

Estruturas de controle no nível de sentença

Marco A L Barbosa
malbarbo.pro.br

Departamento de Informática
Universidade Estadual de Maringá



Este trabalho está licenciado com uma Licença Creative Commons - Atribuição-CompartilhaIgual 4.0 Internacional.
<http://github.com/malbarbo/na-lp-copl>

Conteúdo

Introdução

Sentenças de seleção

Sentenças de iteração

Desvio incondicional

Comandos guardados

Introdução

- O controle do fluxo em um programa ocorre em diversos níveis
 - Dentro das expressões (capítulo 7)
 - Entre as unidades de programas (capítulo 9)
 - Entre as sentenças (capítulo 8 - o qual vamos estudar agora)

- No passado, as sentenças de controle das linguagens eram feitas baseadas nas instruções dos processadores
- Muita pesquisa e discussão entre meados de 1960 e meados 1970
 - Apenas um sentença de controle é necessária, o goto, mas esta sentença tem diversos problemas
 - O goto pode ser substituído por duas sentenças de controle: seleção e repetição
 - Na prática, o aspecto importante não é a quantidade de sentenças de controle, mas a facilidade de leitura e escrita

- Uma **estrutura de controle** é uma sentença de controle e a coleção de sentenças cuja a execução está sobre seu controle
- Questão de projeto relativa a todas as estruturas de controle
 - A estrutura de controle tem múltiplas entradas?

Sentenças de seleção

Sentenças de seleção

- Uma **sentença de seleção** prove meios de escolher entre dois ou mais caminhos de execução no programa
- As sentenças de seleção são divididas em duas categorias
 - Duas vias
 - Múltiplas vias

Sentenças de seleção de duas vias

- Apesar das diferenças no projeto, as sentenças de seleção de duas vias das linguagens contemporâneas são bastante semelhantes. A forma geral é:

```
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?
 - Como o significado dos seletores aninhados devem ser especificados?

Sentenças de seleção de duas vias

- A expressão de controle
 - A expressão de controle é especificada entre parênteses se a palavra reservada `then` (ou outro marca sintática) não é usada
 - Algumas linguagens permitem expressões aritméticas (C/C++, Python), outras permitem apenas expressões booleanas (Ada, Java, Ruby, C#)

Sentenças de seleção de duas vias

- Forma da cláusulas `then` e `else`
 - Em Perl, todas as cláusulas precisam ser sentenças compostas
 - A maioria das linguagens permitem sentenças simples e sentenças compostas
 - As linguagens baseadas em C utilizam chaves para formar sentenças compostas
 - Em Fortran 95, Ada e Ruby as cláusulas `then` e `else` são sequências de sentenças. A seleção completa acaba com uma palavra reservada
 - Python usa indentação para especificar sentenças compostas

```
if x > y:  
    x = y  
    print "case 1"
```

- Quando uma construção de seleção é aninhada com a cláusula `then` de outra construção de seleção, não fica claro com qual `if` a cláusula `then` deve ser associada

Seletores aninhados

Exemplo

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

Esta construção pode ser interpretada de duas formas

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

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

- Em Perl, as cláusulas `then` e `else` têm que ser compostas
- Em Java, a semântica estática especifica que a cláusula `else` faz par com o a cláusula `then` sem par mais próxima
- Uma alternativa sintática é o uso de uma palavra reservada para marcar o fim do seletor (Fortran 95, Ada, Ruby, Lua)

Seletores aninhados

Exemplo em Ruby

```
if sum == 0 then
  if count == 0 then
    result = 0
  else
    result = 1
  end
end
end
```

```
if sum == 0 then
  if count == 0 then
    result = 0
  end
else
  result = 1
end
end
```

Seletores aninhados

Em linguagens que a indentação é significativa, este problema não existe

Exemplo em Python

```
if sum == 0:
    if count == 0:
        result = 0
    else:
        result = 1
```

```
if sum == 0:
    if count == 0:
        result = 0
else:
    result = 1
```


Sentenças de seleção múltipla

- Uma construção de **seleção múltipla** permite a seleção de uma entre várias sentenças (ou grupo de sentenças)
- Questões de projeto
 - Qual é a forma e o tipo da expressão que controla a seleção?
 - Como os segmentos selecionáveis são especificados?
 - O fluxo de execução através da estrutura é restrito a incluir apenas um segmento selecionável?
 - Como os valores dos casos são especificados?
 - Como os seletores de valores não apresentados deve ser tratados?

Sentenças de seleção múltipla

Exemplo das linguagens baseadas em C

