



UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Centro Tecnológico
Departamento de Informática

Prof. Veruska Zamborlini

veruska.zamborlini@inf.ufes.br

<http://www.inf.ufes.br/~veruska.zamborlini>

Aula 8

Estruturas de controle no nível da sentença

2021/2



Esta obra está licenciada com uma licença Creative Commons Atribuição-
Compartilha Igual 4.0 Internacional: <http://creativecommons.org/licenses/by-sa/4.0/>.

Material adaptado
Prof.s Vitor Souza e Eduardo Zambon

Introdução

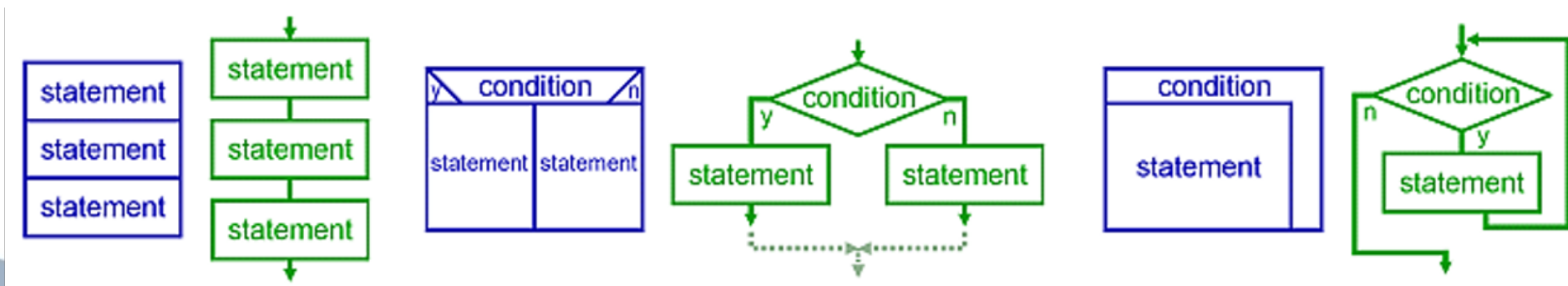
- A essência das linguagens imperativas são as ***atribuições de valores*** realizadas pela ***avaliação de expressões***.
- Tais operações via de regra são utilizadas em conjunto com ***estruturas de controle***:
 - Caminhos alternativos
 - Repetição

Estrutura de Controle

- É a **sentença de controle + coleção de sentenças/expressões** cuja execução ela controla
- Exemplo:
if expressão_de_controle
 then expressão
 else expressão
- Questão de projeto: deve haver múltiplas entradas? (go to)

Teorema de programação estruturada

- Böhm–Jacopini theorem (1966)
- Uma LP de alto nível precisa somente de 3 estruturas:
 1. Execução sequencial
 2. Seleção sobre uma expressão Booleana (if-then-else)
 3. Iteração sobre uma expressão Booleana (while loop)



No princípio tudo era ***go to***

Go to & Jump

- Implementado com instruções assembly de saltos

Instruction	Example	Meaning	Comments
jump	j 1000	go to address 1000	Jump to target address
jump register	jr \$1	go to address stored in \$1	For switch, procedure return
jump and link	jal 1000	\$ra=PC+4; go to address 1000	Use when making procedure call. This saves the return address in \$ra
branch on equal	beq \$1, \$2, 100	if(\$1==\$2) go to PC+4+100	Test if registers are equal
branch on not equal	bne \$1, \$2, 100	if(\$1!=\$2) go to PC+4+100	Test if registers are not equal

