Programação estruturada

Origem: Wikipédia, a enciclopédia livre.

A **programação estruturada** (PE) é um paradigma de programação, uma forma de programação de computadores, com ênfase no uso de subrotinas, laços de repetição, condicionais e estruturas em bloco.[1] Este paradigma surgiu ao final de 1950 junto às linguagens ALGOL 58 e ALGOL 60,[2] foi impulsionado pelas vantagens práticas que o paradigma oferece, e também pelo 'teorema do programa estruturado (en)' (de 1966, também chamado de teorema de Böhm-Jacopini) e a carta aberta de Dijkstra 'Go To Statement Considered Harmful' (de 1968). De fato, muitas linguagens não possuem GOTOs para desincentivar a programação não-estruturada (nota: Donald Knuth advocou o GOTO em algumas circunstâncias[3]; mesmo depois do estabelecimento da programação estruturada, parece que ainda não concorda com a abolição do GOTO, mas falta referência na Wikipédia em inglês).

A PE foi o paradigma dominante na escrita de software até a programação orientada a objetos (POO). Enquanto a PE fia-se em estruturas de controle de alto nível (em oposição ao uso de GOTOs), concepções top-down e refinamento por passos, a POO se baseia no conceito de objetos que possuem atributos (dados) e métodos (procedimentos). Apesar de ter sido sucedida pela POO, a PE ainda é muito influente pois grande parte das pessoas ainda aprende programação através dela. Para a resolução de problemas simples e diretos, a programação estruturada é bastante eficiente (talvez mais eficiente que a POO). Além disso, por exigir formas de pensar relativamente complexas, a POO até hoje ainda não é bem compreendida ou usada pela maioria.

Diversas linguagens relevantes hoje (e.g. Cobol, PHP, Perl e Go) ainda utilizam o paradigma estruturado, embora possuam suporte para a orientação ao objeto e para outros paradigmas de programação.

Índice

Elementos básicos da teoria

Estruturas de controle

Na PE, os programas são vistos como compostos das seguintes 'estruturas de controle' (ECs) [4]:

- Sequência: de instruções ou sub-rotinas executadas em sequência (`a=4; b=4*5`)
- Seleção/condicional: instruções são executadas ou não conforme o estado do programa (if, else, elif/elseif, endif)
- iteração/repetição: instruções são executados até que o programa atinja um determinado estado (for, while, repeat, do..until)
- recursão: instruções executadas com chamadas auto-referenciadas até que certas condições sejam satisfeitas. Exemplo:

```
def fatorial(x):
    if x > 1:
        return x*fatorial(x-1)
    return x
```

Há a utilização de 'sub-rotinas' em que as ECs são agrupadas e utilizadas através de um única instrução (são as funções, métodos, subprogramas, procedimentos). Blocos permitem que uma sequência de instruções seja tratada como uma única instrução.

Refinamento por passos

Uma ideia central na PE é o refinamento por passos, em que o programa é desenvolvido de maneira top-down, por exemplo:

1. comece com o programa principal
    - use as ECs de iteração e seleção
    - escreva as chamadas para as rotinas (r1, r2, etc) quando necessário. Diz-se 'postule r1, r2'.
1. Implemente r1, r2, ..., com chamadas para outras rotinas conforme conveniente.
2. Continue implementando as rotinas até que não sejam necessários procedimentos adicionais.

Na prática, é usual iniciar a programação não exatamente do topo, até porque é comum que haja vários topos[5], mas isso depende da complexidade e modularidade do software.

Desvios

Dentre os desvios mais comuns da programação estruturada, há múltiplos pontos:

- De saída:
    - terminação antecipada: return em uma função, break or continue em um laço de interação, ou um exit em algum programa. Na programação estruturada, a rigor, há um só ponto de saída da rotina sendo executada.
    - Manejo de exceção: clausulas como (try.. except) do Python ou (try.. catch) do C++, também implicam em múltiplos pontos de saída da rotina.
- De entrada: útil e.g. para geradores, streaming, máquinas de estado.

