4 Estruturas de Controle

Onde são apresentadas estruturas que definem a ordem em que comandos são executados em um programa.

Pré-requisito(s):	Objetivos (ao final você deverá ser capaz de):
• Entender o conceito abstrato de variáveis e seu uso • Saber elaborar um pseudocódigo com variáveis, expressões e comandos • Saber escrever um programa em Java que atualiza variáveis, e comandos de leitura e escrita de dados	• Elaborar algoritmos com fluxo de execução seqüencial, fluxos alternativos e fluxos repetitivos • Reconhecer que tipo de estrutura de repetição mais se aplica a um determinado problema • Conhecer as estruturas de controle em Java • Criar programas Java combinando as diversas estruturas de controle

Já sabemos que na elaboração de um algoritmo para solução de algum problema podemos fazer uso de constantes e variáveis, formar e avaliar expressões, e definir comandos. Sabemos também que um algoritmo de sucesso deve descrever uma ordem lógica de ações que invariavelmente leve à solução desse problema. Essas ações são formadas pelos elementos descritos acima e que você já aprendeu como fazê-lo nos capítulos anteriores. Entretanto, é bem provável que na busca pelo algoritmo ideal para solução de algum dos problemas apresentados e sugeridos até aqui, você tenha se deparado com a necessidade de criar, por exemplo, caminhos alternativos de execução que dependam de algum determinado teste; ou ainda com a necessidade de repetir um número fixo ou indefinido de vezes um determinado trecho de código. Este capítulo apresenta como podemos implementar essas variações de fluxos de execução. Em particular, o capítulo apresenta três tipos diferentes de estruturas de controle de fluxo a saber: estruturas seqüenciais, estruturas de decisão, e estruturas de repetição.

4.1 Estrutura de Controle Seqüencial

Um algoritmo com estrutura totalmente seqüencial é composto por um conjunto de comandos que são executados de maneira seqüencial, seguindo a ordem em que aparecem. Esses comandos são separados por ponto-e-vírgula (;). Todos os algoritmos e programas ilustrados no capítulo 3 possuem estrutura seqüencial.

A figura 4.1 traz um algoritmo com estrutura seqüencial em formato de fluxograma para cálculo da média final de duas notas obtidas por um aluno. Note que o algoritmo executa dois comandos de leitura seguidos por um comando de atribuição do resultado da avaliação de uma expressão aritmética e finaliza com um comando de escrita.

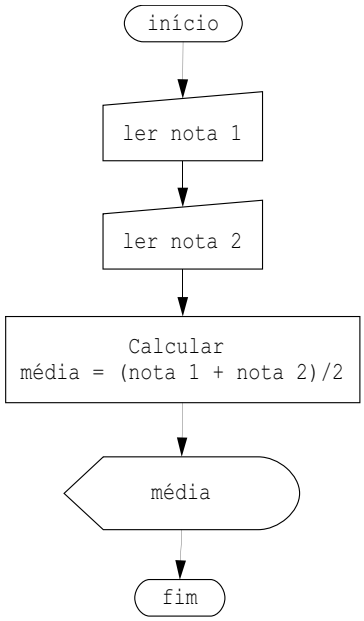

Figura 4.1 – Fluxograma de um algoritmo com estrutura seqüencial.

O modelo geral de um pseudocódigo com essa estrutura está ilustrado na figura 4.2.

```
1.  Algoritmo EstruturaSequencial;
2.  Variáveis
3.     ...; //declaração de variáveis
4.  Início
5.     comando 1;
6.     comando 2;
7.     ...
8.     comando i;
9.     ...
10.    comando n;
11. Fim.
```

Figura 4.2 – Modelo de um pseudocódigo com estrutura seqüencial.

O pseudocódigo equivalente ao fluxograma do exemplo da figura 4.1 está ilustrado na figura 4.3.

```
1.  Algoritmo Media;
2.  Variáveis
3.     nota1, nota2, media: real; //declaração de variáveis
4.  Início
5.     leia(nota1);
6.     leia(nota2);
7.     media <- (nota1 + nota2)/2;
8.     escreva(media);
9.  Fim.
```

Figura 4.3 – Cálculo da média de duas notas com estrutura seqüencial.

Note que o comando de leitura da primeira nota é executado (linha 5), em seguida o comando de leitura da segunda nota é executado (linha 6), na seqüência, o comando que atribui à variável `media` o resultado da avaliação da expressão `(nota1+nota2)/2` é executado (linha 7), e finalmente, o comando de escrita de resultado é executado (linha 8). O fluxo é claramente seqüencial.

4.2 Estruturas de Controle Condicional

Muitas vezes, entretanto, precisamos tomar decisões ao longo do andamento de um algoritmo. Nesses casos, o fluxo de execução dos comandos de um algoritmo é determinado pelos dados, conforme o atendimento de determinadas **condições**. Essas condições são representadas pelas expressões lógicas ou relacionais que vimos no capítulo 2.

Observe o exemplo da figura 4.4. O fluxograma é similar ao do exemplo anterior, mas existe uma condição explícita para escrita da situação final do aluno em questão: se a média for maior ou igual a cinco, será escrito que o aluno está aprovado, caso contrário, será escrito que ele está reprovado.

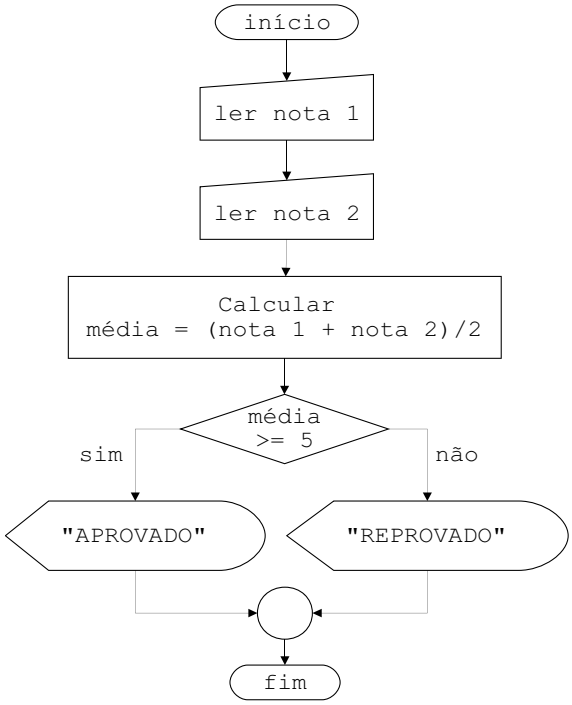

Figura 4.4 – Fluxograma de um algoritmo com estrutura de controle condicional.

