Trabalho 1
Agendador de Reunião

Disciplina: 5200 – Paradigma de Programação Lógica e Funcional

Professor: Lucas Pupulin Nanni

Introdução

O objetivo deste trabalho é a implementação de um agendador de reunião em Racket.

O trabalho é em equipe de até duas pessoas. O compartilhamento de informações entre as equipes é permitido (e aconselhado), mas o compartilhamento de código não é permitido. Trabalhos que tenham porções significativas de código iguais, ou copiadas da internet, serão anulados. Veja a resolução Nº 008/2007-COU para as possíveis sanções disciplinares.

Este trabalho vale 2/3 (dois terços) da nota do primeiro bimestre.

Data de entrega: até o dia 15/06/2016 às 23:55.

Descrição

Um agendador de reunião é um programa que encontra horários compatíveis para diversas pessoas realizarem uma reunião. O agendador recebe como entrada os horários de disponibilidade (semanal) de cada pessoa e a duração da reunião, e encontra (se existir) os possíveis horários para realizar a reunião.

Este trabalho consiste na implementação de um agendador de reunião em Racket. Para exercitar os conceitos do paradigma funcional, só é permitido usar funções com efeito colateral para as operações de entrada e saída, o restante do programa deve ser puramente funcional. Algumas funções com efeito colateral em Racket tem o nome terminado com !. Estas funções não podem ser utilizadas.

O programa deve receber como parâmetro na linha de comando a duração da reunião e uma lista de arquivos com as disponibilidades das pessoas, e escrever na saída padrão os possíveis horários para a realização da reunião. Se não existir nenhum horário que seja possível realizar a reunião, o programa não deve escrever nada na saída padrão.

Cada linha de um arquivo de disponibilidade consiste de um dia da semana seguido por uma sequência de intervalos. Por exemplo, a linha

```
seg 08:30-10:30 15:45-17:15
```

significa que a pessoa tem na segunda-feira os horários das 8:30 às 10:30 e das 15:45 às 17:15 livres para a reunião.

Considere os arquivos `a`

```
seg 08:30-10:30 14:03-16:00 17:10-18:10
ter 13:30-15:45
qua 11:27-13:00 15:00-19:00
sex 07:30-11:30 13:30-14:00 15:02-16:00 17:20-18:30
```

e `b`

```
seg 14:35-17:58
ter 08:40-10:30 13:31-15:13
qui 08:30-15:30
sex 14:07-15:00 16:00-17:30 19:00-22:00
```

Se o programa for chamado com os parâmetros `00:45 a b`, o programa deve escrever na tela a seguinte resposta:

```
seg 14:35-16:00 17:10-17:58
ter 13:31-15:13
```

Observe que o formato da saída é o mesmo dos arquivos de entrada.

Desenvolvimento

Para facilitar o desenvolvimento dos programas, está disponível no Moodle um "projeto" com os arquivos iniciais de código, um programa testador e um Makefile para testar e preparar o envio dos programas.

No diretório `src/`, estão os arquivos com o código Racket inicial. Você deve escrever o seu código no arquivo `src/reuni.rkt`. O arquivo `src/reuni-testes.rkt` contém os testes unitários.

Execução dos testes

Para executar os testes unitários, abra o arquivo **src/reuni-testes.rkt** no DrRacket e clique no botão Correr / Run (ou Ctrl-R).

Os testes funcionais devem ser executados no terminal. Para executar os testes, entre no diretório **reuni** e execute um dos comandos

```
$ make teste-unitario   # executa os testes unitários
$ make teste-funcional  # executa os testes funcionais
$ make teste            # executa os testes unitários e funcionais
```

Envio do trabalho

Para enviar o trabalho, primeiramente crie uma versão compactada do código fonte executando o comando

```
$ make zip
```

Após a execução desse comando, será criado o arquivo **src.zip**. Renomeie o arquivo de forma a conter o RA dos integrantes da equipe, como no exemplo a seguir.

Exemplos:

- ra12345_ra54321.zip (dois integrantes)
- ra12345.zip (um integrante)

O arquivo compactado deve ser enviado até às 23:55 do dia 15/06/2016 pelo Moodle. Arquivos compactados que não seguirem esse procedimento serão desconsiderados.

Avaliação

Este trabalho vale 2/3 (dois terços) da nota do primeiro bimestre. O trabalho será avaliado de acordo com os critérios:

- Clareza lógica: a lógica nos programas deve ser clara.
- Corretude e completude: os programas têm que passar em todos os testes.
- Boas práticas de programação: o código deve estar bem escrito e organizado; os recursos da linguagem devem ser usados corretamente.

Dicas

Esta seção contém algumas dicas de implementação. Use as que achar útil.

- Antes de implementar o programa em Racket, resolva alguns problemas da seção 1 e 2 da lista de 99 problemas em Prolog (para fazer em Racket).
- Implemente o programa na ordem que os testes são executados.
- Aprenda a usar o debugger do DrRacket, será útil.

Créditos

A especificação deste trabalho consiste de uma versão adaptada do trabalho proposto pelo professor Marco A. L. Barbosa.