Programação em linguagem estruturada: estruturas de controle

Junior Rodrigues Ribeiro

USP — ICMC

06 de Fevereiro de 2020

E-mail: `jrodrib@usp.br`

Conteúdo

Estruturas de controle

São três as estruturas de controle que modelam um programa estruturado:

- Sequência (**vá para (goto)***);
- Decisão (se-então-senão, interruptor-caso-padrão);
- Repetição (faça-enquanto, enquanto, para);

Todas elas são importantes e são usadas com muita frequência.

Conteúdo

Sequência

As estruturas de sequência, como o próprio nome diz, indicam que o programa será executado linha após linha, na mesma sequência em que os comandos aparecem.

Antigamente, na programação não estruturada (antecessora da estruturada) havia a possibilidade de declarar seções de código explicitamente, e então mudar o curso de execução do programa com o **vá para (goto)**. Essa prática tornava os programas muito difíceis de ler, e por isso é incisivamente desencorajada.

O próximo exemplo ilustra como o *goto* é problemático.

Uso do **goto**: dificulta a leitura do código pelo programador

Não é recomendado usá-lo!

```
!programa para calcular soma = ∑_{k=1}^{5} k
contador = 0
soma = 0
goto seção2
seção1:
escreva (soma)                          15
escreva ("fim do programa")             fim do programa
goto seçãoFim
seção2:
contador = contador + 1
soma = soma + contador
se(contador < 5) goto seção2
senão goto seção1
seçãoFim:
escreva("Fim :)")                       Fim :)
```

Conteúdo

Decisão/Seleção

As estruturas de decisão modificam o fluxo de execução, de acordo com o valor lógico de uma expressão lógica avaliada.

- O exemplo mais notório é o *se-então-senão* (if-then-else).
  se (expressão lógica) é verdadeira, então execute
  (sequência)
  senão*, execute
  (sequência)
  fim se
- Outra estrutura muito importante é o *interruptor-caso-padrão* (switch-case-default), que é uma sequência de comparações, dividindo o fluxo de execução em uma porção de outros fluxos, conforme cada caso. Uma determinada variável é selecionada e seu valor é comparado com cada caso.

interruptor-caso-padrão (switch-case-default)

```
interruptor (variável):
        caso valha x,
                (sequência)
        caso valha y,
                (sequência)
        caso valha z,
                (sequência)
        caso padrão*,
                (sequência)
fim interruptor
```

Os itens **senão** (else) e **caso padrão** (default) não são obrigatórios, e podem ser omitidos.

Conteúdo

As estruturas de repetição geralmente combinam uma expressão lógica e uma sequência de comandos. São elas: *para*, *enquanto* e *faça-enquanto* (for, while, do-while).

- Para: é uma estrutura utilizada para quando o número de repetições é conhecido. Neste caso, as repetições são realizadas até que o número seja atingido. Outras implementações trabalham com estruturas de dados iteráveis, de forma que a estrutura "para" é repetida uma vez para cada elemento da estrutura de dados, até que ela seja esgotada.

```
para x de 1 até 10, execute
        (sequência)
fim para
```

ou

```
para aluno em listaDeAlunos,
        (sequência)
fim para
```

- Enquanto: a estrutura "enquanto" é utilizada sempre que não se sabe a priori quantas vezes precisamos da repetição. Uma condição lógica é útil para que a repetição termine em algum momento. Do contrário, o programa executará para sempre. Assim, nessa estrutura, precisamos de uma condição de parada, ou seja, enquanto a expressão lógica for verdadeira, a sequência é executada.

  enquanto (expressão lógica) for verdadeira, execute

  (sequência)

  fim enquanto

  Essa estrutura avalia primeiro a expressão lógica para então dar sequência. Caso a expressão lógica já seja inicialmente falsa, a sequência não será executada.