4.2.1 Estrutura Condicional Simples

O fluxograma anterior traz uma estrutura de condição simples, pois a partir da avaliação de uma expressão relacional tem-se apenas duas alternativas de fluxo. Essas alternativas são representadas em pseudocódigo por uma **estrutura se-então-senão**.

O modelo geral de um pseudocódigo com essa estrutura está ilustrado na figura 4.5.

```
1.  Algoritmo EstruturaCondicionalSimples;
2.  Variáveis
3.     ...; //declaração de variáveis
4.  Início
5.     comando 1;
6.     comando 2;
7.     ...
8.     se <condições> então
9.       início
10.        comando 3;
11.        comando 4;
12.        ...
13.      fim;
14.    [senão
15.      início
16.        comando 5;
```

```
17.         comando 6;
18.         ...
19.       fim;]
20.     fimse;
21.     ...
22.     comando n;
23. Fim.
```

Figura 4.5 – Modelo geral de um pseudocódigo com estrutura condicional simples.

Observe que se as `<condições>` forem satisfeitas (verdadeiras) apenas o bloco de comandos delimitados pelas linhas 9 e 13 serão executados. Caso as `<condições>` sejam falsas, apenas o bloco de comandos delimitados pelas linhas 15 e 19 serão executados. Note ainda que o bloco `senão` não é necessariamente obrigatório. Esta característica está representada no modelo geral pela presença dos colchetes `[]`.

O pseudocódigo equivalente ao fluxograma do exemplo da figura 4.5 está ilustrado na figura 4.6 a seguir.

```
1.  Algoritmo MediaCondicional;
2.  Variáveis
3.     nota1, nota2, media: real; //declaração de variáveis
4.  Início
5.     leia(nota1);
6.     leia(nota2);
7.     media <- (nota1 + nota2)/2;
8.     se (media >= 5) então
9.        escreva("APROVADO");
10.    senão
11.       escreva("REPROVADO");
12.    fimse;
    Fim.
```

Figura 4.6 – Pseudocódigo com estrutura condicional simples: situação do aluno depende do valor da média calculada.

Note que tanto o bloco `então` quanto o bloco `senão` não utilizam os termos `início` e `fim` para delimitar comandos. De fato, a presença desses termos só se faz necessária quando existe mais de um comando a ser executado naquele bloco. No exemplo da média, apenas um comando de escrita é executado em cada situação.

4.2.2 Estrutura Condicional Aninhada

Em alguns problemas precisamos encadear vários testes de condições. Isso normalmente ocorre quando existe um grande número de possibilidades ou combinações de situações para que um conjunto de comandos possa ser executado.

Nesses casos dizemos que se trata de uma estrutura de condições aninhadas.

Um exemplo de estrutura geral do pseudocódigo para estruturas condicionais aninhadas está ilustrado na figura 4.7. Uma vez que a possibilidade de combinações é infinita, o modelo de pseudocódigo em questão ilustra apenas um exemplo de modelo geral.

```
1.  Algoritmo EstruturaCondicionalAninhada;
2.  Variáveis
3.     ...; //declaração de variáveis
4.  Início
5.     comando 1;
6.     comando 2;
7.     ...
8.     se <condições> então
9.       início
10.        comando 3;
11.        se <condições> então
12.           se <condições> então
13.              comando 4;
14.           senão
15.              comando 5;
16.        senão
17.           comando 6;
18.      fim;
19.    senão
20.      início
21.        comando 7;
22.        se <condições> então
23.           comando 8;
24.        senão
25.           comando 9;
26.      fim;
27.    fimse;
28.    ...
29.    comando n;
30. Fim.
```

Figura 4.7 – Modelo geral de um pseudocódigo com estrutura condicional aninhada.

Considere como exemplo, um algoritmo que verifica se três valores lidos podem representar lados de um triângulo e, em caso positivo, informa de que tipo de triângulo se trata. Sabemos que cada lado de um triângulo possuem a restrição de que seu tamanho deve ser menor que a soma dos tamanhos dos outros dois lados. Sabemos também que um triângulo se diz *eqüilátero* quando possui três lados iguais, *isósceles*, quando possui apenas dois lados iguais, ou *escaleno*, quando todos os três lados possuem tamanhos diferentes.

O pseudocódigo para solução deste problema está ilustrado na figura 4.8.

```
1.  Algoritmo Triangulo;
2.  Variáveis
3.     l1, l2, l3: inteiro;
4.  Início
5.     leia(l1, l2, l3);
6.     se (l1 < l2 + l3) e (l2 < l1 + l3) e (l3 < l1 + l2) então
7.        se (l1 = l2) e (l2 = l3) então
8.           escreva("TRIÂNGULO EQUILATERO");
9.        senão
10.          se (l1 = l2) ou (l1 = l3) ou (l2 = l3) então
11.             escreva("TRIÂNGULO ISÓSCELES");
12.          senão
13.             escreva("TRIÂNGULO ESCALENO");
14.          fimse;
15.       fimse;
16.    senão
17.      escreva("NÃO FORMAM UM TRIÂNGULO");
18.    fimse;
19. Fim.
```

Figura 4.8 – Pseudocódigo com estrutura condicional aninhada: tipo do triângulo depende da relação entre os tamanhos dos três lados lidos.

Na linha 3, as três variáveis do problema são declaradas, que correspondem aos três lados do triângulo, que serão lidos na linha 5.

