unesp

Linguagens de Programação

Celso Olivete Júnior

olivete@fct.unesp.br

unesp

Na aula passada

- Expressões
- Precedência e associatividade de operador
- Sobrecarga de operador
- Expressões de modo misto
- Várias formas de atribuição

unesp

Na aula de hoje

Estruturas de controle no nível de sentença

unesp

Roteiro

1. Introdução
2. Sentenças de seleção
3. Sentenças de iteração
4. Desvio incondicional
5. Comandos protegidos
6. Conclusões

Níveis de fluxo de controle

- Computações são realizadas por meio da avaliação de expressões e da atribuição dos valores a variáveis

- Para tornar a computação mais flexível e poderosa desenvolveu-se:

  - Formas de selecionar entre caminhos alternativos de fluxo de controle (execução da sentença)

  - Execução repetida de sentenças ou de sequência de sentenças → chamada **de controle de fluxo**

Níveis de fluxo de controle

- **O controle do fluxo** em um programa ocorre em diversos níveis
  - Dentro das expressões (cap.7)
    - Regras de associatividade
    - Regras de precedência de operadores
  - Entre as unidades do programa (cap.9)
    - Será discutido futuramente
  - Entre as sentenças (cap.8)
    - Aula de hoje

Sentenças de controle: evolução

- Sentenças de controle ocorreram na LP FORTRAN I - foram baseadas diretamente no hardware do IBM 704
  - Instruções em linguagem de máquina

- Muita pesquisa e discussão sobre o assunto - sentenças de controle - nos anos 1960
  - Um resultado importante: foi provado que todos os algoritmos que podem ser expressos por diagramas de fluxo podem ser codificados em uma LP com apenas duas sentenças de controle
    1. Uma para escolher dentre dois caminhos de fluxo de controle
    2. Controlar logicamente iterações

Estrutura de controle

- Uma estrutura de controle é uma sentença de controle e a coleção de sentenças cuja execução ela controla

- Questão de projeto
  - A estrutura de controle deve ter múltiplas entradas?
    - Afetam a legibilidade
    - Ocorrem apenas em LP que incluem *goto* e rótulos (*labels)* de instruções

Sentenças de seleção

- Uma *sentença (instrução) de seleção* fornece os meios para escolher entre dois ou mais caminhos de execução em um programa

- Duas categorias gerais:
  1. Dois caminhos
  2. Seleção múltipla, ou n caminhos

Dois caminhos

- Forma geral:

```
if expressão_de_controle
        cláusula então
        cláusula senão
```

- 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 seletores aninhados deve ser especificado?

A expressão de controle

- Expressões de controle são especificadas entre parênteses se a palavra reservada **then** (ou algum outro marcador sintático) não for usada para introduzir a cláusula **então**. Ex:
  - Linguagem C if **(**expressao**)** comando else comando;
  - Linguagem Pascal if expressao then comando else comando

- Em C89, C99, Python e C++, a expressão de controle pode ser aritmética

- Em linguagens como Ada, Java, Ruby e C#, a expressão de controle deve ser booleana

Forma da cláusula

- Em muitas linguagens contemporâneas, as cláusulas **então** e **senão** aparecem ou como **sentenças simples ou como sentenças compostas**
- Em Perl, todas as cláusulas **então** e **senão** devem ser sentenças compostas
- Em Fortran 95, Ada e Ruby, as cláusulas **então** e **senão** são sequências de sentenças
- As linguagens baseadas em C usam chaves para formar sentenças compostas – quando são necessárias?

Simples

```
x = 2;
```

Composta

```
{
  i = 3;
  printf("%d\n", i);
  i++;
}
```

Forma da cláusula

- Python usa indentação para especificar sentenças compostas

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

- Todas as indentadas igualmente pertencem a uma sentença composta

Aninhando seletores

- Exemplo em Java

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

- Para qual **if** a cláusula **else** está associada?
- Regra de semântica estática de Java: **else** sempre casa com a instrução **if** mais próxima

Aninhando seletores (cont.)

