Linguagens de Programação

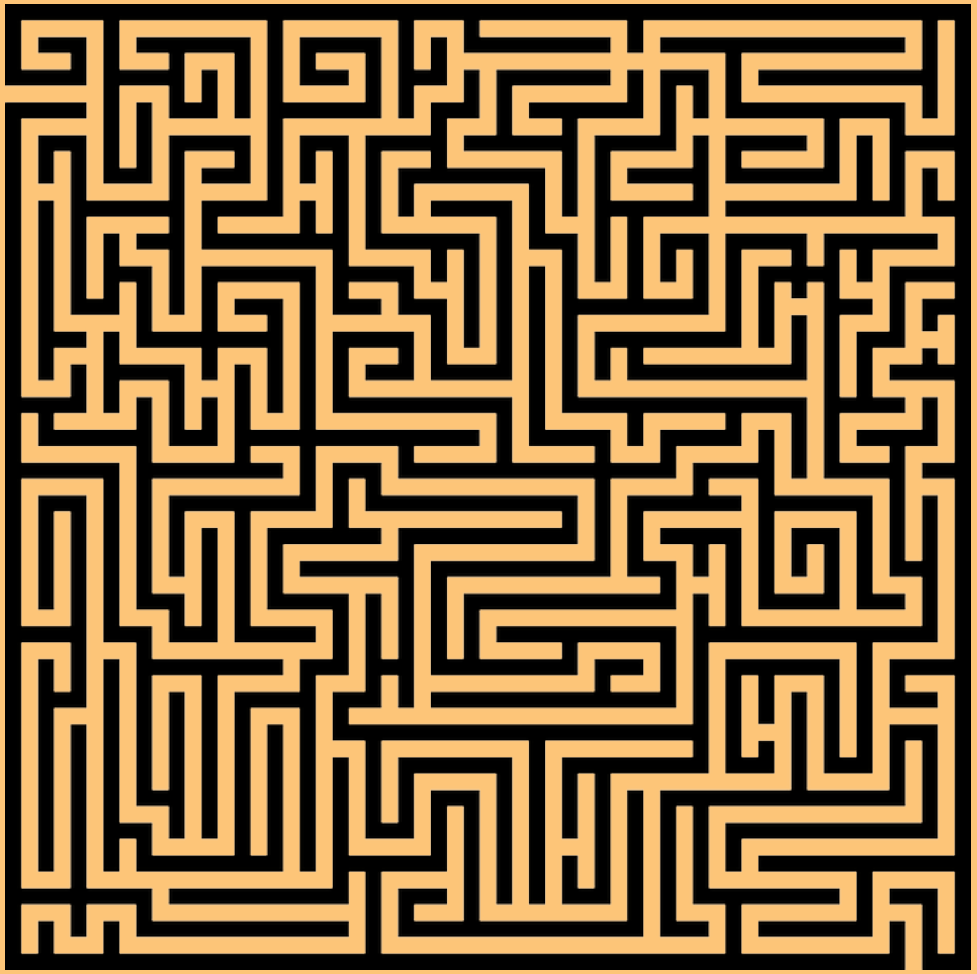
Fluxo de Controle

Carlos Bazilio
carlosbazilio@id.uff.br
http://www.ic.uff.br/~bazilio/cursos/lp

Fluxo de Controle

- Mecanismos utilizados para especificar ordem de execução de instruções recaem em 8 principais categorias:
  - Sequenciamento: usualmente, a ordem no código
  - Seleção: instruções condicionais
  - Iteração: repetição de um conjunto de instruções
  - Abstração procedural: procedimentos e funções
  - Recursão: rotinas que se auto-referenciam
  - Concorrência: 2 ou mais fragmentos de um programa que executam ao mesmo tempo

Problema do Labirinto



```
função timer()
      enquanto tempo > 0
            tempo--

rotina controla_jogador()
      tecla = le_teclado()
      enquanto tempo > 0
            caso tecla
                  A: esquerda()
                  D: direita()
                  W: acima()
                  S: abaixo()
            tecla = le_teclado()

rotina jogo()
      tempo = 60
      timer()
      controla_jogador()

timer() ???
```

Fluxo de Controle

- … (mais categorias):
  - Tratamento de exceção: recurso pelo qual um trecho de código pode ser controlado para, no caso de uma falha, um tratador adequado possa ser acionado
  - Não-determinismo: a ordem de execução das instruções é indeterminada
- Cada linguagem disponibiliza construções que implementam 1 ou + destas categorias
- A importância de cada categoria pode variar em função do paradigma da linguagem adotada

Para refletir ...

"Os **loops** podem melhorar o **desempenho do seu programa**. A **recursão** melhora o **desempenho do seu programador**. Escolha o que for mais importante para a sua situação."