A primeira estrutura condicional `se`, na linha 6, testa as três condições necessárias para que os lados possam representar um triângulo; note o uso do operador lógico `"e"` entre as condições, que indica a necessidade das três condições terem que ser satisfeitas para que o bloco `então` possa ser executado. Caso pelo menos uma delas não seja verdadeira, o bloco `senão` desse `se` (linha 16) será executado e a mensagem `"NÃO FORMAM UM TRIÂNGULO"` será impressa.

Caso as três condições sejam satisfeitas, há um segundo teste, na linha 7, para determinar se se trata de um triângulo *equilátero*; mais uma vez, as duas condições devem ser satisfeitas para que isso seja verdade e o comando da linha 8 possa ser executado. Caso uma das duas condições não seja verdade, resta-nos saber se pelo menos uma delas é satisfeita (por isso o uso do operador lógico `"ou"` ao invés do `"e"`), o que automaticamente determinaria o triângulo como sendo do tipo *isósceles*. Se, no entanto, nenhuma dessas condições fossem verdadeiras, então se trata de um triângulo escaleno.

4.2.3 Estrutura de Múltipla Escolha

Um formato de estrutura condicional aninhada que ocorre com bastante freqüência em problemas é quando existe uma determinada ação a ser executada para cada valor possível de um determinado teste. Nesses casos, temos um encadeamento de testes do tipo `se-senão-se`.

Considere como exemplo um algoritmo que efetua a leitura de um número correspondente a um mês do ano e exibe o nome deste mês por extenso como resultado (figura 4.9).

```
1.  Algoritmo Meses;
2.  variáveis
3.     mes : inteiro;
4.  início
5.     leia (mes);
6.     se (mês = 1) então
7.        escreva ("Janeiro");
8.     senão
9.        se (mês = 2) então
10.          escreva ("Fevereiro");
11.       senão
12.         se (mês = 3) então
13.            escreva ("Março");
14.         senão
15.            ...
16.
17.                         se (mês = 12) então
18.                            escreva ("Dezembro");
19.                         senão
20.                            escreva ("Mês inexistente");
21. fim.
```

Figura 4.9 – Algoritmo com vários blocos `se-senão-se` aninhados.

Note que a depender do número de valores possíveis, a quantidade de blocos `se-senão-se` pode ser muito grande e dificultar bastante a leitura e entendimento do algoritmo em questão.

Para esses casos específicos, uma estrutura alternativa é aconselhável: a **estrutura condicional de múltipla escolha**. Com essa estrutura, o comando associado ao valor em teste é executado diretamente sem a necessidade de se navegar por toda a estrutura condicional tradicional.

O formato geral da estrutura de múltipla escolha está ilustrado na figura 4.10.

```
1.  Algoritmo EstruturaMultiplaEscolha;
2.  Variáveis
3.     ...; //declaração de variáveis
4.  Início
5.     comando 1;
6.     comando 2;
7.     ...
8.     caso <variável ou expressão> seja
9.       valor 1: comando 3;
10.      valor 2: comando 4;
11.      valor 3: comando 5;
12.      valor 4: comando 6;
13.      ...
```

```
14.       nenhum: comando n;
15.     fimcaso;
16. Fim.
```

Figura 4.10 – Modelo geral de estrutura de múltipla escolha.

Utilizando essa estrutura, o pseudocódigo para o problema da impressão dos nomes dos meses seria reescrito como ilustrado na figura 4.11.

```
1.  Algoritmo Meses;
2.  variáveis
3.     mes : inteiro;
4.  Início
5.     leia (mes);
6.     caso (mes) seja
7.        1: escreva ('Janeiro');
8.        2: escreva ('Fevereiro');
9.        3: escreva ('Março');
10.       4: escreva ('Abril');
11.       5: escreva ('Maio');
12.       6: escreva ('Junho');
13.       7: escreva ('Julho');
14.       8: escreva ('Agosto');
15.       9: escreva ('Setembro');
16.      10: escreva ('Outubro');
17.      11: escreva ('Novembro');
18.      12: escreva ('Dezembro');
19.  nenhum: escreva ('Mês inexistente');
20.    fimcaso;
21. Fim.
```

Figura 4.11 – Substituição de vários blocos `se-senão-se` aninhados por uma estrutura de múltipla escolha.

A depender do valor da variável lida `mes`, o comando de escrita do nome do mês correspondente será executado. Caso o usuário entre com um valor diferente dos previstos, então será escrito que o mês especificado é inexistente.

4.3 Estruturas de Repetição

Vários problemas de programação se caracterizam pela repetição de um conjunto de instruções durante um determinado tempo ou até que uma condição seja satisfeita.

Considere, por exemplo, mais uma vez, o algoritmo de cálculo da média final de um aluno em uma disciplina (pseudocódigo da figura 4.6). Como visto, o algoritmo em questão será executado apenas uma vez, ou seja, realiza o cálculo da média para um único aluno. Mas e se quiséssemos calcular a média de todos os alunos de uma determinada turma? Teríamos que escrever o mesmo código tantas

vezes quanto forem a quantidade de alunos na turma (ex: 40 alunos = 40 códigos).

O recurso que as linguagens de programação possuem para se evitar a necessidade de se escrever o mesmo código (ou trecho de código) tantas vezes, é utilizar **estruturas de repetição de comandos**. Uma estrutura desse tipo força o algoritmo a retroceder para o início da seqüência de comandos desejada, assim que ela é finalizada. Os trechos do algoritmo que são repetidos são chamados de **laços** (ou *loop*, em inglês).

Alguns autores consideram os laços como uma das justificativas mais fortes para o uso da programação de computadores, uma vez que, ao contrário do ser humano, a máquina é imune ao cansaço físico e mental.

4.3.1 Repetição com teste no início

A estrutura de repetição com teste no início permite que um trecho do algoritmo seja repetido diversas vezes caso uma determinada condição seja satisfeita. Isto é, antes de executar aquele trecho, o algoritmo precisa verificar se a condição (ainda) é verdadeira.

