Aula 03

Estruturas de controlo repetitivas

Instruções de atribuição com operação e instrução repetitiva

Programação 1, 2015-2016

v1.1, 08-10-2015

, DETI, Universidade de Aveiro

Conteúdo

1 Instrução de atribuição com operação

Atribuição com operação

- Operadores de pré-incremento (`++i`) e pós-incremento (`i++`).
- É comum usar uma versão compacta do operador de atribuição (=) onde este é precedido de uma operação (por exemplo +=, −=, ⋆=, /=, %=,...).
- A instrução resultante é equivalente a uma instrução normal de atribuição em que a mesma variável aparece em ambos os lados do operador =.
- A importância desta notação tem a ver com a simplificação do código e com a clareza da operação a realizar.

```
int x, y, z;
...
x = ++y;     // equivalente a: y = y + 1; x = y;
x = y++;     // equivalente a: x = y; y = y + 1;
y += 5;      // equivalente a: y = y + 5;
z *= 5 + x; // equivalente a: z = z * (5 + x);
y += ++x;    // x = x + 1; y = y + x;
```


2 Estruturas de controlo: repetição

Estruturas de controlo: repetição

- Para além da execução condicional de instruções, por vezes existe a necessidade de executar instruções repetidamente.
- A um conjunto de instruções que são executadas repetidamente designamos por *ciclo*.
- Um ciclo é constituído por uma estrutura de controlo que controla quantas vezes as instruções vão ser repetidas.
- As estruturas de controlo repetitivas podem ser mais do tipo *condicional* (`while` e `do...while`) ou mais do tipo *contador* (`for`).
- Normalmente utilizamos as estruturas repetitivas do tipo condicional quando o número de iterações é desconhecido e as estruturas repetitivas do tipo contador quando sabemos à partida o número de iterações.
- No entanto, podemos de uma forma relativamente fácil converter um tipo no outro.

03.4

2.1 Instrução repetitiva `while` e `do...while`

Instrução repetitiva `while` e `do...while`

```
do {
   // instruções
} while(condição);
```

```
while(condição) {
   // instruções
}
```

- A sequência de instruções colocadas no corpo do ciclo são executadas enquanto a condição for verdadeira.
- Quando a condição for falsa, o ciclo termina e o programa continua a executar o que se seguir.
- A diferença principal entre as duas instruções repetitivas reside no facto de no ciclo `do...while` o bloco de instruções é executado pelo menos uma vez, enquanto no ciclo `while` pode não ser executada nenhuma vez.
- Muito cuidado na definição da condição...

Ciclos com sentido: Garantir sempre que o bloco de instruções do ciclo altera a condição, e que é possível tornar a condição falsa com a execução repetida do bloco.

03.5

Exemplo da leitura de um valor inteiro positivo

```
int x, cont = 0;
do {
   System.out.print("Um valor inteiro positivo: ");
   x = sc.nextInt();
   cont++;
} while(x <= 0);
System.out.printf("Valor %d lido em %d tentativas\n", x, cont);
```

```
int x = -1, cont = 0; // Atenção à inicialização de x
while(x <= 0) {
   System.out.print("Um valor inteiro positivo: ");
   x = sc.nextInt();
   cont++;
}
System.out.printf("Valor %d lido em %d tentativas\n", x, cont);
```

03.6

2.2 Instrução repetitiva `for`

Instrução repetitiva `for`

```
for(inicialização ; condição ; atualização) {
   // instruções
}
```

- A `inicialização` é executada em primeiro lugar e apenas uma vez.
- A `condição` é avaliada no início de todos os ciclos e as instruções são executadas enquanto a `condição` for verdadeira.
- A parte da `atualização` é feita no final de todas as iterações.
- Em geral, a função da `inicialização` e da `atualização` é manipular variáveis de contagem utilizadas dentro do ciclo.
- Ciclo `for` pode ser convertido num ciclo `while`:

```
inicialização
while(condição) {
   // instruções
   atualização
}
```

03.7

Exemplo: Impressão da tabuada de n com $n \leq 10$

```
...
int i, n;
do {
   System.out.print("Tabuada do: ");
   n = sc.nextInt();
} while(n < 1 || n > 10);
for(i = 1 ; i <= 10 ; i++) {
   System.out.printf("%2d X %2d = %3d\n", n, i, n*i);
}
```

