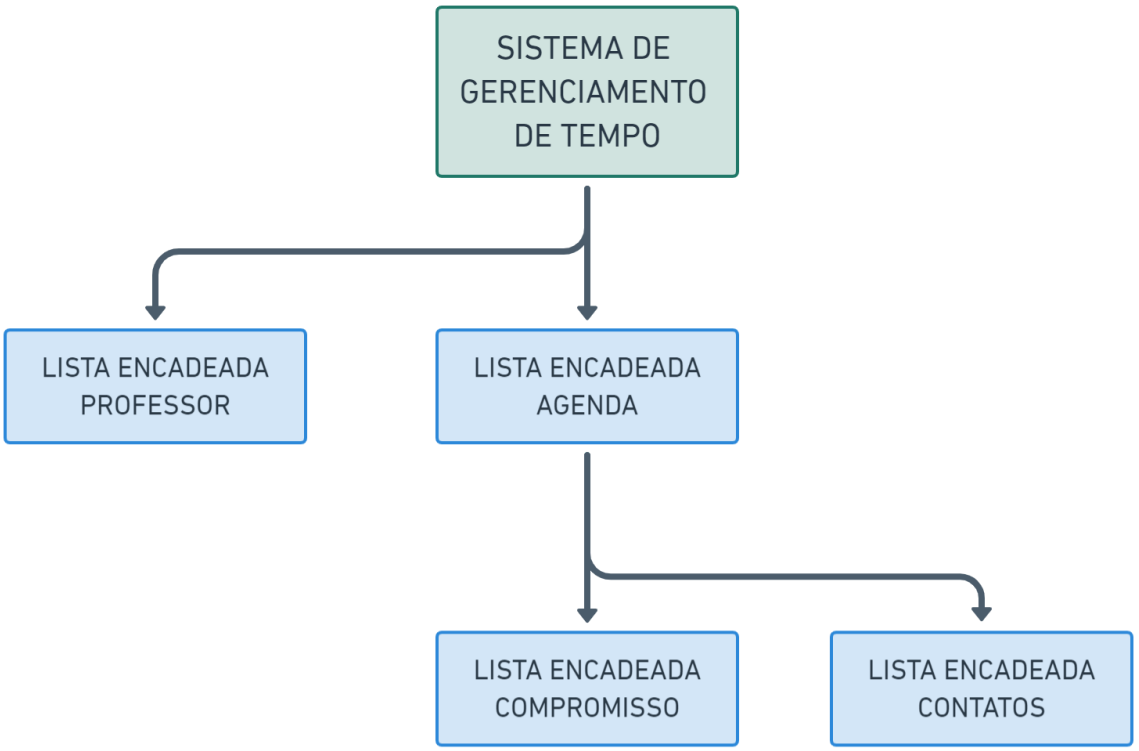
Introdução

O primeiro trabalho prático da disciplina de Algoritmos e Estruturas de Dados 1 tem como objetivo o desenvolvimento de um sistema de gerenciamento de tempo, com o intuito de solidificar o entendimento de conceitos relacionados a disciplina, mais especificamente, sobre TAD's e alocação dinâmica, por meio da implementação de listas encadeadas. O sistema possibilita que o usuário crie e gerencie uma agenda de compromissos e uma lista de contatos digital para cada professor cadastrado. O mesmo foi elaborado na linguagem C, devendo ser compilado preferencialmente usando o compilador GCC 9.2.0 e utilizando um sistema Linux com kernel 5.8.0 ou inferior.

Desenvolvimento

O trabalho foi dividido em várias partes para facilitar tanto o desenvolvimento quanto o entendimento do código. Algumas decisões tomadas para a criação do código foram fundamentais para o resultado obtido.

Primeiramente, optamos por não colocar a lista de professores dentro do TAD agenda. Isso se deu porque a nossa intenção era a de que cada agenda estivesse associada apenas a um único professor afim de manter a proporção 1:1, uma vez que, caso existisse uma lista de professores dentro da agenda, essa proporção não seria alcançada. Para criar um vínculo entre professor e agenda as duas compartilham o mesmo ID, este sendo único para um par professor/agenda. Já os outros TAD's ficam todos contidos no TAD agenda, pois entre eles e a agenda se estabelece uma proporção 1:N.



Hierarquia de Funções

É importante citar que, como uma agenda tem outras listas contidas, uma função com o objetivo de manipular, seja um compromisso ou um contato, é primeiramente definida no escopo da lista do tipo. Essas funções possuem o sufixo "base".

```
int alterar_prioridade_base(tipo_compromisso *compromisso, int novaprioridade);
```

função que altera a prioridade de um dado compromisso(escopo lista de compromissos)

Essas funções base, posteriormente, são chamadas no escopo da lista de agendas, uma vez que precisamos manipular uma agenda, no caso, a que contém o compromisso ou o contato que queremos trabalhar.

```
int altera_prioridade(tipo_lista_agenda *lista_agenda, int nova_prioridade, int ID_compromisso, int ID);
```

função que altera a prioridade(escopo lista de agendas)