A estrutura geral de repetição com teste no início está ilustrada na figura 4.12.

```
1.  Algoritmo EstruturaEnquanto;
2.  Variáveis
3.     ...; //declaração de variáveis
4.  Início
5.     comando 1;
6.     comando 2;
7.     ...
8.     enquanto <condições> faça
9.       comando 3;
10.      comando 4;
11.      ...
12.      comando n;
13.    fimenquanto;
14. Fim.
15.
16.
```

Figura 4.12 – Modelo geral de estrutura de repetição com teste no início.

Observe que a leitura da estrutura é bem intuitiva: `enquanto` uma condição (ou conjunto de condições) for verdadeira, os comandos serão executados.

Fica claro que um passo importante para uso da estrutura é a definição da(s) condição(ões) de parada de repetição do laço. Para resolvermos o problema do cálculo das médias de todos os alunos de uma turma, por exemplo, devemos utilizar como condição de parada a quantidade de alunos cuja média já tenha sido

calculada; quando esse valor chegar a 0 (zero), não restam mais alunos e portanto o laço repetitivo deve ser encerrado.

O pseudocódigo da figura 4.13 ilustra esse exemplo.

```
1.  Algoritmo MediaTurma;
2.  Variáveis
3.     nota1, nota2, media: real;
4.     numalunos: inteiro;
5.     nome: literal;
6.  Início
7.     leia(numalunos);
8.     enquanto (numalunos > 0) faça
9.       leia(nome);
10.      leia(nota1);
11.      leia(nota2);
12.      media <- (nota1 + nota2)/2;
13.      se (media >= 5) então
14.         escreva("O aluno ",nome," está APROVADO");
15.      senão
16.         escreva("O aluno ",nome," está REPROVADO");
17.      fimse;
18.      numalunos <- numalunos - 1;
19.    fimenquanto;
20. Fim.
```

Figura 4.13 – Exemplo de uso da estrutura `enquanto`: a média e a situação de vários alunos são exibidas.

Na linha 7, pede-se ao usuário que indique o número de alunos existentes na turma em questão. Esse valor será armazenado na variável `numalunos`. O laço de repetição engloba os comandos da linha 9 à linha 18. A condição para essa repetição é que o número de alunos restantes seja maior que zero, obviamente (linha 8). Observe que cada vez que o laço é executado para um determinado aluno, o último comando do laço (linha 18) decrementa o valor armazenado na variável `numalunos`; para uma turma de 40 alunos, por exemplo, ao final do primeiro laço, essa variável irá ser atualizada com o valor anterior dela (40) menos 1 (pois um aluno acabou de ser processado) e seu novo valor será 39. Ao final de 40 repetições, o valor da variável será 0 (zero) e a condição do laço não será mais satisfeita, encerrando a execução do algoritmo.

Este mesmo algoritmo em formato de fluxograma está ilustrado na figura 4.14.

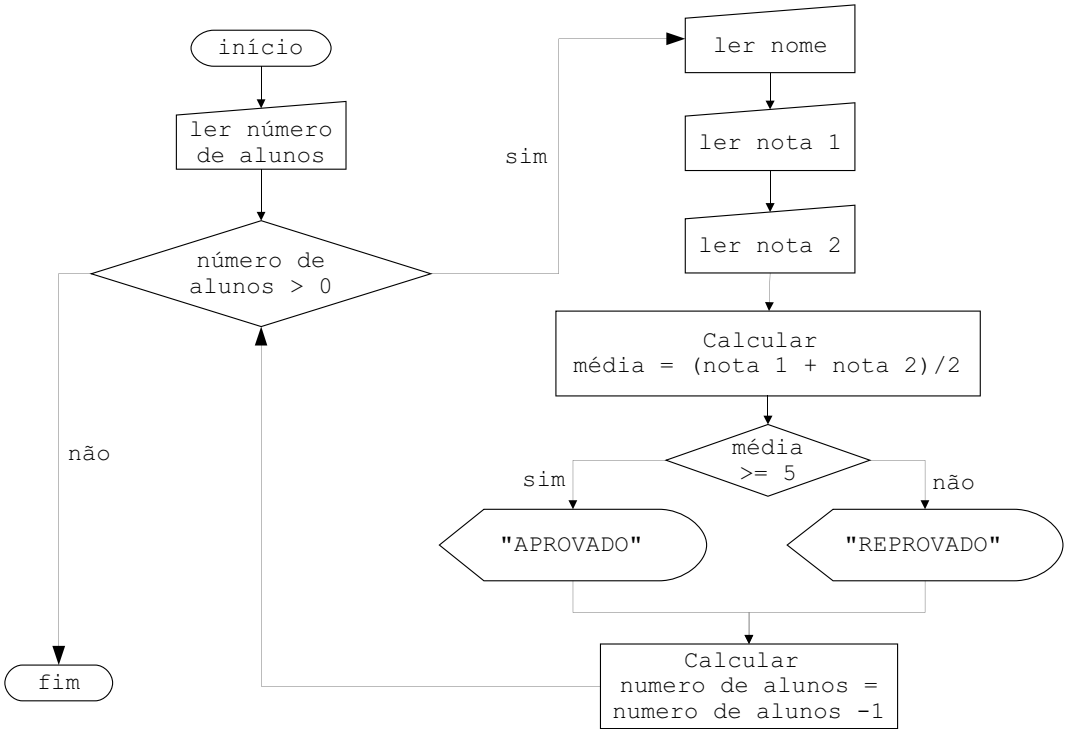

Figura 4.14 – Exemplo de uso da estrutura `enquanto com fluxograma`.

Um exemplo clássico de uso de uma estrutura de repetição é o algoritmo para cálculo do fatorial de um número. O fatorial de um número n (`n!`) é calculado da seguinte forma: `n! = n*(n-1)!`, com `0! = 1`. Na prática, o resultado do fatorial de n pode ser calculado multiplicando-se n sucessivamente por valores inteiros imediatamente menores até o valor 1. Ou seja, `n! = n*(n-1)*(n-2)*....*1`. Observe o comportamento do pseudocódigo da figura 4.15.

```
1.  Algoritmo Fatorial;
2.  variáveis
3.     Resultado, Numero: inteiro;
4.  Início
5.     Resultado <- 1;
6.     leia (Numero);
7.     enquanto (Numero > 1) faça
8.        Resultado <- Resultado * Numero;
9.        Numero <- Numero - 1;
10.    fimenquanto;
11.    escreva (Resultado);
12. Fim.
```

Figura 4.15 – Calculo do fatorial de um número com uso da estrutura `enquanto`.