- Faça-enquanto: a estrutura "faça-enquanto" só tem uma diferença em relação à estrutura "enquanto": ela executa a sequência de instruções para depois verificar a expressão lógica (a sequência é executada pelo menos uma vez). É frequentemente usada em menus.

  faça

  (sequência)

  enquanto (expressão lógica) for verdadeira

Conteúdo

Aninhamento de estruturas (nesting)

As estruturas apresentadas podem também ser usadas repetidas vezes e combinadas. Um bom exemplo é escrever uma matriz (3x5) na tela.

```
para linha de 1 até 3,
        para coluna de 1 até 5,
                escreva( matriz[linha][coluna] + _espaço)
        fim para
        escreva(_quebra_de_linha)
fim para
```

<Teste de mesa>

Outro exemplo: escrever o nome de alunos aprovados

```
para aluno em listaDeAlunos,
        se aluno.médiaFinal >= 5.0,
                escreva( aluno.nome + "está aprovado.")
        senão, se aluno.médiaFinal >= 3.0
                        escreva( aluno.nome + "está de recuperação")
                senão
                        escreva( aluno.nome + "está reprovado")
                fim se
        fim se
        escreva(_quebra_de_linha)
fim para
```

Outro exemplo: menu de opções

```
faça
        escreva("1 - listar produtos")
        escreva("2 - cadastrar produtos")
        escreva("3 - editar produtos")
        escreva("4 - deletar produtos")
        escreva("0 - sair")
        opção = leia( )
        interruptor(opção)
                caso 1
                        listarProdutos( )
                caso 2
                        cadastrarProdutos( )
```

```
            caso 3
                editarProdutos( )
            caso 4
                deletarProdutos( )
            caso 0
                escreva("Você escolheu sair. Até breve!")
            padrão
                escreva("Opção inválida.")
        fim interruptor
enquanto(opção != 0)
```

Conteúdo

Estruturas de desvios

Há também a possibilidade de desviar o fluxo de execução de um programa, usando *tentar-pegar* (try-catch), *retornar* (return), *parar* (break) e *continuar* (continue).

- Tentar-Pegar: é uma estrutura parecida com Se-Então-Senão, mas para tratamento de erros.
  tentar
  (sequência)
  pegar (erro)
  (sequência para tratar o erro)
  fim tentar
  Exemplo: Tentar abrir um arquivo. Se o arquivo não existir ocorrerá um erro. Algo que pode ser feito é dizer ao usuário que o arquivo não existe.

- Retornar: é usado dentro de funções. Diz ao programa que não há interesse de continuar a execução, mas sim, de retornar à linha na qual a função foi chamada (usando um se-senão?). Existe a possibilidade de carregar consigo alguma informação para o código que a chamou. Em algumas linguagens, esta estrutura é implícita ao final da função, não precisando ser declarada.

```
função cadastro(parâmetro1, parâmetro2)
        (sequência)
        se (expressão lógica)
                retorne 5
        senão, se (expressão lógica)
                        retorne -5
                senão
                        (sequência)
                fim se
        fim se
        retorne 0
fim função
código = cadastro("João", "Maria")
```

- Parar: é usado dentro de repetições. Diz ao programa que não há interesse de continuar as repetições (usando um se-senão?). O fluxo de execução é enviado ao final do laço.

```
enquanto (expressão lógica)
        (sequência)
        se (expressão lógica)
                pare »
        fim se
        (sequência)
fim enquanto»
(sequência)
```

- Continuar: também é usado dentro de repetições. Diz ao programa que não há interesse de continuar aquela repetição em específico (usando um se-senão?). O fluxo de execução é enviado ao começo do laço, pronto para a iteração seguinte.

```
» enquanto (expressão lógica)
        (sequência)
        se (expressão lógica)
                continue »
        fim se
        (sequência)
fim enquanto
(sequência)
```

Conteúdo

Referências


MARTIN, J. Técnicas estruturadas e CASE. Obra traduzida. São Paulo. Makron Books, 1991.


Linguagem Python `https://docs.python.org/pt-br`


Linguagem C/C++ `http://www.cplusplus.com`

♪ . . . . 𝄞 . . . . **F I M** . . . . 𝄢 . . . . ♪