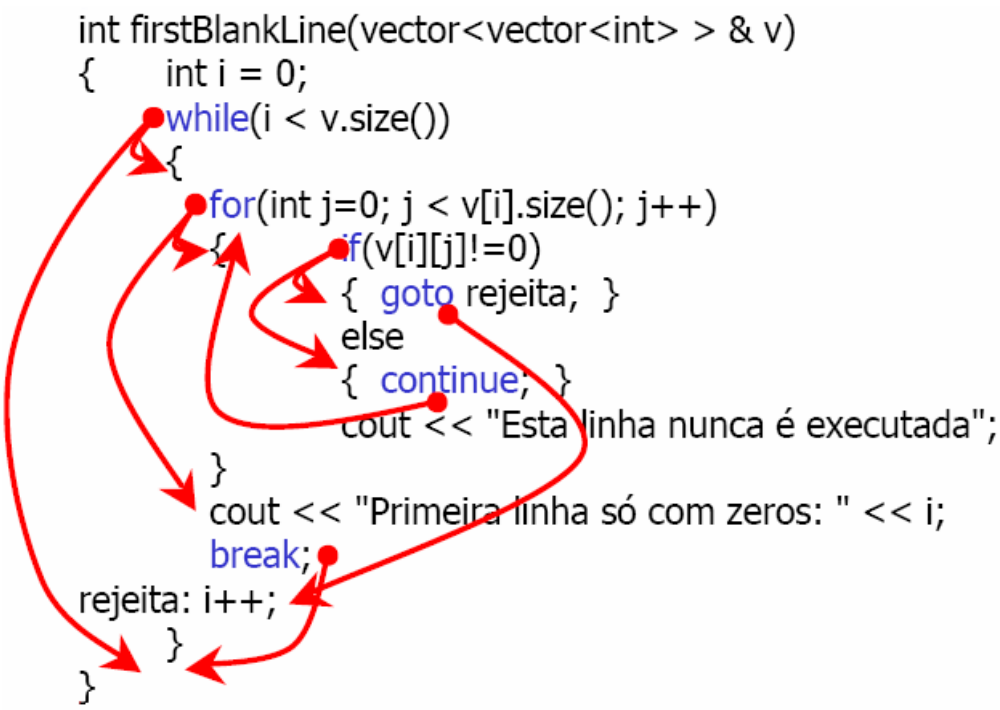
Estruturas de controle no nível da instrução (referência: Cap. 8 do livro de Sebesta)

Introdução



- Fluxo de controle ao nível de instrução - sequência de execução entre as diversas instruções do programa.
- Níveis de fluxo de controle:
  - 1. Entre instruções do programa
  - 2. Dentro de expressões
    - Regras de associatividade
    - Regras de precedência de operadores)
  - 3. Entre unidades de programa (subprogramas)

Instruções de controle: evolução

- Muita pesquisa e discussão acerca desse assunto nos anos 60
- Um resultado importante
  - Foi provado que todos os algoritmos representados por um fluxograma podem ser codificados usando apenas duas instruções de controle:
    - Uma para escolher entre dois caminhos a seguir (if...else)
    - Uma para iterações controladas logicamente (while)

Estruturas de controle

- Instruções de Controle – meio de selecionar um caminho entre diferentes caminhos de execução, podendo por vezes provocar a repetição de conjunto de instruções (ex.: if's, ciclos, goto's, switch's, …).
- Uma estrutura de controle é uma instrução de controle e sua coleção de comandos que controla (que fazem parte da estrutura).
- Problema geral de projeto:
  - Quais as estruturas de controle que devem estar contidas numa L.P. além de seleção e laços lógicos pré-testados?
  - Uma estrutura de controle deve possuir múltiplas entradas?

Instruções compostas

- Instrução Composta – método para abstrair uma coleção de instruções como uma única instrução
- Introduzido no ALGOL 60 na forma de:

```
begin
comando 1
...
comando n
end
```

- Um bloco é uma Instrução Composta que pode definir um novo escopo (possui variáveis locais).