O número de que se deseja calcular o fatorial é lido na linha 6. Esse número é decrescido a cada repetição executada (linha 9) até que se atinja o valor 1

(condição de parada definida na linha 7). Durante o processo, a variável `Resultado` irá ser atualizada repetidamente multiplicando-se seu valor atual pelo valor da variável `numero` (linha 8), que está sendo decrescida.

4.3.2 Repetição com teste no final

Em alguns casos é mais adequado que o teste de parada seja feito ao final da execução de um bloco de comandos e não no início, como ocorre na estrutura `enquanto-faça`. Com a estrutura de repetição com teste no final o trecho do algoritmo referente ao laço é executado ao menos uma vez, necessariamente. A estrutura geral do pseudocódigo pode ser vista na figura 4.16.

```
1.  Algoritmo EstruturaRepita;
2.  Variáveis
3.     ...; //declaração de variáveis
4.  Início
5.     comando 1;
6.     comando 2;
7.     ...
8.     repita
9.       comando 3;
10.      comando 4;
11.      ...
12.      comando n;
13.    até <condições>;
14. Fim.
```

Figura 4.16 – Modelo geral da estrutura `repita`.

O pseudocódigo da figura 4.17 ilustra um algoritmo para exibir os 25 primeiros termos e o somatório de uma PG (Progressão Geométrica) de razão 3, onde o valor do primeiro termo é lido.

```
1.  Algoritmo ProgressaoGeometrica;
2.  variáveis
3.     Contador,
4.     Termo,
5.     Soma : inteiro;
6.  Início
7.     Contador <- 1;
8.     Soma <- 0;
9.     leia(Termo);
10.    repita
11.       escreva (Termo, ", ");
12.       Soma <- Soma + Termo;
13.       Termo <- Termo * 3;
14.       Contador <- Contador + 1;
15.    até (Contador > 25);
16.    escreva (Soma);
17. Fim.
```

Figura 4.17 – Uso da estrutura repita para escrita de 25 termos e cálculo do somatório de uma PG.

Observe que o primeiro termo é lido (linha 9) e é escrito na saída com o primeiro comando do laço (linha 11) independentemente de qualquer teste. Esse laço será repetido por mais 24 vezes (até que a variável `Contador` armazene o valor 25), quando então o laço não será mais repetido e será escrito o valor do somatório da PG (linha 17).

4.3.3 Repetição com variável de controle

Quando sabemos de antemão o número de repetições a serem efetuadas, podemos utilizar uma estrutura de repetição que executa um bloco de comandos um número predeterminado de vezes, sem a necessidade de testar alguma condição. Esse número de vezes é controlado por uma variável específica para esse fim, chamada de **variável de controle**.

A estrutura geral de uma repetição que faz uso dessa variável está ilustrada na figura 4.18.

```
1.  Algoritmo EstruturaPara;
2.  Variáveis
3.     Controle: inteiro;
4.     ...                 //declaração de outras variáveis
5.  Início
6.     comando 1;
7.     comando 2;
8.     ...
9.     para Controle <- <ValorInicio> até <ValorFim> faça
10.      comando 3;
11.      comando 4;
12.      ...
13.      comando n;
14.    fimpara;
    Fim.
```

Figura 4.18 – Modelo geral da estrutura de repetição com variável de controle.

A variável `Controle` deve ser do tipo inteiro e controla o número de repetições do laço. `<ValorInicio>` indica o valor inicial que a variável controle deve assumir e `<ValorFim>` indica o valor final da variável, ou seja, o valor até o qual ela será incrementada.

O problema da figura 4.17, por exemplo, seria mais facilmente representado através de um pseudocódigo composto pela estrutura de repetição `para`, uma vez que conhecemos de antemão o número de repetições desejadas (figura 4.19).

```
1.  Algoritmo ProgressaoGeometrica;
2.  variáveis
```

```
3.      Contador, Termo, Soma: inteiro;
4.   Início
5.      Soma <- 0;
6.      leia(Termo);
7.      para Contador <- 1 até 25 faça
8.         escreva (Termo, ", ");
9.         Soma <- Soma + Termo;
10.        Termo <- Termo * 3;
11.     fimpara;
12.     escreva (Soma);
13.  Fim.
```

Figura 4.19 – PG com estrutura `para`.

Considere agora o problema do cálculo do valor da expressão $\sum_{N=1}^{50} \frac{1}{N^2}$, onde o valor de `N` é lido. Uma vez que sabemos de antemão, o número de vezes em que a soma será efetuada (50 vezes), é mais um exemplo de algoritmo onde o emprego da estrutura `para` é bem adequado. O pseudocódigo do algoritmo para solução desse problema está ilustrado na figura 4.20.

```
1.  Algoritmo SomatorioInversoQuadrado;
2.  variáveis
3.     N: inteiro;
4.     Somatorio: real;
5.  Início
6.     Somatorio <- 0;
7.     leia(N);
8.     para N <- 1 até 50 faça
9.        Somatorio <- Somatorio + 1/N**2;
10.    fimpara;
11.    escreva (Somatorio);
12. Fim.
```

Figura 4.20 – Uso da estrutura para cálculo de um somatório de N termos.

Observe que a cada passo do laço, o valor da variável `Somatorio` é atualizada com o acréscimo do valor do termo em questão (linha 9). Por razões óbvias, a variável `Somatorio` deve obrigatoriamente ser inicializada com valor 0 (zero).

4.4 Codificação de comandos em Java

4.4.1 Estruturas condicionais em Java

O código Java para cálculo das raízes de uma equação de segundo grau, ilustrado no capítulo 3 pode ser melhorado. Sabemos que uma equação de segundo grau só possui raízes no domínio dos números reais se e apenas se o valor calculado do

delta da equação seja maior ou igual a zero. Certo?