Leigh Caldwell

Goto

- Instrução mais básica de desvio
- Debatida profundamente no passado por não conter uma construção estruturante
- Disponível em linguagens como o C e Pascal
- Tem como “desvantagem” principal o fato de permitir desvios entre instruções de escopos/aninhamentos distintos, pois:
  - Dificulta legibilidade
  - Aumenta esforço do compilador para tornar a execução consistente

Goto
Exemplo em Pascal

```
function busca(chave: string) : string;
var s: string; …
    procedure busca_arquivo(fnome: string);
    begin …
        for ... (* itera sobre o conjunto de linhas *)
            …
            if achou(chave, linha) then
                begin
                    s := linha;
                    goto 100;
                end;
            …
    end;
begin (* busca *) …
    for ... (* itera sobre o conjunto de arquivos *)
        …
        busca_arquivo(fnome);
        …
100: return s;
end;
```

Goto

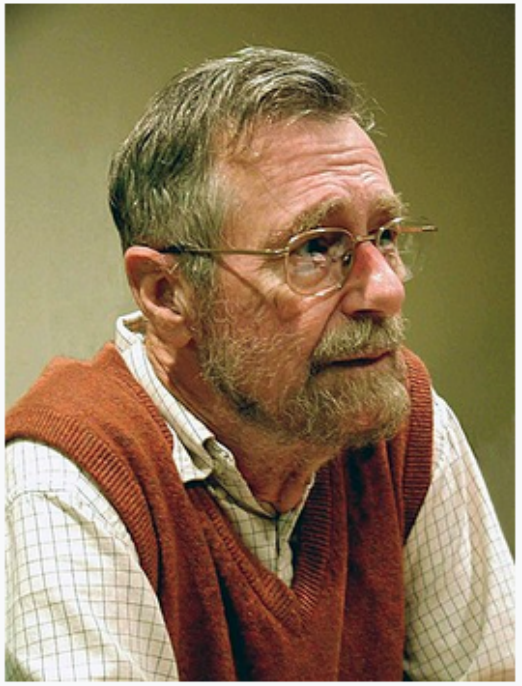
Edsger W. Dijkstra

From Wikipedia, the free encyclopedia

Edsger Wybe Dijkstra (/ˈdaɪkstrə/; Dutch: [ˈɛtsxər ˈʊibə ˈdɛikstra] (listen); 11 May 1930 – 6 August 2002) was a Dutch computer scientist, programmer, software engineer, systems scientist, science essayist,[9][10] and pioneer in computing science.[11] A theoretical physicist by training, he worked as a programmer at the Mathematisch Centrum (Amsterdam) from 1952 to 1962. A university professor for much of his life, Dijkstra held the Schlumberger Centennial Chair in Computer Sciences at the University of Texas at Austin from 1984 until his retirement in 1999. He was a professor of mathematics at the Eindhoven University of Technology (1962–1984) and a research fellow at the Burroughs Corporation (1973–1984). In 1972, he became the first non-American, non-British, and continental European winner of the Turing Award.

One of the most influential figures of computing science's founding generation,[2][3][5][6][12][13] Dijkstra helped shape the new discipline from both an engineering and a theoretical perspective.[14][15] His fundamental contributions cover diverse areas of computing science, including compiler construction, operating systems, distributed systems, sequential and concurrent programming, programming paradigm and methodology, programming language research, program design, program development, program verification, software engineering principles, graph algorithms, and philosophical foundations of computer programming and computer science. Many of his papers are the source of new research areas. Several concepts and problems that are now standard in computer science were first identified by Dijkstra and/or bear names coined by him.[16][17] As a foremost opponent of the mechanizing view of computing

Edsger W. Dijkstra



Dijkstra in 2002

Born	11 May 1930 Rotterdam, Netherlands
Died	6 August 2002 (aged 72) Nuenen, Netherlands
Citizenship	Netherlands
Education	Leiden University (B.S., M.S.)

Seleção

- A construção base para instruções deste tipo é o if-then-else
- Algumas variações:
  - *elseif* <*condição*> *:* permite que os ramos else sejam conjugados com outra condição
  - *case/switch :* alternativa ao aninhamento de várias instruções de seleção

