O fluxo do programa

Em linguagens de paradigma procedural, também chamadas de imperativas, controlamos o fluxo do programa através de condicionais, loops e variáveis de controle que acabam por definir quando e enquanto um bloco de código será executado. Em Haskell, não usamos loops nem variáveis de controle, mas casamento de padrões, guardas e bastante recursão.

```
void maiorQue10 ( int x ) {
    if ( x > 10 ) {
        printf (" Eh maior mesmo\n" );
    } else {
        printf ( "Nao eh maior nada\n" );
    }
}

void printXVezes ( int x, char y[] ) {
    int i;
    for ( i=0; i<x; i++ ){
        printf ("%s\n", y);
    }
}
```

Caso as funções acima, escritas em C, caso fossem escritas em Haskell, provavelmente seriam algo assim:

```
maiorQue10 :: Int -> IO()
maiorQue10 x
    | x > 10 = putStr "Eh maior mesmo\n"
    | otherwise = putStr "Nao eh maior nada\n"

printXVezes :: Int -> String -> IO()
printXVezes x y = putStr (printXVezes2 x y)

printXVezes2 :: Int -> String -> String
printXVezes2 x y
    | x == 0 = ""
    | otherwise = ( y ++ "\n" ) ++ ( printXVezes2 ( x - 1 ) y )
```

A função maiorQue10 traz os mesmos resultados nas duas implementações. Recebe um inteiro e imprime uma string no console informando se esse inteiro é ou não é maior que 10. Em Haskell, os 'if' e 'else if' dão lugar a '|' e o else é representado pelo '| otherwise'.

A função printXVezes recebe um inteiro x e uma string y e imprime y por x vezes. Em C, isso é feito com um simples loop. Em haskell, precisamos criar uma recursão onde a condição de parada é é igual a 0?" e, a cada vez que a função é chamada, o x vai como x-1. O resultado é que x strings são concatenadas uma a outra até que a condição de parada seja atingida.

Então, na prática, quando precisamos executar algo várias vezes (ideia de loop), utilizamos a boa e velha recursão, lembrando que essa recursão precisa de uma condição de parada. Para essa condição de parada, as vezes utilizamos casamento de padrões. A função printXVezes2, inclusive, poderia ser reescrita da seguinte forma:

```
printXVezes3 :: Int -> String -> String
printXVezes3 0 _ = ""
printXVezes3 x y = ( y ++ "\n" ) ++ ( printXVezes3 ( x - 1 ) y )
```

Se o inteiro em questão for 0, independente da string, será retornado "" pondo um fim na recursão. Na segunda linha, diferente da primeira, ambos string e inteiro precisam ser referenciados. Portanto, precisamos nomeá-los. Nesse caso, de "x" e "y".