- Para forçar a semântica alternativa em Java, o **if** interno é colocado em uma sentença composta, como em:

```
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 **então** e **senão** sejam compostas

Aninhando seletores (cont.)

- Sequências de sentenças como cláusulas:

Ruby

```
if sum == 0 then
    if count == 0 then
      result = 0
    end
else //casa com o IF mais externo
      result = 1
end
```

Aninhando seletores (cont.)

- Python (usa indentação)

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

Aninhando seletores (cont.)

- Python (usa indentação)

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

Construções de seleção múltipla

- Permite a seleção de uma dentre qualquer número de sentenças ou de grupos de sentenças (por exemplo: ***switch***)

- Questões de projeto:
  1. Qual é a forma e o tipo da expressão que controla a seleção?
  2. Como são especificados os segmentos selecionáveis?
  3. O fluxo de execução por meio da estrutura pode incluir apenas um único segmento selecionável?
  4. Como os valores de cada caso são especificados?
  5. Como valores da expressão de seleção que não estão representados devem ser manipulados, se é que o devem?

Exemplos de seletores múltiplos

- C, C++ e Java

```
switch (expressão) {
            case expressão_constante_1: sentença_1;
            …
            case constante_n: sentença_n;
            [default: sentença_n+1]
            }
```

Exemplos de seletores múltiplos

- C, C++ e Java

```
switch (index) {
            case 1: {
                        sentença_1;
                        break;
                    }
            case 2: {
                        sentença_2;
                        break;
                    }
            ...
            default: print("Erro no switch")
            }
```

- O que ocorreria se o **break** fosse omitido?

Exemplos de seletores múltiplos

- Escolhas de projeto para o *switch* de C
  1. A expressão de controle pode ser apenas do tipo inteiro
  2. As sentenças selecionáveis podem ser sequências de sentenças, sentenças compostas ou blocos
  3. Qualquer número de segmentos pode ser executado em uma execução da construção
  4. O segmento opcional default é usado para valores não representados (se o valor da expressão de controle não é representado e nenhum segmento padrão está presente, a construção não faz nada)

Exemplos de seletores múltiplos

- Exemplo: *switch* em C

```
switch (index) {
            case 1:
            case 3: soma+=1;
            case 2:
            case 4: subtracao-=1;
            default: print("Erro no switch");
            }
```

Exemplos de seletores múltiplos

- Exemplo: *switch* em C

```
switch (index) {
            case 1:
            case 3: soma+=1;
            case 2:
            case 4: subtracao-=1;
            default: print("Erro no switch");
            }
```

- Esse código imprime a mensagem de erro a cada execução
- casos 2 e 4 sempre são executados quando os casos 1 e 3 forem

Exemplos de seletores múltiplos

- C#
  - Se difere do C ao ter uma regra de semântica estática que proíbe a execução implícita de mais de um segmento
  - Cada segmento selecionável deve terminar com uma sentença de desvio incondicional explícita (goto ou break)
  - Em C#, expressão de controle e as construções case podem ser cadeias (strings)

Exemplos de seletores múltiplos

- O ***switch*** do PHP é semelhante ao de C, mas permite mais flexibilidade de tipos → valores de cada caso podem ser de qualquer um dos tipos escalares ➔ cadeias, inteiros ou de dupla precisão

Sentenças de iteração

- Sentenças que fazem com que uma sentença ou uma coleção de sentenças seja executada zero, uma ou mais vezes. Uma construção de iteração é frequentemente chamada de um laço (por exemplo: **for**)

- Questões de projeto:
  1. Como a iteração é controlada?
  2. Onde o mecanismo de controle deve aparecer na construção de laço?

Laços controlados por contador

- Uma sentença de controle iterativa de contagem tem uma variável de laço, que inclui os valores inicial e final e o tamanho do passo

- Questões de projeto:
  1. Qual é o tipo e o escopo da variável de laço?
  2. Deve ser legal para a variável ou para os parâmetros de laço serem modificados nele, e, se isso for possível, essa mudança afeta o controle do laço?
  3. Os parâmetros de laço devem ser avaliados apenas uma vez ou uma vez para cada iteração?

Exemplos de sentenças de iteração

- Sintaxe de FORTRAN 95