Conceito-chave: GOTO

Seja um programa uma sequência de instruções a serem seguidas (e.g. por um computador). Considere um ponteiro que indica a instrução a ser executada na próxima oportunidade. Um GOTO é um reposicionamento arbitrário deste ponteiro. Embora seja um comando poderoso, o uso de GOTOs é considerado, em geral, má prática, havendo quem o defenda em algumas situações.[3]

Na programação imperativa, que possui ênfase na modificação de valores em endereços de memória (i.e. instruções de atribuição), o uso de GOTOs é abundante. Em muitos contextos, pode-se assumir que 'programação estruturada' é sinônimo de programação sem GOTO (sem pulos, sem redirecionamentos arbitrários do ponteiro da sequência de instruções em execução). Estes foram os dois primeiros paradigmas dominantes na programação de computadores. A imperativa desde o início da programação até os anos 1970. A estruturada até o final década de 1990, e então deu lugar à POO.

Críticas usuais à PE

Dentre as críticas à PE, constam[4]:

- PE é orientada para a resolução de um problema em particular.
    - Um escopo mais amplo é muitas vezes conveniente.
- PE é realizada pela decomposição gradual da funcionalidade.
    - As estruturas advindas de funcionalidade/ação/controle não são as partes mais estáveis de um programa.
    - Foco em estruturas de dados ao invés de estruturas de controle é uma alternativa.
- Sistemas reais não possuem um único topo.[5]
    - Pode ser apropriado considerar alternativas à abordagem top-down.

Veja também a POO, paradigma que foi estabelecido depois de décadas de PE.

PE vs POO

A PE não é errada nem a POO certa. A POO tende a dar melhores resultados em programas maiores e para reutilização dos programas (suas partes ou sub-rotinas). Como explicitado ao longo deste artigo e do artigo sobre POO, ambos os paradigmas possuem vantagens e desvantagens. A melhor prática parece ser evitar extremismo (i.e. moldes rígidos): há casos em que é melhor priorizar a POO ou a PE, e mesmo quando uma estratégia é evidentemente melhor, o purismo tende a gerar software menos bem escrito ao custo de mais trabalho.

Tópicos avançados

- Diagrama Nassi-Shneiderman: uma representação gráfica (structograma) para PE, desenvolvida em 1972.
- Carta de estrutura (structure chart): um diagrama usado na PE para organizar os módulos em árvore.

Referências

1. «Programação estruturada» (http://www.dca.fee.unicamp.br/cursos/EA876/apostila/HTML/node7.html). Faculdade de Engenharia Elétrica e de Computação - UNICAMP. Consultado em 22 de novembro de 2016
2. Clark, Leslie B. Wilson, Robert G.; Robert, Clark (2000). *Comparative programming languages* (https://books.google.co.uk/books?id=bVSjoO8f2fMC&lpg=PR11&ots=zItCa2GxG6&dq=Comparative%20Programming%20Languages&lr&pg=PA20#v=onepage&q=Comparative%20Programming%20Languages&f=false) 3rd ed. Harlow, England: Addison-Wesley. p. 20. ISBN 9780201710120. Consultado em 25 de novembro de 2015
3. Knuth, Donald E. (1 de dezembro de 1974). «Structured Programming with go to Statements» (http://dl.acm.org/citation.cfm?id=356635.356640). *ACM Computing Surveys (CSUR)*. **6** (4): 261–301. ISSN 0360-0300 (https://www.worldcat.org/issn/0360-0300). doi:10.1145/356635.356640 (https://dx.doi.org/10.1145%2F356635.356640)
4. http://people.cs.aau.dk/~normark/oop-csharp/html/notes/theme-index.html
5. Bertrand Meyer (2009). *Touch of Class: Learning to Program Well with Objects and Contracts*. [S.l.]: Springer Science & Business Media. ISBN 978-3-540-92144-8