Linguagens de Programação

Fabio Mascarenhas - 2017.2

http://www.dcc.ufrj.br/~fabiom/lp

Corotinas

- Uma *corotina* é como uma função que pode suspender a sua execução, retornando ao chamador mas permitindo que a execução seja retomada do ponto onde parou:

```
fun gen(i)
  let n = i in
    while 0 < n do
      yield n;
      n := n - 1
    end
  end
end
```

```
let c = coro gen(10) in
  resume c +
  resume c +
  resume c
end
```

- A primitiva `coro` cria uma corotina a partir de uma chamada de função; a primitiva `resume` inicia/retoma a execução da corotina, e a primitiva `yield` suspende a execução, passando um valor de volta para o chamador

resume

- Para “saltar” para algum ponto do programa basta que guardemos a continuação daquele ponto, e então usamos ela ao invés da continuação atual
- Isso nos dá uma estratégia para implementar as corotinas e resume/yield: uma corotina é a continuação para a qual vamos saltar no resume
- Precisamos de mais um pedaço de estado global: a corotina atual
- Quando entramos em uma corotina, ela vira a corotina atual, e salvamos a anterior na própria estrutura de dados da corotina
- Corotinas não são reentrantes! Não podemos dar resume em uma corotina que não foi suspensa por um yield

yield

- Para sair de uma corotina, restauramos a corotina anterior e saltamos para a sua continuação, depois de guardar a continuação atual
- Vamos deixar o que acontece com uma corotina que chegou ao final sem ter dado um yield indefinido, por enquanto
- Em MicroC, cada corotina também precisa de sua própria seção da memória para sua pilha, e seu próprio stack pointer!
- Também precisamos guardar a pilha de corotinas no tratador de exceção, para que uma exceção restaure corretamente o estado antes do catch