

Algoritmos e Estruturas de Dados II

Trabalho Prático 1

Entrega: 09/09/07

Devolução: 23/09/07

Trabalho individual

A maioria dos professores do DCC se sente sobrecarregada com encargos e reuniões e acaba por não conseguir cumprir todos os seus compromissos. Tendo em vista o aumento esperado dos encargos didáticos no próximo ano, devido ao programa REUNI, o Departamento resolveu desenvolver um sistema automatizado de *gerenciamento de tempo*, e, para tanto, contrata você para fazê-lo. A especificação do sistema segue abaixo.

Tipos Abstrato de Dados

Você deve implementar **três** tipos abstratos de dados, definidos a seguir:

1. Compromisso

A agenda de um professor pode ter vários tipos de compromissos, definidos a partir de um identificador numérico de tipo bem como informações adicionais associadas. O sistema a ser desenvolvido deve suportar 5 tipos de compromissos:

- **AULA:** aulas para turmas de graduação, pós-graduação ou especialização. Este compromisso tem associado as seguintes informações: data, hora, duração (em minutos) e nome da disciplina ministrada.
- **ORIENTAÇÃO:** orientação de alunos. Este compromisso possui uma data, hora, duração (em minutos) e nome do aluno orientado.
- **REUNIÃO:** reuniões do departamento. Inclui data, hora, duração (em minutos) e nome da reunião (ex: COLEGIADO DA GRADUAÇÃO ou COPEQ)
- **EVENTO:** participação em eventos técnicos como congressos e conferências. Possui data, hora, duração (em dias), nome do evento. Você pode assumir que um evento começa as 0hs do dia inicial e terminam as 24hs do último dia.
- **COMPROMISSO PARTICULAR:** qualquer compromisso particular, que possui data, hora, duração (em minutos) e razão (ex: médico).

Você pode assumir que nome de disciplina, nome de aluno, nome de reunião, nome de evento e razão de compromisso particular são cadeias de caracteres, todas com um tamanho máximo $M = 100$.

Cada compromisso possui também um identificador numérico único, que servirá para realização de alterações após a criação do mesmo.

Cada tipo de compromisso tem um grau de prioridade que, por default, deve ser:

AULA 2	ORIENTAÇÃO 1	REUNIÃO 4
EVENTO 3	COMPROMISSO PARTICULAR 2	

Um professor deve poder alterar (aumentar ou diminuir) o grau de prioridade de qualquer compromisso. Os graus de prioridade deverão ser utilizados para definir quais compromissos serão cumpridos e quais deverão ser cancelados. Se dois compromissos têm conflito de horário, aquele com a maior prioridade deverá ser cumprido (Vide regras de desempate abaixo).

Um professor ainda poderá definir se um dado compromisso pode ser adiado ou não, definindo o campo *adiável* apropriadamente. Note que AULA e EVENTO não podem ser adiados.

Em caso de conflito, se o compromisso com menor prioridade for *adiável*, ele será adiado. Caso contrário, ele terá que ser cancelado (veja descrição abaixo). Neste caso, cada compromisso tem também um campo *status* associado informando se ele poderá ser cumprido, se ele deverá ser cancelado ou adiado. Em outras palavras, este campo pode assumir um dos código (valores numéricos) “A SER CUMPRIDO”, “ADIADO” ou “CANCELADO”.

Um compromisso deve suportar, *no mínimo*, as seguintes operações:

- `inicializaCompromisso(tipo, data, hora, duração e nome)`: inicializa um compromisso com os valores informados. Esta função deve atribuir um identificador numérico único para o compromisso e retorná-lo como resultado. Além disto, no momento de criação, todo compromisso deve poder ser cumprido e não pode ser adiado. Logo, o *status* de um compromisso recém-criado será sempre “A SER CUMPRIDO”, e o campo *adiável* deverá ser inicializado com FALSE.
- `alteraPrioridade(novaPri)`: altera a prioridade de um compromisso com um novo valor passado como parâmetro.
- `retornaPrioridade()`: retorna a prioridade de um compromisso.
- `eAdiavel(flag)`: define se compromisso passado como parâmetro é adiabél ou não. O parâmetro *flag* deve ser TRUE se compromisso é adiabél e FALSE caso contrário.
- `temConflito(compromisso1, compromisso2)`: esta operação deve retornar se há ou não conflito nos horários dos dois compromissos passados como parâmetros.
- `atribuiStatus(flag)`: determina se o compromisso poderá ser cumprido, se ele deverá ser cancelado ou adiado. O parâmetro *flag* deve conter um dos três valores “A SER CUMPRIDO”, “ADIADO” ou “CANCELADO”.
- `retornaStatus()`: retorna se compromisso poderá ser cumprido, se ele deverá ser cancelado ou adiado.
- `imprimeCompromisso()`: imprime todas as informações associadas a um compromisso seguindo a ordem identificador, tipo, data, hora, duração, nome de

disciplina/aluno/reunião/evento (ou razão), *adiável*, *status*. Você deve imprimir o tipo na forma de cadeia de caracteres (ex: REUNIÃO) e não o valor numérico.