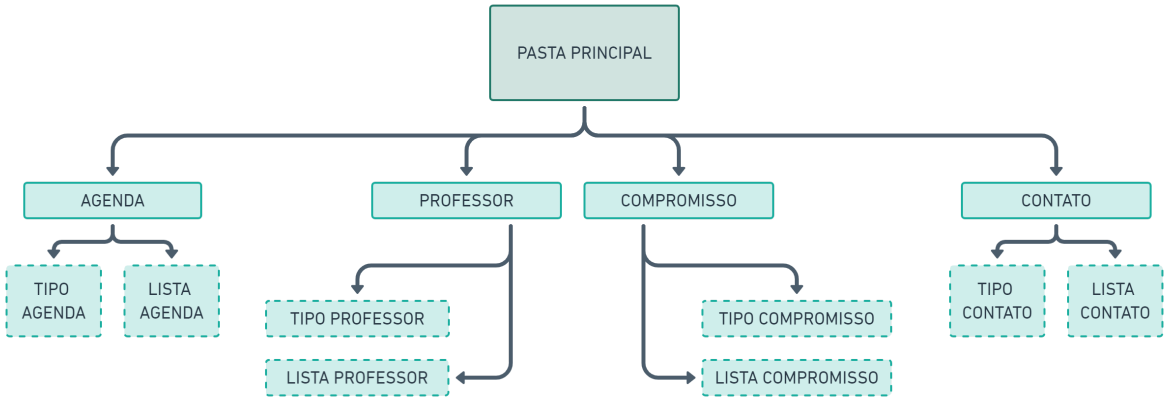
Podemos notar que a função “altera_prioridade” recebe uma lista de agendas como parâmetro, logo, se torna necessário encontrar uma agenda específica dentro da lista de agendas. Para isso, implementamos uma função que procura a agenda cujo ID foi passado. Essa função é chamada, nesse caso, dentro da função “altera_prioridade”. Achando a agenda desejada, um loop encontrará o compromisso que desejamos alterar a prioridade e, após isso, a função base é chamada efetivando a mudança.

```
celula_agenda* procura_agenda_id(tipo_lista_agenda *lista_agenda, unsigned int ID);
```

função que procura uma agenda específica

Hierarquia de Arquivos

Visando conseguir uma boa organização dos arquivos, eles foram divididos em uma pasta para cada um dos TAD’s e, dentro delas, foram divididos novamente em duas pastas: uma que armazena o tipo em si, com suas definições e funções mais básicas e outra dos tipos listas, a qual contêm as listas encadeadas e as operações sobre elas. Além disso, fazer dessa maneira contribuiu para manter a coesão sem prejudicar o acoplamento, cada qual com suas funções e definições de tipo.



flow-chart do esquema de pastas do trabalho

Módulos

Foram desenvolvidos nesse trabalho prático 4 módulos, sendo eles o TAD agenda, TAD professor, TAD compromisso e TAD contato, ambos os 4 utilizando listas encadeadas.

TAD agenda

O principal TAD do trabalho, que contém todos os outros com exceção do TAD professor. Portanto, qualquer operação que tem como objetivo manipular algum TAD contido dentro de uma agenda consequentemente manipula também o TAD agenda.

```
 9  struct agenda {
10
11      unsigned int ID;
12      unsigned int ano;
13      char *nome;
14
15      tipo_lista_compromisso lista_compromisso;
16      tipo_lista_contato     lista_contato;
17  };
```

tipo agenda, que contem todos os dados

Podemos notar nas linhas 15 e 16 que uma agenda possui uma lista de compromissos e uma lista de contatos.

TAD contatos

No contexto da agenda, um tad que armazena contatos seria adequado, pois contribuiria ainda mais para a organização. Essa implementação parte da ideia de que o professor que possui a agenda poderá inserir contatos referentes aos seus compromissos.

```
 5  struct contato {
 6
 7      char *nome;
 8      char *sobrenome;
 9      char *comentario;
10      char *email;
11      char *telefone;
12
13  };
```

tipo professor, que contém os dados de um professor

TAD compromisso

O TAD compromisso, como especificado, possibilita um professor com uma agenda cadastrada a criar compromissos com alguns dados como prioridade, data, horário e uma breve descrição, os mesmos sendo ordenados a partir da sua prioridade. Esse TAD suporta operações como cadastro de um novo compromisso, imprimir na tela seus dados e alterar a prioridade.

```
5   typedef struct{
6
7       int prioridade;
8       int identificador;
9       int dia;
10      int mes;
11      int ano;
12      int horas;
13      int minutos;
14      int duracaocompromisso;
15      char descricao[100];
16
17  } tipo_compromisso;
```

tipo compromisso, que contém os dados de um compromisso

TAD professor

Como foi especificado, o grupo deveria implementar mais 2 TAD's escolhidos pelo mesmo. O TAD professor se encaixa muito bem no contexto de um sistema para gerenciamento de tempo destinado a professores, visto que, nesse TAD constam informações que podem ser relevantes e serve como uma forma de controle para saber quais professores têm uma agenda no sistema.

```
3   struct professor {
4
5       char *nome;
6       char *curso;
7       char *esp;  // Pos, doutorado ou mestrado
8       char *CPF;
9
10      unsigned int senha;
11      unsigned int idade;
12      unsigned int ID;
13
14  };
```

tipo professor, que contém os dados de um professor

Conclusão

Ao realizar a implementação desse sistema tivemos a oportunidade de aprofundar os conceitos vistos em aula, de uma maneira prática que nos permitiu adquirir a noção de como eles seriam trabalhados numa situação de desenvolvimento real. Além disso, serviu para aprimorarmos nossos conhecimentos na própria linguagem, visto que foi necessário aprender outras técnicas de programação. Tivemos também a oportunidade de aprimorar a capacidade de trabalho em equipe, visto que, foi necessário delegar funções para que o trabalho fosse concluído com êxito.