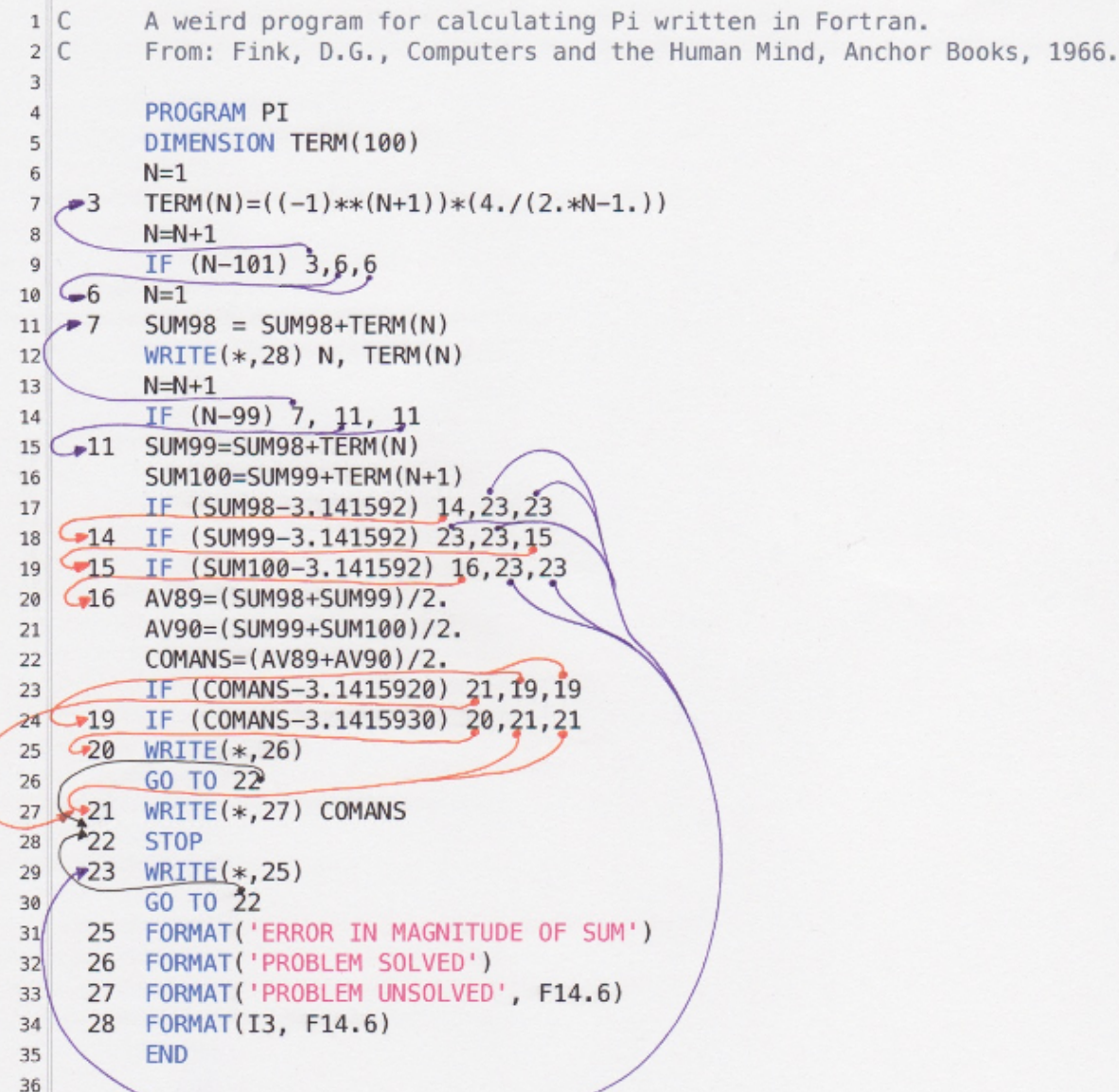
Código Spagetti

■ Exemplo em Fortran

```

1 C      A weird program for calculating Pi written in Fortran.
2 C      From: Fink, D.G., Computers and the Human Mind, Anchor Books, 1966.
3
4      PROGRAM PI
5      DIMENSION TERM(100)
6      N=1
7      TERM(N)=((-1)**(N+1))*(4./(2.*N-1.))
8      N=N+1
9      IF (N-101) 3,6,6
10     N=1
11     SUM98 = SUM98+TERM(N)
12     WRITE(*,28) N, TERM(N)
13     N=N+1
14     IF (N-99) 7, 11, 11
15     SUM99=SUM98+TERM(N)
16     SUM100=SUM99+TERM(N+1)
17     IF (SUM98-3.141592) 14,23,23
18     IF (SUM99-3.141592) 23,23,15
19     IF (SUM100-3.141592) 16,23,23
20     AV89=(SUM98+SUM99)/2.
21     AV90=(SUM99+SUM100)/2.
22     COMANS=(AV89+AV90)/2.
23     IF (COMANS-3.1415920) 21,19,19
24     IF (COMANS-3.1415930) 20,21,21
25     WRITE(*,26)
26     GO TO 22
27     WRITE(*,27) COMANS
28     STOP
29     WRITE(*,25)
30     GO TO 22
31     25 FORMAT('ERROR IN MAGNITUDE OF SUM')
32     26 FORMAT('PROBLEM SOLVED')
33     27 FORMAT('PROBLEM UNSOLVED', F14.6)
34     28 FORMAT(I3, F14.6)
35     END
36

```



Go to em C

```

if (num % 2 == 0)
    // jump to even
    goto even;
else
    // jump to odd
    goto odd;

even:
    printf("%d is even", num);
    // return if even
    return;
odd:
    printf("%d is odd", num);

```


Go to or not ***go to***?

Dijkstra: Not *go to*

- Dijkstra, 1968 - Go To Statement Considered Harmful
- “Eu estou convencido de que o comando *go to* deveria ser abolido de todas as linguagens de programação de alto nível”
- *“Since a number of years I am familiar with the observation that the quality of programmers is a decreasing function of the density of go to statements in the programs they produce. Later I discovered why the use of the go to statement has such disastrous effects and did I become convinced that the go to statement should be abolished from all higher level programming languages.”*

Knuth: *go to*

- Knuth, 1974, Structured Programming with go to Statements
- “...o jeito mais ‘gracioso’ de fazer isso [sair de vários níveis de controle] é com a abordagem direta do *go to* ou de seus equivalentes. “
