Abordagem ao paradigma Funcional em Haskell e C++

Explicação genérica sobre o paradigma funcional

Como dado a entender na introducao, a ideia fulcral do paradigma Funcional e a composicao. Enquanto que no paradigma Imperativo/Procedimental e comum um procedimento ser uma longa lista de instrucoes a executar, sendo tanto o "input" como "output" implicitos. Com isto quer-se dizer que os procedimentos podem ter acesso a estado global e/ou partilhado entre varios procedimentos. Esta partilha nao esta especificada de forma nenhuma, e o programador que tem o trabalho de mentalmente verificar que o seu uso esta dentro do expectavel.

O paradigma Funcional evita este problema parcial ou completamente, ao desencorajar ou impedir esta practica, e ao mesmo tempo encorajar e facilitar "boa practica".

Um exemplo extremo e pouco realista seria:

```
void accoes (void)
{
    accao1();
    accao2();
    accao3();
}
```

Deste pequeno excerto, podemos concluir que:

1. Como nenhum dos procedimentos `accao1`, `accao2` ou `accao3` recebe argumentos, e o resultado nao e utilizado, entao estes procedimentos nao fazem nada de util, e portanto, `accoes` tambem nao faz nada de util;
2. Ou, cada um dos procedimentos faz algo de util, mas para tal acede e altera alguma estrutura de dados partilhada; esta relacao input-output nao e explicita.

Em contraste, numa linguagem funcional escreveriamos (em notacao `Haskell`) `accoes = accao3 . accao2 . accao1` para representar a mesma sequencia de accoes, mas sem partilha de memoria nem estruturas de dados a serem mutadas: cada uma das accoes e uma funcao que devolve uma estrutura de dados, dada outra estrutura de dados.

Este problema de alteracao implicita de estado agrava-se ainda mais num contexto concorrente, num modelo "tradicional", com threads e partilha de memoria.

Haskell como linguagem funcionalmente pura

Breve descrição sobre como pensar funcionalmente em C++