Instruções de seleção

- Uma instrução de seleção oferece os meios de escolher entre dois ou mais caminhos de execução em um programa
- Duas categorias gerais:
  - Seleção bidirecional
  - Seleção n-dimensional ou múltipla
- Forma geral:

```
if control_expression
then clause
else clause
```

- Questões de projeto:
  - Qual é a forma e o tipo da expressão que controla a seleção?
  - Como as cláusulas then e else são especificadas?
  - O que pode ser selecionado? (Instruções simples/compostas, seqüência de comandos)?
  - Como deve ser especificado o aninhamento de instruções de seleção?

Exemplos de seleção

- Exemplo em FORTRAN (unidirecional)
- FORTRAN: IF (expressão booleana) instrução
- Problema
  - pode selecionar apenas uma única instrução
  - para selecionar mais, um GOTO deve ser usado, como no seguinte exemplo (atribui a I e J os valores 1 e 2 se a condição for atendida)

```
IF (.NOT. condition) GOTO 20
I = 1
J = 2
20 CONTINUE
```

- Lógica negativa é prejudicial para a legibilidade
- Esse problema foi solucionado no FORTRAN 77

- Exemplo em ALGOL 60 (unidirecional – somente then):

```
if (boolean_expr) then
begin
...
end
```

- Exemplo em ALGOL 60 (bidirecional – then ...else):

```
if (boolean_expr)
then instrução (Cláusula then)
else instrução (Cláusula else)
```

- "instrução" - um instrução simples ou composta.

Exemplo de aninhamento de seletores

- Exemplo em Pascal (aninhamento direto de seletores):

```
if ... then
  if ... then
    ...
  else ...
```

- Qual then associa-se com else?
- Regra do Pascal: else associa-se com o then mais próximo.

- Exemplo em Java

```
if (sum == 0)
      if (count == 0)
            result = 0;
else result = 1;
```

- O else pertence a qual dos ifs?
- Java possui uma regra semântica estática
  - O else é associado ao if mais próximo

- Para forçar uma semântica alternativa, uma forma sintática diferente é necessária, na qual o if-then é colocado em uma instrução composta:

```
if (sum == 0)
    {
        if (count == 0) result = 0;
    }
else result = 1;
```

- A solução acima é usada em C, C++ e C#
- Perl requer que todas as cláusulas then e else sejam compostas

- Ex. em ALGOL 60:
  - Não permite aninhamento direto (if logo abaixo de outro if)
  - Permite aninhamento indireto (if dentro de blocos begin...end)

```
if ... then
 begin
    if ... then
       ...
    else ...
 end
```

```
if ... then
 begin
   if ... then ...
 end
else
 ...
```

- Ex. em Ada
  - Fechamento de seleção com palavra especial (término)
  - FORTRAN 77, e Modula-2 também contêm términos

```
if ... then
  if ... then
    ...
  else
    ...
  end if
end if
```

```
if ... then
  if ... then
    ...
  end if
else
  ...
end if
```

- Em do Modula-2:
  - Utiliza a mesma palavra especial de término "END" para todas as estruturas de controle.
  - Esta construção resulta em legibilidade pobre.
- Vantagens da seleção com término:
  - Maior Flexibilidade;
  - Maior Legibilidade.

Construções de seleção múltipla

- Permite a seleção de uma instrução dentre qualquer número de instruções ou de grupos de instruções
- Questões de Projeto:
  - 1. Qual é a forma e o tipo da expressão que controla a seleção?
  - 2. Instruções únicas, seqüências de instruções ou instruções compostas podem ser selecionadas?
  - 3. O fluxo de execução através da estrutura limita-se a incluir apenas um único segmento selecionável?
  - 4. Que decisão deve ser tomada no caso do valor da expressão seletora não estiver representado?