Podemos incluir esse teste fazendo uso de uma estrutura condicional simples, que em Java é representada por

if (condições) {...} else {...},

onde os *{ }* fazem o papel dos termos reservados `início` e `fim`, utilizados em pseudocódigos. O código Java da figura 4.21 ilustra a inclusão do teste do valor do `delta`. As raízes `x1` e `x2` só serão calculadas se `delta >= 0` (linha 13); senão, será escrita a mensagem "`Equação não possui raízes reais!`" (linha 19).

```
1.  public class SegundoGrau {
2.    public static void main ( String args[] ) {
3.      double a, b, c, delta, x1 = 0, x2 = 0;
4.      //Na equação 2X*X+3X-10=0, "a" é o 2, "b" é o 3 e c é o -
5.  10.
6.      System.out.print("Digite o valor de a:");
7.      a = System_in.readFloat();
8.      System.out.print("Digite o valor de b:");
9.      b = System_in.readFloat();
10.     System.out.print("Digite o valor de c:");
11.     c = System_in.readFloat();
12.     delta = ((b*b)-(4*a*c));
13.     if (delta >= 0) {
14.       x1 = ( ( -b + (Math.sqrt (delta) ) ) / ( 2*a ) );
15.       x2 = ( ( -b + (Math.sqrt (delta) ) ) / ( 2*a ) );
16.       System.out.println("x1 = " + x1);
17.       System.out.println("x2 = " + x2);
18.     } else
19.       System.out.println("Equação não possui raízes reais!");
20.   }
21. }
```

Figura 4.21 – Código Java para cálculo das raízes de uma equação de 2o grau com teste de condição simples `if` .

Um outro exemplo do uso da estrutura condicional `if` em Java está ilustrado na figura 4.22. O objetivo do programa, de propósitos puramente didáticos, é identificar se um determinado número aleatório gerado possui valor par ou ímpar.

```
1.  public class Aleatorio {
2.     public static void main(String[] args) {
3.        double numgerado;
4.        int num;
5.        numgerado = Math.random()*10;
6.        num = (int)numgerado;
7.        if (num%2 ==0)
8.           System.out.println("Saiu um número par: "+num);
9.        else
10.          System.out.println("Saiu um número ímpar: "+num);
11.    }
12. }
```

Figura 4.22 – Código Java que testa se um número gerado aleatoriamente possui valor par ou ímpar.

A instrução `random()` da classe `Math`, disponível por Java, gera números aleatórios decimais compreendidos entre 0 e 1. Dessa forma, pode-se notar facilmente que o código da linha 5 armazena na variável `numgerado` valores reais entre 0 e 10. No entanto, estamos interessados em valores inteiros. A linha 6 utiliza um recurso interessante da linguagem, que permite a conversão de valores de um tipo em outros tipos. Esse tipo de conversão forçada é chamada de ***type casting***. No exemplo, fazemos uso do type casting para converter um valor `double` para inteiro, adicionando o tipo para o qual queremos converter entre parênteses (`int`) imediatamente antes da variável ou expressão que representa o valor do tipo de origem (`numgerado`). Na conversão de um tipo de maior capacidade para um tipo de menor capacidade, naturalmente, há perda de informação; de `double` para `int`, em particular, o valor é truncado (apenas a parte inteira é considerada). Caso `numgerado` seja 1,9855, por exemplo, será convertido para 1. Finalmente, na linha 7 utilizamos o operador aritmético `%` de Java para extrair o resto da divisão do número por 2; se este valor for igual a 0 (zero), o número é par, caso contrário, é ímpar.

A figura 4.23 traz um exemplo Java ilustrativo da estrutura condicional de múltipla escolha para definir que tipo de operação irá executar sobre um determinado valor lido.

```
1.  public class Operacoes {
2.     public static void main(String[] args){
3.           double x, z = 0;
4.           int op;
5.           System.out.println("Digite um valor:");
6.           x = System_in.readFloat();
7.           System.out.println("Escolha a opção:\n" +
8.                     "\t1 - Raiz quadrada\n" +
9.                     "\t2 - Cubo\n" +
10.                    "\t3 - Seno\n" +
11.                    "\t4 - Cosseno");
12.          op = System_in.readInt();
13.          switch (op) {
14.          case 1: z = Math.sqrt(x); break;
15.          case 2: z = Math.pow(x, 3); break;
16.          case 3: z = Math.sin(x); break;
17.          case 4: z = Math.cos(x);
18.          }
19.          System.out.print("Resultado = " + z);
20.    }
21. }
```

Figura 4.23 – Exemplo Java de uso da estrutura de múltipla escolha `switch`.

As operações possíveis são informadas ao usuário através do comando de escrita da linha 7. O usuário então escolhe uma das quatro opções e o programa realiza a operação respectiva avaliando a escolha do usuário através da estrutura Java de múltipla escolha `switch` (linhas 13 à 18).

O comando **`break`** de Java é um comando de salto que força a saída imediata da estrutura mais interna. No exemplo (linhas 14, 15 e 16), ele é utilizado para que após uma das quatro opções ser executada, o controle de execução do programa saia imediatamente da estrutura `switch`, evitando os demais testes.

4.4.2 Estruturas de repetição em Java

O pseudocódigo da figura 4.13 ilustrou o uso da estrutura de repetição enquanto para o cálculo das médias aritméticas finais dos alunos de uma turma. A figura 4.24 mostra a codificação Java equivalente para esse mesmo exemplo.

```
1.  public class MediaTurma {
2.     public static void main(String[] args) {
3.            float nota1, nota2, media;
4.            int numalunos;
5.            String nome;
6.            numalunos = System_in.readInt();
7.            while (numalunos > 0) {
8.               nome =  System_in.readString();
9.               nota1 = System_in.readFloat();
10.              nota2 = System_in.readFloat();
11.              media = (nota1 + nota2)/2;
12.              if (media >= 5)
13.                 System.out.println("O aluno "+nome+
14.                                    " está aprovado");
15.              else
16.                 System.out.println("O aluno "+nome+
17.                                     " está reprovado");
18.              numalunos--;
19.           }
20.    }
21. }
```

Figura 4.24 – Código Java com estrutura de repetição `while`.

