Universidade Estadual de Maringá – Departamento de Informática
1028 – Linguagens de Programação – Ciência da Computação
1225 – Linguagens de Programação – Informática
Prof. Marco Aurélio

Trabalho do 4º bimestre

1 Introdução

O objetivo deste trabalho é praticar a programação funcional escrevendo uma calculadora em racket.

Além de entregar o código do trabalho, o aluno deverá apresentar o trabalho para o professor. O aluno que demonstrar falta de conhecimento do código, ficará com nota zero.

O trabalho é individual. O compartilhamento de informações entre os alunos é permitido (e aconselhado), mas o compartilhamento de código não é permitido. Trabalhos que tenham código igual serão anulados. Veja a resolução N° 008/2007-COU para as possíveis sansões disciplinares.

Data de entrega: até o dia 28/11/2011 às 23:00h

Forma de entrega: o trabalho deve ser enviado para o email `malbarbo@gmail.com`, em um arquivo compactado com o nome `codigodisciplina_ra.zip` (substitua `codigodisciplina` pelo código da disciplina e `ra` pelo número do seu RA). O formato de compactação deve ser zip, outros formatos não serão aceitos. Tenha o cuidado de **não** enviar **arquivos compilados**.

Apresentação: cada aluno deve agendar um horário com o professor para fazer a sua apresentação. A data limite para a apresentação é 01/12/2011.

2 Descrição

Desenvolver uma calculadora na linguagem racket. A calculadora deve ler expressões na forma infixa ("convencional") e calcular o resultado da expressão.

A calculadora deve aceitar expressões matemáticas (inclusive com parênteses) com números inteiros na base 10 e com os seguintes operadores binários:

Operador	Operação	Associatividade	Prioridade
$+$	soma	esquerda	10
$-$	substração	esquerda	10
$*$	multiplicação	esquerda	20
$/$	divisão	esquerda	20
$\wedge$	exponenciação	direita	30

Na página da disciplina está disponível para download o testador para a calculadora (`calc-tests.rkt`) e o esqueleto do programa (`calc.rkt`). Para executar o testador, abra o arquivo `calc-tests.rkt` no drracket e clique em "Correr".

O programa deve realizar o cálculo da expressão em três fases:

1. Transformar a string que representa a expressão em uma lista, função `parse-expression`.
2. Converter a expressão (em forma de lista) para a notação infixa, função `infix-to-prefix`.

3. Avaliar uma string que representa uma expressão (usando as duas funções anteriores), função `eval-infix`.

O código do programa deve ser escrito no arquivo `calc.rkt`. Veja o exemplo em `exemplo-eval.rkt` de como o eval pode ser usado (necessário para a fase 3). Não é necessário fazer validação das expressões, todas as expressões no teste de unidade são válidas.

Atenção: Você pode utilizar funções pré-definidas do racket, mas não é permitido usar funções de mudança de estado. O programa deve ser puramente funcional (com exceção do namespace usado na função eval).

3 Avaliação

O trabalho será avaliado de acordo com os critérios:

- Corretude e completude: a calculadora tem quer 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.

4 Desafio

Adicionar a calculadora os operadores unários pré-fixos $-$ e $+$ com prioridade 25, o operador unário pós-fixo ! (fatorial) com prioridade 27, e as função pré-fixas min e max, que podem receber qualquer quantidade de parâmetros entre parênteses separados por vírgula, com prioridade 40. Criar também um arquivo chamado `calc-tests-desafio.rkt` com os testes para estes novos operadores.

Este desafio vale 1,0 extra na nota do bimestre.

5 Dicas

As informações abaixo são apenas dicas. Analise-as individualmente e verifique quais são interessantes.

- Antes de começar a fazer o trabalho, leia o capítulo do livro sobre linguagens funcionais, depois leia os capítulos 1, 2, 3.1-3.8, 4.1-4.7 do guia do racket e escreva o código para os exemplos do livro. Tentar escrever o código deste trabalho sem ter escrito outros programas mais simples, é bastante difícil.
- Para representar um registro, você pode usar uma lista. Por exemplo, para representar um ponto cartesiano, você utiliza uma lista com dois elementos, como `'(1 2)`. Para consultar os valores x e y escreva duas duas funções `ponto-x` e `ponto-y`, que recebem como parâmetro um ponto (lista com dois valores) e retorna o valor correspondente.
- Para representar uma tabela associativa, você pode usar uma lista de pares, veja a função assoc.
- Para fazer repetição em uma linguagem funcional, você deve usar funções recursivas. O caso base da função recursiva em geral é a condição de parada da repetição. Veja o exemplo, que acompanha o download desta atividade, para encontrar o valor máximo e o valor mínimo de uma lista. Leia com cuidado a versão imperativa, escrita em python, e a versão funcional, escrita em racket. Entenda a relação entre as versões.

- O algoritmo Shunting-yard pode facilmente ser adaptado para converter uma expressão na forma infixa para a forma prefixa.
- Antes de escrever o código, tenha certeza de que o seu algoritmo funciona (tanto para fase 1 como para a fase 2). Faça testes de mesa para você ter certeza de que entendeu o algoritmo. A dificuldade desta atividade não é o algoritmo, mas a escrita do algoritmo em uma linguagem funcional. Caso o seu programa não esteja funcionando corretamente, use (entenda como ele funciona primeiro) o debugador integrado do drracket.
- Escreva o código do programa para passar nos testes na ordem em que eles são executados. Só vá para o próximo caso de teste quando o atual estiver funcionando.