- Seletores múltiplos antigos:
  - FORTRAN - IF aritmético
  - IF (expressão aritmética) N1, N2, N3
    - Vai para rótulo N1 se expressão for negativa
    - Vai para rótulo N2 se expressão for igual a zero
    - Vai para rótulo N3 se expressão for maior que zero
  - Segmentos requerem GOTOs
  - Sem encapsulamento, ou seja, os segmentos selecionáveis podem estar em qualquer lugar no código

```
IF (expressão) 10, 20, 30
10 ...
   ...
   GO TO 40
20 ...
   ...
   GO TO 40
30 ...
   ...
   GO TO 40
40
```

- Selectores Múltiplos Modernos
  - Pascal "case";
  - Sintaxe:

```
case expressão of
    constante_1 : instrução_1;
    ...
    constante_n : instrução_n
end
```

  - Vários dialetos de Pascal possuem cláusula otherwise ou cláusula else
  - Considerações de Projeto "case" do Pascal:
    - Expressão é qualquer tipo ordinal (int, boolean, char, enum);
    - Segmentos podem conter instruções simples ou compostas;
    - Construção é encapsulada;
    - Somente um segmento pode ser executado por seleção da construção;

  - C switch

```
switch (expression) {
    case const_expr_1: stmt_1;
    ...
    case const_expr_n: stmt_n;
    [default: stmt_n+1]
}
```

  - Escolhas de projeto para o switch do C
    - 1. A expressão de controle pode ser apenas do tipo inteiro
    - 2. As instruções selecionáveis podem ser seqüências de instruções ou blocos de instruções
    - 3. O controle pode fluir por mais de um segmento selecionável em uma única execução
      - Para evitar este efeito o programador deve usar o comando break no fim de cada segmento;
      - É um compromisso entre confiabilidade e flexibilidade.

- 4. A cláusula default serve para valores que não foram representados (se não existir default, nada é feito)

- Ex. do Ada (case):
  - Seletor é similar ao case de Pascal, exceto que:
    - Lista de constante pode incluir:
      - Subfaixas ex.: 10..15
      - Operador Boleano OR ex.: 1..5 | 7 | 15..20
  - Lista de constante dever ser completa:
    - Normalmente realizado via cláusula others;
    - Isto torna o comando mais confiável.

```
case expression is
    when choice list => stmt_sequence;
    …
    when choice list => stmt_sequence;
    when others => stmt_sequence;]
end case;
```

- Seletores múltiplos usando if (elseif)
  - Existente em ALGOL 68, FORTRAN 77, Modula-2, Ada.
  - Ex. em Ada:

```
if (condição) then
...
elsif (condição) then
...
elsif (condição) then
...
else
...
end if
```

  - Muito mais confiável que aninhamento de if's ;
  - Permite saída de cada grupo selecionado.

Instruções Iterativas

- Uma instrução ou um bloco de instruções é executado zero, uma ou mais vezes através de iteração ou recursão

- Questões de projeto:
  - Como a iteração é controlada?
    - Contador;
    - Expressão lógica (booleana) ou relacional.
  - Onde colocar a condição de controle do laço?
    - No início do laço (laço pré-testado);
    - No fim do laço (laço pós-testado).

Laços controlados por contador

- Uma instrução iterativa de contagem possui uma variável do laço, e meios de especificar os valores inicial e terminal, e o tamanho do passo
- Considerações de projeto:
  - Qual o tipo e o escopo da variável do laço?
  - Qual o valor da variável de controle no final do comando?
  - É válido alterar a variável de controle do laço no corpo do comando? Em caso afirmativo o controle do laço é afetado?
  - Os parâmetros do laço são somente avaliados uma única vez ou a cada iteração?
- Ex. em Pascal:
  - Sintaxe:
    - for var := inicio (to | downto) final do comando
  - Questões de Projeto "for" do Pascal:
    - Variável do laço deve ser do tipo ordinal, de escopo visível (local ou não local);
    - Após terminar de forma normal, o valor da variável do laço é indefinido;
    - A variável do laço não pode ser alterada no corpo do laço;
    - Parâmetros do laço são avaliados uma única vez.