- *"Sometimes it is necessary to exit from several levels of control, cutting across code that may even have been written by other programmers; **and the most graceful way to do this is a direct approach with a go to or its equivalent.**"*

Go to and not go to

- Knuth, 1974, Structured Programming with go to statements
- I believe that by presenting such a view *I am not in fact disagreeing sharply with Dijkstra's ideas*, since he recently wrote the following: 'Please *don't fall into the trap of believing that I am terribly dogmatical about [the go to statement]*. I have the uncomfortable feeling that others are making a religion out of it, as if the conceptual problems of programming could be solved by a single trick'.

Go to and not go to

- **C/C++/C#** – *permite goto*: deixa o programador se virar
 - C# – versão restrita de goto em comandos switch
- **Java** – *proíbe goto* mas *permite break/continue* com labels (*goto restrito*)

Sentenças de Seleção

Seleção de dois caminhos (if-then-else)

- Questões de projeto:
 - Qual é a forma e o tipo da expressão que controla a seleção?
 - Como são especificadas as cláusulas então e senão?
 - Como o significado dos eleitores aninhados deve ser especificado?

Seleção de dois caminhos (if-then-else)

- Forma e tipo (booleano e/ou numérico)

```
if expressao_de_controle:
    expressao
else
    expressao
```

```
if (expressao de controle)
    expressao;
else
    expressao;
```

```
if expressao_de_controle then
    expressao
else
    expressao
end
```

- Ambiguidade: problema do else pendente

```
if expressao_de_controle:
    if expressao_de_controle:
        expressao
else
    expressao
```

```
if (expressao de controle) {
    if (expressao de controle)
        expressao;
}
else
    expressao;
```

```
if expressao_de_controle then
    if expressao_de_controle then
        expressao
    end
else
    expressao
end
```

Seleção múltipla

- Questões de projeto:
 - Qual é a forma e o tipo da expressão que controla a seleção?
 - Discreto? Ordinal?
 - Como são especificados os segmentos selecionáveis?
 - Como o significado dos seletores aninhados deve ser especificado?