03.8

2.3 Instruções de salto `break` e `continue`

Instruções de salto `break` e `continue`

- Podemos terminar a execução de um bloco de instruções com duas instruções especiais: `break` e `continue`.
- A instrução `break` permite a saída imediata do bloco de código que está a ser executado. É usada normalmente no switch e em estruturas de repetição, terminando-as.
- A instrução `continue` permite terminar a execução do bloco de instruções dentro de um ciclo, forçando a passagem para a iteração seguinte (não termina o ciclo).
- O código:

```
for(A ; COND1 ; B) {
   C
   if (COND2)
      continue;
   D
}
```

- é equivalente a:

```
for(A ; COND1 ; B) {
   C
   if (!(COND2)) {
      D
   }
}
```

03.9

Exemplo (1)

```
int x, cont = 0;
do {
   System.out.print("Um valor inteiro positivo: ");
   x = sc.nextInt();
   cont++;
   if(cont >= 10) // depois de 10 tentativas, termina o ciclo
      break;
} while(x <= 0);
if(x > 0) {
   System.out.printf("Valor %d lido em %d tentativas\n",x,cont);
}
else {
   System.out.printf("Ultrapassadas 10 tentativas\n");
}
```

03.10

Exemplo (2)

```
int i, n, soma = 0;
do {
   System.out.print("Valor de N [1...99]: ");
   n = sc.nextInt();
} while(n < 1 || n > 100);
for(i = 1 ; i <= n ; i++){
   // se numero par avança para a iteração seguinte
   if(i % 2 == 0) {
      continue;
   }
   soma += i;
}
System.out.printf("A soma dos impares e' %d\n", soma);
```

03.11

Ciclos "mentirosos"

- A utilização das instruções de "salto" pode levantar alguns problemas na correcta compreensão do algoritmo.
- Vejamos este exemplo:

```
int i, n;
for(i = 1; i <= MAX; i++) {
   System.out.print("Number: ");
   n = sc.nextInt();
   if (n % 2 == 0) // condição de saída do ciclo
      break;
}
```

- O que é que esperamos que aconteça no fim do ciclo?
- Se apenas olharmos para a condição de manutenção do ciclo, esperaríamos que a variável `i` tivesse o valor `(MAX+1)`!
- Isso não é necessariamente verdade, logo a condição de manutenção de ciclo expressa não é necessariamente verdadeira.
- Torna-se necessário, apenas para extrair a condição de manutenção do ciclo, compreender todo o código do mesmo.

03.12

- O mesmo código pode ser reescrito da seguinte forma:

```
int i, n;
i = 1;
do
{
   System.out.print("Number: ");
   n = sc.nextInt();
   i++;
} while (i <= MAX && n % 2 != 0); // condição de manutenção
                                  // do ciclo
```

- Aqui não precisamos de olhar para as instruções dentro do ciclo para extrair a condição de manutenção que se lhe aplica.
- Note que o condição do `if/break` foi negada quando passou para o `do...while`.

Instruções de "salto": Deve-se evitar a utilização abusiva de instruções de salto nos algoritmos.

03.13

2.4 Ciclos que terminam a meio

- Na utilização de instruções repetitivas podemos identificar os seguintes padrões de repetição (em pseudo-código):

```
loop while (COND)
    A
end
// AA... -> (A)*
```

```
loop
    A
while (COND) end
// A AA... -> A(A)*
```

```
loop
    A
while (COND)
    B
end
// A BABA... -> A(BA)*
```

- Se para os dois primeiros a linguagem `Java` fornece uma solução adequada, o terceiro caso – de ciclos que terminam a meio – requer algum trabalho extra.
- Podemos implementá-lo das seguintes formas:

```
while(COND) {
    A
    if ((COND)) {
        B
    }
}
```

```
A
while(COND) {
    B
    A
}
```

```
while(true) {
// the condition is
// imposed inside
// by an if/break
    A
    if !(COND))
        break;
    B
}
```

03.14

3 Instrução repetitiva e erros externos

- Na aula anterior já vimos um padrão algorítmico para sistematicamente lidar com erros externos assente na instrução condicional e na terminação do programa.
- Em certos casos, existe uma outra alternativa para lidar com estas falhas, que consiste simplesmente em repetir o código até que a condição desejada seja garantida:

```
// read an evaluation [0:20] grade:
Scanner sc = new Scanner(System.in);
double grade;
do {
    System.out.print("Grade [0:20]: ");
    grade = sc.nextDouble();
    if (grade < 0 || grade > 20)
        System.err.println("ERROR: invalid grade !");
} while(grade < 0 || grade > 20);
// remaining code can safely assume grade in [0:20]
```

03.15