- Instrução for do C, do C++ e do Java

  for ([expr_1] ; [expr_2] ; [expr_3])
  corpo do laço

- Todas as variáveis envolvidas podem ser alteradas no corpo do laço
- A primeira expressão é avaliada apenas uma vez, mas as outras duas são avaliadas em cada iteração
- C++ difere de C de duas maneiras:
  - 1) A expressão de controle pode ser booleana
  - 2) A expressão inicial pode incluir definições de variáveis (escopo é da definição até o fim do corpo do laço)
    - for (int cont = 0; cont < comp; cont++) {…}

Quadro comparativo

Linguagem	Tipo Var. de controle do laço	Teste do laço	Valor da Var. no fim do laço	Var. laço pode ser alterada?
FORTRAN I	int	pós	mais recente	não
FORTRAN 77	int, float, double	pós	mais recente	não
Algol 60	int, float	pré	mais recente	não
Pascal	int, enumerado	pré	mais recente	sim
Ada	int,enumerado	pré	eliminado	não
C/C++/Java	n/ tem var.	pré	mais recente	sim

Laços controlados logicamente

- controle da repetição baseia-se em uma expressão booleana e não em um contador
- Questões de projeto:
  - O controle deve ser pré-teste ou pós-teste?
  - O laço controlado logicamente deve ser uma forma especial de laço de contagem ou uma instrução separada?
- Formas gerais:

```
while (ctrl_expr)
loop body
```

```
do
  loop body
while (ctrl_expr)
```

- Pascal possui instruções separadas para laços de pré-teste e de pós-teste
  - while-do e repeat-until

```
while(condição=true) do    ❶
begin
     ...
end
```

```
repeat  ❷                  do  ❸
     ...                        ...
until(condição=false)      while(condição=true)
```

- C e C++ também possuem, mas a expressão de controle para o pós-teste é tratada como no pré-teste

```
❶while(condição==true) corpo do laço;

❸do
   corpo do laço;
 while(condição==true);
```

- Java é similar a C, exceto porque a expressão de controle deve ser booleana

- Ada possui uma versão com pré-teste, mas nenhuma com pós-teste
- FORTRAN 77 e 90 não possui nenhuma das duas
- Perl possui dois laços lógicos de pré-teste, while e until, mas nenhum de pós-teste

Mecanismo de Controle de Laços Localizados pelo Usuário

- Em algumas situações é conveniente para o programador escolher a localização do controle do laço (outra que não o topo ou a base)
  - O programador deve poder escolher onde colocar o controle do laço, como por exemplo no meio do corpo do laço.
- Solução simples para alguns laços (e.g., break)

- Questões de projeto para laços aninhados:
  - 1. Deve-se permitir que o mecanismo apareça em um laço controlado ou somente em um laço sem qualquer controle?
  - 2. Somente um corpo do laço deve ser finalizado, ou laços envolventes também podem ser finalizados?

- Exemplo em Ada
  - Término condicional ou incondicional, para qualquer laço e para qualquer quantidade de níveis.

Término condicional	Término incondicional

`for ... loop`
`  ...`
`  exit when soma > 1;`
`  ...`
`end loop`
	`for ... loop`
	`  ...`
	`  if soma > 1 then exit;`
	`  ...`
	`end loop`

- Exemplo em C, C++ e Java (break):
  - Término incondicional não rotulado, para qualquer laço ou "switch", em apenas um nível.
  - Em C/C++ existe também a instrução "continue" que termina as instruções e volta para o controle do laço.
    - Ex. da instrução "continue":

```
while(soma < 1000)
{ getnext(valor);
   if (valor < 0 ) continue;
    soma += valor
}
```

    - Nota: Se valor for negativo a instrução de atribuição não é executada.
  - Exemplo da instrução "break":

```
while ( soma < 1000)
{ getnext(valor);
   if (valor < 0 ) break;
    soma += valor
}
```

    - Nota: Neste caso se valor for negativo o laço é finalizado.