Em Java, um laço `enquanto` é implementado pela estrutura *while (condições) {...}*.

No código em questão, existe apenas uma condição, que é o número de alunos ser maior que 0 (zero) (linha7). Observe o comando `numalunos--` na linha 18; esse comando é uma versão simplificada equivalente ao o comando de atribuição `numalunos = numalunos - 1`. Como o incremento e decremento de uma unidade para variáveis inteiras possui um uso bastante corriqueiro em programas, sobretudo em relação a contadores para estruturas de repetição, a linguagem Java

possui sintaxe especifica para esses usos: `var++` ou `++var` para incremento do valor de var e `var--` ou `--var`, para decremento.

Um problema clássico de programação é o cálculo da seqüência de Fibonacci. A seqüência de Fibonacci é iniciada por dois valores, 0 e 1, e o termo subseqüente é sempre igual à soma dos dois termos anteriores: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

O código Java da figura 4.25 imprime os n primeiros termos da seqüência de Fibonacci, onde n é fornecido pelo usuário.

```
1.  public class Fibonacci {
2.     public static void main(String[] args){
3.            System.out.print("Número de termos da seqüência: ");
4.            int num = System_in.readInt();
5.            int p = 0, u = 1, t, cont = 2;
6.            System.out.print("Seqüência: "+p+" "+u+" ");
7.            do {
8.               t = u + p;
9.               System.out.print(t+" ");
10.              p = u;
11.              u = t;
12.              cont++;
13.           } while (cont < num);
14.    }
15. }
```

Figura 4.25 – Cálculo da seqüência de Fibonacci com a estrutura de repetição `do-while` de Java.

Esse programa faz uso da estrutura de repetição repita. Em Java ela é implementada pela estrutura *do {...} while (condições)*, que pode ser lida como *faça algo enquanto a(s) condição(ões) for verdadeira*. De fato, no exemplo, o primeiro termo da seqüência que é gerado a partir da soma dos anteriores (`t = u + p`, linha 8) é necessariamente impresso (linha 9). Dessa forma, independentemente do número de termos especificados pelo usuário, este programa sempre gerará uma seqüência de no mínimo três termos: 0 1 1.

Finalmente, o programa da figura 4.26 ilustra a versão em Java para o problema do cálculo do fatorial de um número.

```
1.  public class Fatorial {
2.     public static void main(String[] args){
3.            int num, fat = 1;
4.            System.out.print("Digite um valor: ");
5.            num = System_in.readInt();
6.            if (num < 0)
7.               System.out.println("Erro - número negativo!");
8.            else
9.               if (num == 0)
10.                 System.out.println("0! = 1");
11.              else
```

```
12.                 if (num > 0) {
13.                   for (int i = 1; i <= num; i++)
14.                      fat = fat * i;
15.                   System.out.println(num+"! = "+fat);
16.                 }
17.     }
18. }
```

Figura 4.25 – Cálculo do fatorial em Java com estrutura de repetição `for`.

O número a se calcular o fatorial é solicitado ao usuário (linhas 4 e 5). Caso o usuário forneça um valor negativo, o programa acusa um erro e é encerrado (linhas 6 e 7). Caso o usuário forneça o valor 0 (zero), uma mensagem padrão é impressa uma vez que 0! = 1, por definição. Caso o usuário forneça um valor inteiro positivo, o programa utiliza uma estrutura com variável de controle para realizar as multiplicações sucessivas necessárias para o cálculo do fatorial. Em Java, a estrutura *para* é implementada com a sintaxe

for (varcontrole = valorinicio; condição; atualização de varcontrole) {....}

Podemos dizer que a estrutura *for* de Java é apenas uma maneira mais compacta de implementarmos um estrutura *while*. A estrutura acima, por exemplo, é absolutamente equivalente à estrutura *while* abaixo.

```
...
varcontrole = valorinicio;
while (condição) {
        ....
        atualização da varcontrole;
}
```

No exemplo, fazemos `i` variar de `1` até `num`, atualizamos o valor de `fat` e incrementamos `i` a cada repetição.

Resumo

- Um algoritmo para solução de um problema segue uma ordem (ou fluxo) de execução de ações, que no geral, é seqüencial.
- Uma estrutura condicional permite que blocos de ações sejam ou não executados dependendo do resultado de algum teste lógico.
- As estruturas condicionais podem ser simples, aninhadas ou de múltipla escolha: as simples possuem apenas um bloco se-então-senão, as aninhadas possuem vários blocos de decisão simples encadeados, e as estruturas de múltipla escolha permitem que ações diferentes sejam executadas a partir de valores

diferentes.

- Um estrutura de repetição permite que um mesmo bloco de ação seja executado repetidamente.
- Existem três tipos de estruturas de repetição: estrutura com teste no início, estrutura com teste no final, e estrutura de repetição com controle de variável.
- Java implementa todos os tipos de estrutura de controle citados.
- O comando `if-then-else` de Java permite implementação de estruturas condicionais simples e aninhadas.
- O comando `switch` permite a implementação da estrutura condicional de múltipla escolha em Java.
- O comando `while` de Java representa um laço com teste de condição no início.
- O comando `do-while` permite laço de repetição com teste de condição ao final.
- O comando `for` permite repetição de um bloco de ações um número pré-determinado de vezes em Java.

Na próxima aula...

… veremos como o programador pode definir seus próprios tipos de dados para resolver problemas que requerem um grande número de variáveis ou que lidam com registros de dados.

Referências e Sugestões de Leitura

Estruturas de controle em pseudocódigos são discutidas no capítulo 4 de

LEITE, M., Técnicas de Programação: uma Abordagem Moderna, BRASPORT, 2006.

e no capítulo 3 de

FORBELLONE, A., EBERSPÄCHER, H. Lógica de Programação. Prentice Hall, 3ed, 2005.