```
switch (index) {  
    case 1:  
    case 3: odd += 1  
           sumodd += index;  
           break;  
    case 2:  
    case 4: even += 1;  
           sumeven += index;  
           break;  
    default: printf("Error in switch, index %d\n", index);  
}
```

- Exemplo das linguagens baseadas em C
 - As expressões podem ser do tipo inteiro, caractere ou enumerado
 - Mais que um segmento pode ser executado por vez
 - Nem todos os casos precisam ser especificados

Sentenças de seleção múltipla

Exemplo em C#

```
switch (value) {  
    case -1:  
        Negatives++;  
        break;  
    case 0:  
        Zeros++;  
        goto case 1;  
    case 1:  
        Positives++;  
        break;  
    default:  
        Console.WriteLine("Error in switch\n");  
        break;  
}
```

Sentenças de seleção múltipla

- Exemplo em C#
 - Não é permitido a execução implícita de mais de um segmento
 - As expressões também podem ser strings

Sentenças de seleção múltipla

Exemplo em Ada

```
case expression is
    when choice list => statement_sequence;
    ...
    when choice list => statement_sequence;
    [when other => statement_sequence;]
end case;
```

Sentenças de seleção múltipla

- Exemplo em Ada
 - A expressão tem que ser do tipo ordinal
 - Apenas um segmento pode ser executado
 - Lista de escolhas
 - Um das formas 7, 10..15, 10 | 15 | 20
 - Os valores precisam ser mutuamente exclusivos
 - Exaustiva

Sentenças de seleção múltipla

Exemplo em Ruby (semelhante a ifs aninhados)

```
leap = case
  when year % 400 == 0 then true
  when year % 100 == 0 then false
  else year % 4 == 0
end
```


Sentenças de seleção múltipla

Exemplo em Ruby (semelhante ao switch)

```
case int_val
  when -1 then neg_count++
  when 0 then zero_count++
  when 1 then pos_count++
  else puts "Error: int_val is out of range"
end
```

Sentenças de seleção múltipla

Exemplo em Python (seleção múltipla usando if)

```
if count < 10:  
    bag1 = True  
else:  
    if count < 100:  
        bag2 = True  
    else:  
        if count < 1000:  
            bag3 = True
```

Sentenças de seleção múltipla

Exemplo em Python (seleção múltipla usando if)

```
if count < 10:  
    bag1 = True  
elif count < 100:  
    bag2 = True  
elif count < 1000:  
    bag3 = True
```

Sentenças de iteração

Sentenças de iteração

- Uma **sentença de iteração** é aquela que causa a execução de uma sentença (ou coleção de sentenças) zero, uma ou mais vezes
- Também chamada de laço (loop)
- Questões de projeto
 - Como a iteração é controlada?
 - Onde o mecanismo de controle deve aparecer na construção do laço?

Sentenças de iteração

- O **corpo** de uma construção iterativa é a coleção de sentenças cuja a execução é controlada pela sentença de iteração
- A sentença de iteração junto com o corpo é chamada de **construção iterativa**

Laços controlados por contador

- Uma sentença de iteração de contagem tem uma variável, chamada **variável do laço**, onde o valor da contagem é mantida
- Existe uma maneira de especificar o valor inicial, o valor final e o passo da variável do laço
- Questões de projeto
 - Qual é o tipo e o escopo da variável do laço?
 - A variável do laço pode ser alterada? Isto altera o controle do laço?
 - Os parâmetros do laço devem ser avaliados uma única vez, ou em cada iteração?

Laços controlados por contador

- Exemplos em Fortran 95

```
Do label var = initial, terminal [, stepsize]
```

```
Do var = initial, terminal [, stepsize]
```

```
...
```

```
End Do
```


- Exemplos em Fortran 95
 - A variável do laço deve ser do tipo inteiro
 - A variável do laço pode não ser alterada
 - O parâmetros são avaliados apenas uma vez e podem ser alterados
 - Não é possível saltar para dentro do laço

Laços controlados por contador

- Exemplo em Ada:

```
for variable in [reverse] discrete_range loop
    ...
end loop;
```

```
Count : Float := 1.35;
for Count in 1..10 loop
    Sum := Sum + Count
end loop;
```

- Exemplo em Ada:
 - O tipo da variável do laço é definida pelo intervalo
 - A variável do laço só existe dentro do laço e não pode ser alterada pelo programador
 - O intervalo pode ser alterado, mas não afeta a execução do laço
 - Não é possível saltar para dentro do laço

Laços controlados por contador

- Exemplo das linguagens baseadas em C:

```
for (exp1; exp2; exp3)
    loop body
```

```
// C
```

```
for (count = 1; count <= 10; count++) {
    ...
}
```

```
// C99, C++, Java
```

```
for (int count = 1; count <= len; count++) {
    ...
}
```

Laços controlados por contador

- Exemplo das linguagens baseadas em C:
 - Não existe uma variável de laço específica
 - `exp1` é avaliada apenas uma vez antes da execução do laço
 - `exp2` e `exp3` são avaliadas a cada iteração
 - Qualquer expressão pode ser alterada

Laços controlados logicamente

- O controle do laço é baseado em uma expressão booleana (não em um contador)
- Questões de projeto
 - O controle deve ser pré-testado ou pós-testado?
 - O laço controlado logicamente deve ser uma forma especial de laço controlado por contador ou uma sentença separada?

Laços controlados logicamente

- Exemplo das linguagens baseadas em C

```
while (control_expression)  
    loop body
```

```
do  
    loop body  
while (control_expression);
```

- Fortran 95 não tem laço lógico
- Ada tem apenas um laço pré-testado

Laços com controles posicionados pelo usuário

- Em algumas situações é conveniente para o programador escolher o local do controle do laço
- Questões de projeto
 - O mecanismo condicional deve ser parte integral da saída?
 - Apenas um corpo de laço pode ser terminado, ou laços externos também podem ser terminados?

Laços com controles posicionados pelo usuário

- Exemplo em Java

```
while (sum < 1000) {  
    getNext(value);  
    if (value < 0) continue;  
    sum += value;  
}
```

```
while (sum < 1000) {  
    getNext(value);  
    if (value < 0) break;  
    sum += value;  
}
```

Laços com controles posicionados pelo usuário

- Exemplo em Java

```
outerLoop:
```

```
for (row = 0; row < numRows; row++) {  
    for (col = 0; col < numCols; col++) {  
        sum += mat[row][col];  
        if (sum > 1000.0) {  
            break outerLoop;  
        }  
    }  
}
```

Iteração baseada em estrutura de dados

- A iteração é controlada pelos elementos em uma estrutura de dados
- A função responsável por percorrer a estrutura de dados é chamada **iterador**
- No início de cada iteração o iterador é chamado, a cada chamada um valor é retornado (em uma ordem específica)

Iteração baseada em estrutura de dados

- Devido a flexibilidade do `for` do C, ele pode ser usado para simular uma iteração definida pelo usuário

```
for (ptr = root; ptr != null; ptr = traverse(ptr))  
    ...
```

- Exemplo em PHP

```
reset $list;  
print ("First number: " + current($list) + "<br />");  
while ($current_value = next($list))  
    print ("Next number: " + $current_value + "<br />");
```

- Exemplo em C#

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

Desvio incondicional

Desvio incondicional

- Uma **sentença de desvio incondicional** transfere o controle da execução para um local específico no programa
- Muito debatido no final da década de 1960
- goto na maioria das linguagens
- É a sentença mais poderosa de controle de fluxo
- Linguagens sem goto: Java, Python, Ruby

Comandos guardados

- Sugerido por Dijkstra em 1975
- A ideia era dar suporte a uma metodologia que garantisse a corretude durante o desenvolvimento
- Outra motivação era o não determinismo, que as vezes é necessário em programas concorrentes

- Construção de seleção

```
if i = 0 -> sum := sum + i
[] i > j -> sum := sum + j
[] j > i -> sum := sum + i
fi
```

```
if x >= y -> max := x
[] y >= x -> max := y
fi
```

- Construção de laço

```
do q1 > q2 -> temp := q1; q1 := q2; q2 := temp;  
[] q2 > q3 -> temp := q2; q2 := q3; q3 := temp;  
[] q3 > q4 -> temp := q3; q3 := q4; q4 := temp;  
od
```

- Robert Sebesta, Concepts of programming languages, 9ª edição. Capítulo 8.