- Exemplo do FORTRAN 90 (EXIT):
  - Término incondicional para qualquer laço, e qualquer número de níveis.
  - FORTRAN 90 possui a instrução "CYCLE" (que tem a mesma semântica do "continue" do C).

Iteração baseada em Estruturas de Dados

- A iteração está associada a uma estrutura de dados;
- Controle do laço é dado pela existência ou não de mais elementos na estrutura de dados;
- O mecanismo de controle é uma chamada de função que retorna o próximo elemento, em alguma ordem, desde que exista elementos, caso contrário o laço termina.

- O for do C, do C++ e do Java pode ser usado para simular uma instrução de Instruções iterativas:

```
for (p=root; p==NULL; traverse(p)){
...
}
```

- Ex. em C++ (mostrar todos os elementos de um array):
  - A instrução "for" do C++ pode ser utilizada para percorrer toda a estrutura de dados.

```
char * nomes[]={"José","João","Joca",0};
for (char ** p = nomes; *p!= 0; p++)
{ cout << *p << endl; }
```

- Ex. em Perl (mostrar todos os elementos de um array):
  - Perl possui um iterador implícito para arrays e hashes.

```
$nomes = {"José","João","Joca"};
foreach $nome (@nomes)
{ print $nome }
```

- A instrução foreach do C# itera nos elementos de vetores:

```
Strings[] = strList = {“Bob”, “Carol”, “Ted”};
foreach (Strings name in strList)
Console.WriteLine (“Name: {0}”, name);
```

Desvio incondicional

- Uma instrução de desvio incondicional transfere o controle para outra posição do programa.
- Exemplo mais conhecido: GOTO

- A instrução de desvio incondicional (goto) é a instrução mais poderosa para controle do fluxo de execução (esta nasceu no FORTRAN).
- Problema:
  - n Baixa Legibilidade → Baixa Confiabilidade;
- Algumas linguagens não têm instruções de desvio incondicional, como por ex. Modula-2 e Java.
- Algol, Pascal, C e C++ permitem "goto" mas desaconselham a sua utilização.

Forma dos Rótulos (Label):

- Constantes inteiras sem sinal, seguido de dois-pontos.
  - Utilizado em Pascal. Ex. "100:"
- Constantes inteiras sem sinal (não possui dois pontos).
  - Utilizado em FORTRAN. Ex. "100"
- Identificadores com dois-pontos.
  - Utilizado em ALGOL 60, C e C++. Ex. "FIM:"
- Identificador em << ... >>
  - Utilizado em Ada. Ex. "<<FIM>>"
- Variáveis como rótulo.
  - Utilizado em PL/I.

Restrições aos desvios - Restrições do "goto" em Pascal:

- O alvo de um goto não pode ser uma instrução, num grupo de instruções que não esteja ativo (não iniciado).
- Pela razão anterior, o alvo não pode ser uma instrução de um grupo que esteja no mesmo nível ou em nível mais profundo que o nível do goto.
- O alvo pode ser qualquer subprograma do escopo corrente (de qualquer nível), desde que a instrução não seja uma instrução de grupo.
- O comando goto pode terminar qualquer número de subprogramas (neste caso, um goto para fora desses subprogramas) .