Iteração

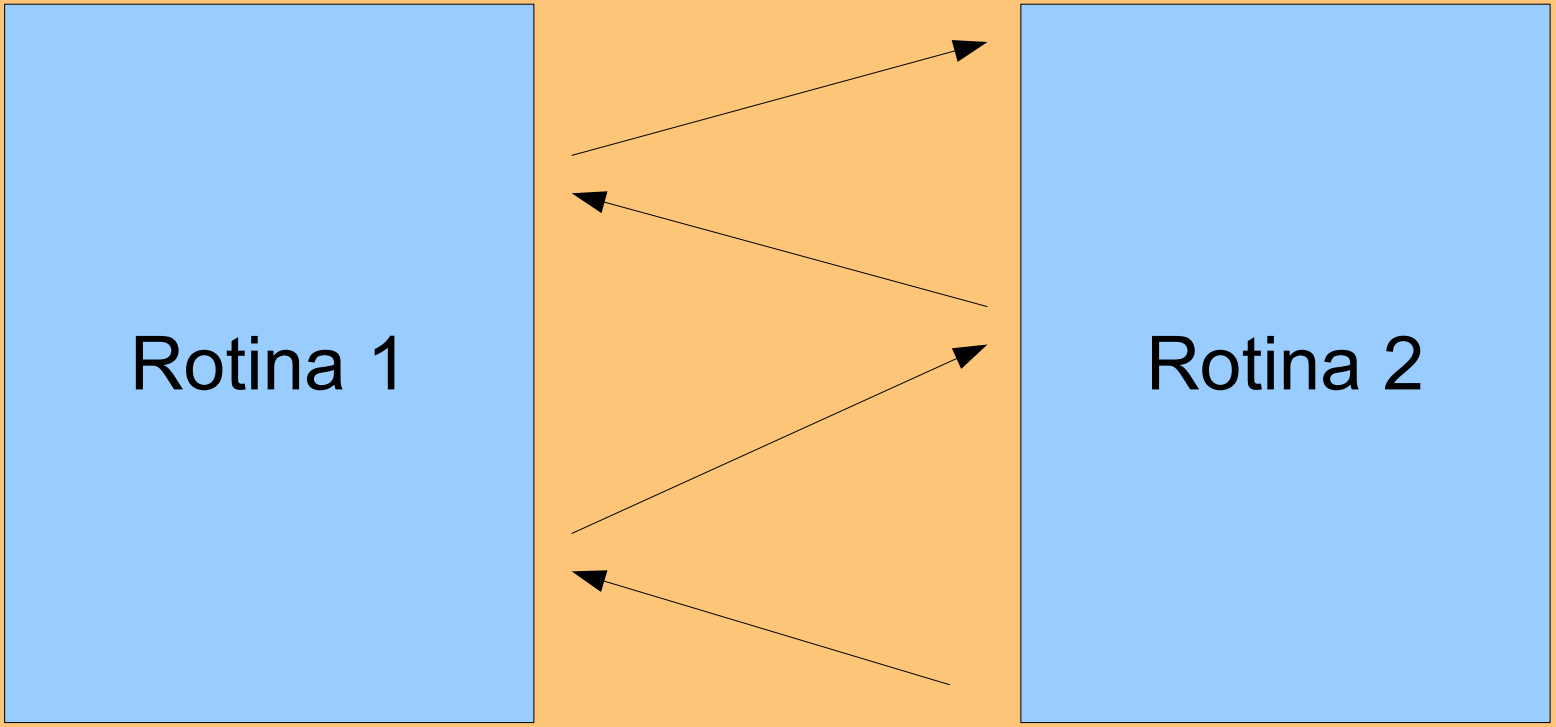
- Recurso para se repetir a execução de um conjunto de instruções
- As construções básicas são o *for* (repetição definida), *while* (repetição indefinida) e a chamada recursiva de funções
- Existem inúmeras variações destas construções; o *for* de C é um bom exemplo:
  - for (i=0; i<=10; i++) {}
  - for (i=0,j=0; (i+j)<=100; i+=2, j++) {}
  - for (;;) {}

Iteração

- Iterações também são utilizadas em enumerações (faixas de valores)
  - for i:=0 to 10 by 2 do …
  - for c:='a' to 'z' do …
- Iteradores são construções oferecidas por diversas linguagens que permitem uma iteração mais genérica
  - cidades = [“petropolis”, “riodasostras”, “macae”, “niteroi”]
  - for c in cidades do ...

Co-rotina

- Construção que permite a cooperação entre 2 rotinas para a execução de uma tarefa



Co-rotina

- A Rotina 1 poderia representar o for, enquanto que a Rotina 2 representa o iterador
- O iterador retorna os elementos, os quais podem estar armazenados de forma distinta:
  - Um vetor de strings
  - Um arquivo
  - Uma lista encadeada
- Ou seja, o iterador faz com que o programa abstraia da forma como um conjunto de valores é armazenado e percorrido

Recursão

- Funções recursivas se caracterizam por funções que se auto-referenciam

```
(define fatorial
      (lambda (n)
            (if (= n 0)
                  1
                  (* n (fatorial (- n 1))))))
```

- Usualmente, as implementações recursivas tendem a ser mais custosas que as iterativas
- Este custo pode ser amenizado com a implementação de recursões “em cauda”

Recursão

"Os **loops** podem melhorar o **desempenho do seu programa**. A **recursão** melhora o **desempenho do seu programador**. Escolha o que for mais importante para a sua situação."

Leigh Caldwell

Recursão em Cauda

- Este tipo de recursão se caracteriza pela computação dos valores ao longo da chamada da função, e não do retorno

```
(define fatorial
      (lambda (n)
            (fatorial_aux n 1)))
(define fatorial_aux
      (lambda (n acum)
            (if (= n 0)
                  acum
                  (fatorial_aux (- n 1) (* acum n)))))
```

- Desta maneira, ao final das chamadas recursivas, o valor a ser computado já está produzido, diferente da versão anterior

Tratamento de Exceções

- Qual é o retorno da função fopen() em C?
- Quantas situações este retorno pode representar?
- Quais situações podem acontecer na tentativa de abertura de um arquivo?
- Tratamento de exceções é um mecanismo que procura capturar estas situações de uma forma estruturada