```
DO rótulo variável = inicial, final [, tamanho do passo]
```

```
do i=0,100,5
      bloco de comandos;
```

- Tamanho do passo pode ser qualquer valor, menos zero
- Parâmetros do laço podem ser expressões e ter valores negativos e positivos

- Questões de projeto:
  1. A variável de laço deve ser do tipo INTEGER
  2. A variável de laço não pode ser mudada no laço, mas os parâmetros podem. Eles são avaliados apenas uma vez, isso não afeta o controle do laço
  3. Parâmetros do laço são avaliados apenas uma vez

Exemplos de sentenças de iteração

- Ada

```
for variável in [reverse] faixa_discreta loop
...
end loop
```

- Questões de projeto:
  1. Uma faixa discreta é uma subfaixa de um tipo inteiro ou de enumeração
  2. Variáveis de laço não existem fora do laço
  3. Se presente a palavra **reverse,** indica que os valores da faixa são atribuídos na ordem reversa
  4. A variável de laço não pode ser mudada no laço, mas a faixa discreta pode. Isso não afeta o controle do laço
  5. A faixa discreta é avaliada apenas uma vez

Exemplos de sentenças de iteração

- Ada - exemplo

```
count : Float : 1.35;
for count in 1..10
    sum = sum + count;
end loop
```

- A variável count não é afetada pelo laço **for**. Quando encerrado o laço, seu valor será **1.35**

Exemplos de sentenças de iteração

- Linguagens baseadas em C

```
for ([expr_1] ; [expr_2] ; [expr_3]) statement
```

  - O corpo do laço pode ser uma única sentença, uma sentença composta ou uma sentença nula
  - O valor de uma expressão de sentenças múltiplas é o valor da última sentença na expressão
  - Se a segunda expressão está ausente, é um laço infinito

- Escolhas de projeto:
  1. Tudo pode ser mudado no laço
  2. A primeira expressão é avaliada uma vez, mas as outras duas são avaliadas com cada iteração

Exemplos de sentenças de iteração

- C++ se difere de C de duas maneiras:
  1. A expressão de controle pode ser booleana
  2. A primeira expressão pode incluir definições de variáveis (o escopo de uma variável definida na sentença for é a partir de sua definição até o final do corpo do laço)

```
for ([expr_1] ; [expr_2] ; [expr_3]) statement
```

- Java e C#
  - Diferem de C++ porque a expressão de controle de laço é restrita a valores booleanos

Sentenças de iteração: laços controlados logicamente

- Controle de repetição é baseado em uma expressão booleana e não em um contador

- Questões de projeto:
  1. O controle deve ser de pré ou pós-teste?
  2. O laço controlado logicamente deve ser uma forma especial de um laço de contagem ou uma sentença separada?

Sentenças de iteração: laços controlados logicamente

- C e C++ incluem tanto laços controlados logicamente com pré-teste quanto com pós-teste:

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

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

- Java é semelhante a C e C++, exceto que a expressão de controle deve ser booleana

Sentenças de iteração: laços controlados logicamente

- Ada tem um laço lógico com pré-teste, mas nenhuma versão pós-teste
- FORTRAN 95 não tem um laço lógico, nem com pré-teste, nem com pós-teste
- Perl e Ruby têm dois laços lógicos com pré-teste: while e until. Perl também tem dois laços com pós-teste

Sentenças de iteração: laços controlados logicamente

- Exemplo de laços com pré-teste em Perl