Exemplo de "goto" em C:

```
printf("Enter m for mesg, or e to end:");
scanf("%c",&letter);
if(letter=='m')
goto A;
else
goto B;
A: printf("\nHello!, you pressed m");
goto FIM;
B: printf("\nBye!, ending program");
   FIM:
```

Comandos protegidos

- Propósito
  - Fornecer instruções de controle que suportem uma metodologia de projeto que assegure a exatidão durante o desenvolvimento
- Motivação: suportar uma metodologia de projeto de programação (verificação durante a execução do programa).
- Idéia básica
  - Se a ordem de avaliação não é importante, o programa não deve especificar uma
- Instrução de Seleção:

```
if <boolean> -> <instrução>
...
[] <boolean> -> <instrução>
fi
```

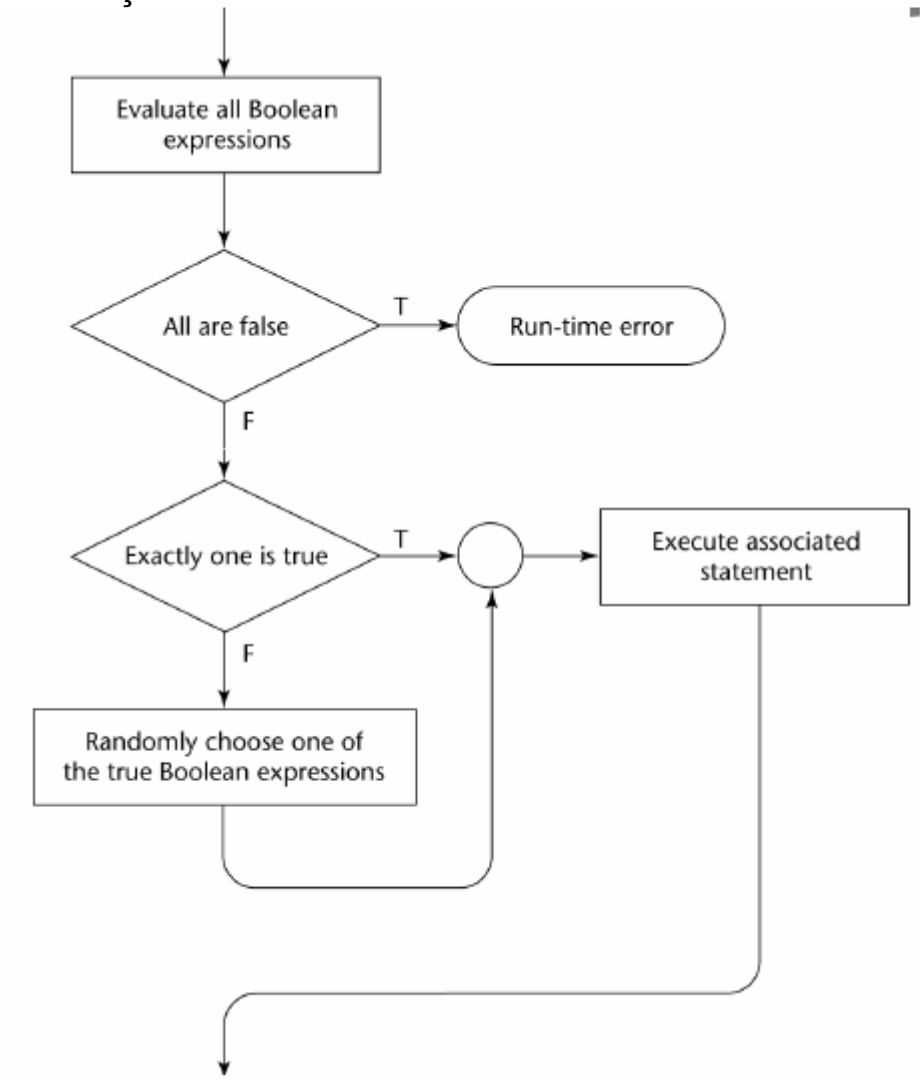
- Semântica: quando a construção é alcançada
  - Avalie todas as expressões booleanas
  - Se mais de uma for verdadeira, escolha uma de forma não determinística
  - Se nenhuma for verdadeira, um erro em tempo de execução é gerado

- Exemplo em Ada:

```
if i =0 -> soma := soma + i      (1)
[] i > j -> soma := soma + j     (2)
[] j > i -> soma := soma + i     (3)
fi
```

- Se i = 0 e j > i, a construção escolhe não deterministicamente a primeira ou a terceira instrução para ser executada;
- Se i for igual a j e diferente de zero, temos um erro pois nenhuma das expressões é válida.