Seleção múltipla

```
switch (expressao) {
    case valor_1_da_expressao: sentenca_1;
    ...
    case valor_n_da_expressao: sentenca_n;
    [default: sentenca_0;]
}
```

```
switch (expressao) {
    case valor_1_da_expressao: sentenca_1; break;
    ...
    case valor_n_da_expressao: sentenca_n; break;
    [default: sentenca_0;]
}
```

```
case
    when expressao_de_controle then expressao
    ...
    when expressao_de_controle then expressao
    else expressao
end
```

```
if expressao_de_controle:
    expressao
elif expressao_de_controle:
    expressao
...
elif expressao_de_controle:
    expressao
else expressao
```

Sentenças de Iteração

Laços controlados por contador

- Questões de projeto:
 - Qual é o tipo e o *escopo* da variável de laço?
 - Discreto? Ordinal?
 - É permitido modificar variável/parâmetros de laço dentro do laço?
 - Os parâmetros devem ser avaliados apenas uma vez ou a cada interação?

```
for (int i=0; i < 10; i++) {
    var = expressao;
}
```

```
for i in range(10):
    var = expressao;
```

Laços controlados logicamente

- Questões de projeto:
 - O controle deve ser pré ou pós-teste?
 - É uma sentença específica ou um caso especial do laço por contagem?

```
while (expressao_de_controle)  
    corpo_do_laco;
```

```
do  
    corpo_do_laco;  
while (expressao_de_controle)
```

Mecanismos posicionados pelo usuário (goto restrito)

- Questões de projeto:
 - O mecanismo deve ser uma parte integral da saída?
 - É possível sair apenas de um corpo de laço ou de todos que eventualmente o envolvam?
- ***break, last***: saídas incondicionais rotuladas (Java e Perl) e não rotuladas (C, C++, Python, Ruby)
- ***continue***: transfere o controle para o mecanismo do menor laço (C, C++, Python) ou para um laço especificado por rótulo (Java, Perl)

Iteração baseada em estrutura de dados

- Algumas linguagens oferecem uma estrutura ou objeto iterador para controlar a iteração: Perl, Java, PHP, C#, Python.

```
String[] strList = {...}  
foreach (String in strList)  
    Console.WriteLine(name);
```

```
for variavel_1 [, variavel_2, ...] in iterador(tabela) do  
    ...  
end
```

Comandos Protegidos

Comandos Protegidos

- Proposto por Dijkstra, 1975. *Guarded commands, non-determinacy and formal derivation of programs*
- Propõe o conjunto mínimo de construtos segundo o teorema de Böhm–Jacopini
- Intenção era criar uma linguagem que permitisse a dedução formal de programas "corretos por construção"
- Muito útil em sistemas concorrentes (CSP, Ada)

*"Program testing can be used to show the presence of bugs,
but never to show their absence!"*

Guarded Command Language (GLC)

- Todas as expressões são avaliadas
 - Guardas podem ter sobreposição
 - Se mais de uma expr. é verdade, uma é escolhida de forma não-determinística
- Se nenhuma expressão é verdade, ocorre um erro

```

if <Boolean expression> -> <statement>
[] <Boolean expression> -> <statement>
[] . . .
[] <Boolean expression> -> <statement>
fi

```

```

do <Boolean expression> -> <statement>
[] <Boolean expression> -> <statement>
[] . . .
[] <Boolean expression> -> <statement>
od

```

Subprogramas (próxima aula)

Subprogramas (exceto co-rotinas)

- Possuem um único ponto de entrada
- Suspendem a execução de quem chama
 - Execução sequencial
- Retornam ao ponto de chamada ao terminar
 - Evita saltos arbitrários no código
 - Logo, evita código macarrônico