Capítulo 1

Imutabilidade vs mutabilidade

Uma das características mais importantes do paradigma funcional, nomeadamente na linguagem Haskell pois, existem linguagens funcionais impuras, é a noção de imutabilidade das expressões.
Isto faz com que, por exemplo, não seja possível alterar o valor de variáveis já existentes mas sim, criar novas variáveis com os novos valores.
Analisando o excerto de código a seguir podemos verificar que quando *action* é invocado, o valor impresso é o primeiro valor de *a=123* pois é o valor avaliado para *a* até então.

```
> a = 123
> action = print a
> a = 456
> action
123
```

A linguagem *c++* tenta também lidar com esta noção de imutabilidade.
A noção de funções puras é dissecada para uma noção de *referential transparency* bem como as expressões.
Uma expressão é referencialmente transparente se o programa não se comportar de maneira diferente ao substituír a expressão inteira apenas pelo seu valor de retorno. Então, se uma expressão é referencialmente transparente, não tem efeitos colaterais observáveis e, portanto, todas as funções usadas nessa expressão são consideradas puras.
A ideia de imutabilidade é particularmente útil em ambientes em que se gera concorrência, pois, existem variáveis partilhadas que podem gerar comportamentos inesperados nos programas se não for devidamente protegida a sua alteração. Em *c++* está disponível a keyword *const* que permite controlar a imutabilidade de uma variável. Ao declarar uma variável *const x* estamos a dizer ao compilador que esta variável é imutável e, qualquer tentativa de alteração à variável irá originar um erro. De seguida analisamos a declaração de uma variável *const* e os possíveis erros que podem ser cometidos.

```
const std::string name{"John Smith"};
std::string name_copy = name;
std::string& name_ref = name; // erro
const std::string& name_constref = name;
std::string* name_ptr = &name; // erro
const std::string* name_constptr = &name;
```

falta fazer a explicação de cada um dos erros

No entanto, por vezes existe necessidade de ter variáveis mutáveis mas que, em determinados casos, a sua alteração esteja protegida. Esse controlo pode ser feito recorrendo a mutexes que permitem o controlo de concorrência em determinadas zonas crítica de código.
Esta abordagem será aprofundada mais à frente na secção de concorrência.