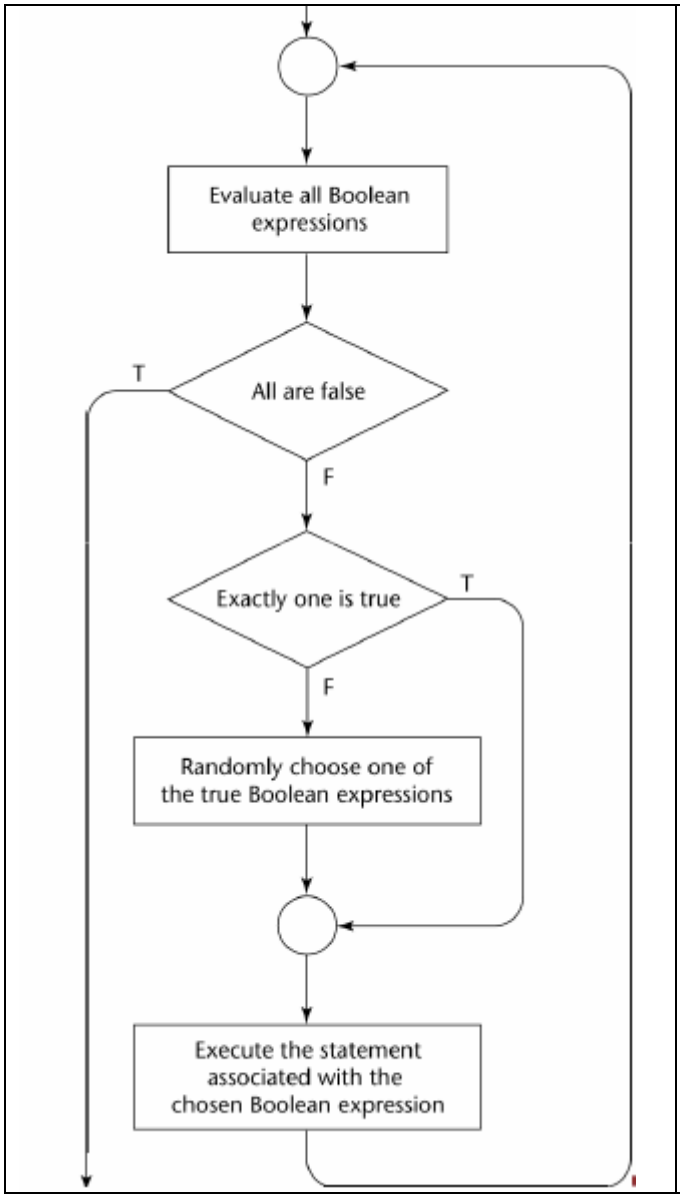
Ilustração



- Instrução de Laço:

  do <boolean> -> <instrução>
  [] <boolean> -> <instrução>
  ...
  [] <boolean> -> <instrução>
  od

- Semântica: para cada iteração
  - Avalie todas as expressões booleanas
  - Se mais de uma for verdadeira, escolha uma de forma não-determinística; e depois inicie o laço novamente
  - Se nenhuma for verdade, saia do laço



- Exemplo em ADA: (ordenar: n1 < n2 < n3)

```
do n1 > n2 -> max:= n1; n1:=n2; n2:=max;
[] n3 > n2 -> max:= n3; n2:=n3; n3:=max;
od
```

Conclusões do Capítulo:

- A escolha das instruções de seleção e laços lógicos de pré-teste é um compromisso entre tamanho da linguagem e escritabilidade.

Ainda não existe um concordância sobre a utilização ou não de instruções de controle incondicional "goto" numa linguagem.