```
#!/usr/bin/perl
$i = 1;
print "primeiro a raiz quadrada de 1 até 10...\n\n";
while($i <= 10)
 {
        print "A raiz quadrada de ", $i, " é", sqrt($i), "\n";
        $i++;
 }
$i = 1;
print "e agora os quadrados de 1 até 10...\n\n";
until($i > 10)
 {
        print "O quadrado de ", $i, " é", $i * $i, "\n";
        $i++;
 }
```

Mecanismos de controle de laços posicionados pelo usuário

- Em algumas situações, é conveniente para um programador escolher uma posição para o controle do laço em vez do início ou o final do laço

- Questões de projeto
  1. O mecanismo condicional deve ser uma parte integral da saída?
  2. É possível sair apenas de um corpo de laço ou e possível sair também dos laços que o envolvem?

Mecanismos de controle de laços posicionados pelo usuário: `break` e `continue`

- C , C++, Python, Ruby e C# têm saídas não rotuladas incondicionais (break)
- Java e Perl têm saídas incondicionais rotuladas (break em Java, last em Perl)
- C, C++ e Python incluem uma sentença de controle não rotulada, continue, que transfere o controle para o mecanismo de controle do menor laço que o envolve
- Java e Perl têm sentenças similares ao continue

Mecanismos de controle de laços posicionados pelo usuário: `break` e `continue`

- **`break`** e **`continue`** ➔ exemplos

Retorna o controle para o mecanismo do menor laço

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

Mecanismos de controle de laços posicionados pelo usuário: `break` e `continue`

- **`break`** e **`continue`** ➔ exemplos

Um valor negativo encerra a repetição

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

Desvio incondicional

- Uma sentença de desvio incondicional transfere o controle da execução para uma posição especificada no programa.
- Mecanismo mais conhecido: **goto**
- Maior preocupação: legibilidade
- Algumas linguagens não têm suporte para **goto** (por exemplo, Java)
- C# oferece **goto** (pode ser usado em sentenças **switch**)
- Sentenças de saída de laços são restritas e camuflam sentenças **goto**

Comandos protegidos

- Finalidade: fornecer sentenças de controle que suportariam uma metodologia de projeto de programas que garantisse a corretude durante o desenvolvimento
- Base para dois mecanismos linguísticos para programação concorrente (em CSP e Ada)
- Ideia básica: se a ordem de avaliação não é importante, o programa não deve especificar uma

Seleção de comandos protegidos

- Forma

```
if <expressão booleana> -> <sentença>
[] <expressão booleana> -> <sentença>
 ...
[] <expressão booleana> -> <sentença>
fi
```

- **fi** é a abertura escrita de trás para frente ➔ oriunda de Algol 68

- Semântica: quando a construção é alcançada:
  - Avalia todas as expressões booleanas
  - Se mais de uma é verdadeira, escolhe uma de maneira não determinística para execução
  - Se nenhuma é verdadeira, ocorre um erro em tempo de execução

Seleção de comandos protegidos

- Exemplo

```
if i = 0 → soma := soma + i;
[] i > j → soma := soma + j;
[] j > i → soma := soma + i;
fi
```

1. Se i = 0 e j > i então é escolhida, de forma não determinística, entre as sentenças de atribuições 1 e 3
2. Se i = j e não é zero, um erro em tempo de execução ocorre porque nenhuma das condições é igual a 0

Comandos protegidos

- A conexão entre sentenças de controle e verificação de programas é grande
- A verificação é impossível com sentenças goto
- A verificação é simplificada com apenas laços lógicos e seleções ou apenas comandos protegidos

Conclusões

- Variedade de estruturas no nível de sentença
- Programas escritos com apenas seleção e laços lógicos com pré-teste são geralmente menos naturais em sua estrutura, mais complexos e, dessa forma, mais difíceis de serem escritos e de serem lidos
- As estruturas de controle das linguagens de programação funcionais e programação lógica são todas bastante diferentes

Exercícios

- Questões de revisão: 1, 4, 5, 7, 8 e 13
- Conjunto de problemas: 1, 4 e 13
- Exercícios de programação: 1, 3 e 4 (considere apenas a linguagem C)