2. Agenda

O sistema sendo desenvolvido deve permitir a criação de agendas, uma para cada professor.

Assim, cada agenda deve conter

- um identificador único do professor,
- o nome do professor,
- o ano,
- uma sequência de eventos. Você pode assumir um número máximo, pré-definido, de eventos no ano em cada agenda.

Um professor deve ser capaz de, *no mínimo*, realizar as seguintes operações com sua agenda:

- `criaAgenda(idProf, nome, ano)`: cria uma nova agenda para o professor com identificador e nome informados. O ano também é passado como parâmetro. Esta operação deve criar uma agenda vazia, sem nenhum compromisso.
- `recuperaAgenda(data)`: recebe uma data, e retorna uma mensagem na tela com o nome do professor, o ano e o número de compromissos na agenda com data após a data fornecida.
- `insereCompromisso(tipo, data, hora, duração, nome)`: recebe as informações associadas de um novo compromisso, que deve ser inicializado (chamando a operação respectiva no TAD Compromisso) e inserido na agenda. A operação deve imprimir uma mensagem informando sucesso ou falha. Em caso de sucesso, o identificador do compromisso deve ser impresso.
- `removeCompromisso(idCompromisso)`: recebe o identificador de um compromisso. Esta operação deve remover o compromisso identificado da agenda do professor, caso ele exista.
- `imprimeAgenda()`: esta operação deve imprimir todos os compromissos agendados do professor, ordenados cronologicamente pelo instante de início. Em caso de empate, deve-se imprimir primeiro os compromissos com maior prioridade. Se ainda houver empate, deve-se respeitar a ordem de impressão: EVENTO, REUNIÃO, AULA, ORIENTAÇÃO, COMPROMISSO PARTICULAR.
- `resolveConflito()`: esta operação deve visitar toda a agenda a procura de conflitos entre compromissos. Cada conflito encontrado deve ser resolvido a partir da atribuição de *status* de CANCELADO ou ADIADO para alguns compromissos. A resolução de conflitos deve seguir a ordem de prioridade dos eventos. Em caso de empate, deve-se priorizar compromissos que iniciam primeiro. Caso ainda haja empate, deve-se seguir a mesma ordem de impressão, informada acima.
- `retornaNAdiamentos()`: retorna o número total de compromissos com status “ADIADO”.

- `retornaNCancelamentos()`: retorna ao número total de compromissos com status “CANCELADO”.
- `retornaNCompromissos()`: retorna o número total de eventos na agenda.
- `imprimeCompromissosAdiados()`: esta operação deve imprimir todos os compromissos agendados que foram adiados, seguindo a mesma ordem da operação `imprimeAgenda`.
- `imprimeCompromissosCancelados()`: esta operação deve imprimir todos os compromissos agendados que foram cancelados devido a conflitos, seguindo a mesma ordem da operação `imprimeAgenda`.
- `imprimeCompromissosACumprir()`: esta operação deve imprimir todos os compromissos que ainda pode ser cumpridos, seguindo a mesma ordem da operação `imprimeAgenda`.

Note que a declaração exata das operações listadas tanto para o TAD Compromisso quanto para o TAD Agenda fica a critério dos alunos. Lista-se acima apenas as funcionalidades mínimas esperadas bem como parâmetros essenciais para suas execuções. Caso o aluno sinta necessidade de criar novas operações, ou de modificar a lista de parâmetros das apresentadas, as modificações deverão ser discutidas e justificadas no relatório.

Sistema de Gerenciamento de Tempo

O sistema de gerenciamento de tempo a ser desenvolvido deve permitir a criação, atualização e visualização das agendas de todos os $P = 60$ professores do departamento. O sistema deve ser interativo permitindo que o usuário (isto é, um professor) defina qual operação deseja realizar em cada interação. Em outras palavras, o sistema deve fornecer um menu para que o usuário interaja com o sistema e eventualmente saia do mesmo. Escolhida a operação, o sistema deverá solicitar que o usuário entre com as informações necessárias para a sua realização e imprima os resultados (se houver) na tela.

O menu não precisa ser sofisticado, bastando uma interface ASCII simples com código de escolha de cada operação que o usuário *pode* fazer em cada instante. Testes de corretude e de sanidade deverão ser realizados em cada operação.

O sistema deve poder ser utilizado por todos os P professores, embora o acesso seja sequencial.

Apresente uma descrição do TAD, com as estruturas de dados e funções/procedimentos implementados. Apresente também a ordem de complexidade de cada função ou procedimento implementado.

As regras de submissão estão no site da disciplina. Em particular, atente para:

- O programa deve ser organizado em, *no mínimo*, 3 módulos: `compromisso.c` (`compromisso.h`), `agenda.c` (`agenda.h`) e `main.c`
- O programa deve estar bem indentado e comentado
- O programa não deve fazer uso de comando *goto* nem de variáveis globais

- Caso apareçam números fixos no código, estes devem ser definidos como constantes.
- O trabalho é *individual*. Trabalhos copiados serão penalizados conforme informado em sala.

ATENÇÃO: Soluções que não correspondam à implementação de um **Tipo Abstrato de Dados** serão duramente penalizadas por **não** atenderem a especificação.