Tipos abstratos

8.1 Acrescente casos à função `parentAux` apresentada na aula teórica de forma a que `parent` verifique se os parêntesis curvos, retos e chavetas numa cadeia estão correctamente casados; por exemplo:

`parent "[(1+2)*{3}+4]"` = `True` `parent "[1+2)*3"` = `False`

Para experimentar o seu programa deve descarregar o módulo *Stack* apresentado na aula teórica.

8.2 A *notação polaca invertida* (em inglês: "*reverse polish notation*" e abreviado para *RPN*) coloca cada operador aritmético após os dois operandos; por exemplo, a expressão $42 \times 3 + 1$ escreve-se "`42 3 * 1 +`". Nesta notação não necessitamos de parêntesis nem de precedências entre operadores.

Neste exercício pretende-se implementar uma calculadora RPN usando o módulo *Stack* apresentado nas aulas; a ideia é partir a expressão em palavras usando `words` e percorrer a lista de palavras operando sobre uma pilha auxiliar de `Floats`:

- se a palavra for uma operação `+`, `-`, `*`, `/` removemos os dois valores no topo da pilha, efetuamos a operação correspondente e empilhamos o resultado;
- se a palavra for uma sequência de algarismos, empilhamos o seu valor (usando `read` para converter uma `String` num `Float`).

Se a expressão RPN estiver bem formada, no final da percorrer a lista de palavras devemos ter na pilha um único valor que é o resultado da expressão.

(a) Escreva uma função `calcular` :: `String` → `Float` que calcula o valor duma expressão em RPN; por exemplo: `calcular "42 3 * 1 +" == 127`.

(b) Escreva um programa principal que leia uma expressão em RPN da entrada padrão como uma cadeia de caracteres da entrada padrão e calcule o seu valor. Experimente o programa com expressões correctas e incorrectas e interprete os resultados.

8.3 Considere a implementação de conjuntos como árvores de pesquisa; pretende-se completar a operação de união entre conjuntos

`union` :: `Ord` $a \Rightarrow$ `Set` $a \rightarrow$ `Set` $a \rightarrow$ `Set` a

Sugestão: defina a união por recursão sobre o primeiro conjunto; há dois casos a considerar.

```
union Empty                b = ...  -- caso base
union (Node x left right) b = ...  -- caso recursivo
```

Complete a definição acima e efetue alguns testes para verificar a sua implementação.

8.4 Considere novamente a implementação de conjuntos como árvores de pesquisa; vamos agora implementar a operação de interseção

$$\texttt{intersection} :: \texttt{Ord}\ a \Rightarrow \texttt{Set}\ a \rightarrow \texttt{Set}\ a \rightarrow \texttt{Set}\ a$$

Defina esta operação por recursão sobre o primeiro conjunto; desta vez será útil considerar três casos:

```
intersection Empty                b = ...     -- caso base
intersection (Node x left right) b
   | member x b  = ...                -- caso recursivo 1
   | otherwise   = ...                -- caso recursivo 2
```

Complete a definição acima e efetue alguns testes para verificar a sua implementação.

8.5 Pretende-se acrescentar ao módulo *Set* apresentado na aula teórica uma função para filtrar conjuntos por um predicado:

$$\texttt{filter} :: \texttt{Ord}\ a \Rightarrow (a \rightarrow \texttt{Bool}) \rightarrow \texttt{Set}\ a \rightarrow \texttt{Set}\ a$$

Implemente esta função por recursão sobre o conjunto usando a esquema seguinte:

```
filter p Empty = ...   -- caso base
filter p (Node x left right)
   | p x =  ...      -- caso recursivo 1
   | otherwise = ... -- caso recursivo 2
```

Nos exercícios seguintes pretende-se que use as implementações de tabelas e conjuntos em Data.Map e Data.Set; se tiver instalado a "Haskell Platform" deve conseguir importar estes módulos.

8.6 Usando o módulo *Data.Map* escreva um programa para contar o número de ocorrências de letras num texto lido da entrada padrão. Este tipo de tabela chama-se um *histograma*.

Sugestão: represente a histograma de ocorrências usando uma tabela de associações `Map Char Int`. Tenha o cuidado de normalizar as letras todas para uma mesma forma (i.e. todas maiúsculas ou todas minúsculas). Pode usar as funções *isLetter*, *toUpper* e *toLower* do módulo *Data.Char*.

8.7 Modifique o programa verificador de ortografia dum exercício da folha 6 para conjuntos do módulo *Data.Set* em vez de listas para representar o conjunto das palavras dos dicionário. Como o dicionário é extenso e os conjuntos em *Data.Set* são implementados com árvores auto-equilibradas, a verificação de ortografia deverá ficar consideravelmente mais rápida do que com listas.