DEITEL, H., DEITEL, P. Java como Programar, Prentice-Hall, 2005.

explora as Estruturas Condicionais em Java no capítulo 4 e as Estruturas de Repetição em java no capítulo 5

Exercícios Propostos

(PSEUDOCÓDIGO) Para cada um dos problemas a seguir crie um algoritmo para resolvê-lo utilizando a sintaxe de pseudocódigo.

(Algoritmos com estruturas de decisão)

4.1 – Entrar via teclado, com três valores distintos. Exibir o maior e o menor deles ou uma mensagem avisando que os números são idênticos.

4.2 – Calcular e exibir a área de um retângulo, a partir dos valores da base e altura que serão digitados. Se a área for maior que 100, exibir a mensagem "Terreno grande", caso contrário, exibir a mensagem "Terreno pequeno".

4.3 – Entrar com o peso (em kg) e a altura (em metros) de uma determinada pessoa. Após a digitação, exibir se esta pessoa está ou não com seu peso ideal. Veja relação peso/altura² (R) abaixo:

R < 20 → Abaixo do peso

20 <= R <= 25 → Peso ideal

R >= 25 → Acima do peso

4.4 – A partir de três valores que serão digitados, verificar se formam ou não um triângulo. Em caso positivo, exibir sua classificação: "Isósceles, escaleno ou eqüilátero". Um triângulo escaleno possui todos os lados diferentes, o isósceles, dois lados iguais e o eqüilátero, todos os lados iguais. Para existir triângulo é necessário que a soma de dois lados quaisquer seja maior que o outro, isto, para os três lados.

4.5 – Verificar se três valores quaisquer (A,B, C) que serão digitados formam ou não um triângulo retângulo. Lembre-se que o quadrado da hipotenusa é igual a soma dos quadrados dos catetos.

4.6 – A partir dos valores da aceleração (a em m/s²), da velocidade inicial (v0 em m/s) e do tempo de percurso (t em s). Calcular e exibir a velocidade final de automóvel em km/h. Lembre-se que a fórmula para o cálculo da velocidade é V = V0 + a*t. Exibir mensagem de acordo com a tabela:

V <= 40 → Veículo muito lento

40 <= V < 60 → Velocidade permitida

60 <= V < 80 → Velocidade de cruzeiro

80 <= V < 120 → Veículo rápido

V > 120 → Veículo muito rápido

4.7 – Uma escola com cursos em regime semestral realiza duas avaliações durante o semestre e calcula a média do aluno, da seguinte maneira: MEDIA = (P1 +

2P2) / 3

Fazer um programa para entrar via teclado com o valor da primeira nota (P1) e o programa deverá calcular e exibir quanto o aluno precisa tirar na segunda nota (P2) para ser aprovado, sabendo que a média de aprovação é igual a cinco.

4.8 – Entrar via teclado com dois valores quaisquer. Após a digitação, exibir um seletor de opções ("menu") com as seguintes opções:

1 – Multiplicação

2 – Adição

3 – Divisão

4 – Subtração

5 – Fim de processo

Solicitar uma opção por parte do usuário, verificar se é ou não uma opção válida (letras ou números) e processar a respectiva operação. Enviar mensagem de erro se a opção escolhida não existir no seletor. Encerrar o programa somente quando o usuário escolher a opção de finalização. Repare que a opção de número três é de divisão e o programa deverá enviar mensagem de erro, (somente nesta opção) se o denominador for zero.

(Algoritmos com estruturas de decisão e repetição)

4.9 – Criar uma rotina de entrada que aceite somente um valor positivo.

4.10 – Entrar via teclado com o sexo de determinado usuário, aceitar somente "F" ou "M" como respostas válidas.

4.11 – Exibir a tabuada do número cinco no intervalo de um a dez.

4.12 – Exibir a soma dos números inteiros positivos do intervalo de um a cem.

4.13 – Exibir os vinte primeiros valores da série de Bergamaschi. A série: 1, 1, 1, 3, 5, 9, 17, ...

4.14 – Calcular e exibir a soma dos "N" primeiros valores da seqüência 1/2, 2/3, 3/4, 4/5, ... O valor "N" será digitado, deverá ser positivo, mas menor que cinqüenta. Caso o valor não satisfaça a restrição, enviar mensagem de erro e solicitar o valor novamente.

4.15 – Entrar via teclado com "N" valores quaisquer. O valor "N" (que representa a quantidade de números) será digitado, deverá ser positivo, mas menor que vinte. Caso a quantidade não satisfaça a restrição, enviar mensagem de erro e solicitar o valor novamente. Após a digitação dos "N" valores, exibir:

a) O maior valor;

b) O menor valor;

c) A soma dos valores;

d) A média aritmética dos valores;

e) A porcentagem de valores que são positivos;

f) A porcentagem de valores negativos;

Após exibir os dados, perguntar ao usuário de deseja ou não uma nova execução do programa. Consistir a resposta no sentido de aceitar somente “S” ou “N” e encerrar o programa em função dessa resposta.

(JAVA) Escreva programas em Java para solucionar os problemas a seguir.

(Estrutura de controle seqüencial)

4.16 – Determine qual é a idade que o usuário faz no ano atual. Para isso solicite ao usuário o seu ano de nascimento e o ano atual.

4.17 – Solicite a quantidade de homens e de mulheres de uma turma qualquer da faculdade. Em seguida calcule e exiba o percentual (separadamente) de homens e mulheres desta turma.

4.18 – Calcule o valor final de uma dívida. Para isto pergunte ao usuário o valor inicial do débito, a quantidade de meses e os juros mensais. Use o calculo de juros simples.

(Estruturas de controle condicionais)

4.19 – A partir de 3 notas fornecidas de um aluno, informe se ele foi aprovado, ficou de recuperação ou foi reprovado. A média de aprovação é >= 7.0; a média de recuperação é >= 5.0 e < 7.0; e a média do reprovado é < 5.0.

4.20 – Verifique a validade de uma data de aniversário (solicite apenas o número do dia e do mês). Além de falar se a data está ok, informe também o nome do mês. Dica: meses com 30 dias: abril, junho, setembro e novembro.

4.21 – Acrescente no exercício anterior a apresentação do signo do horóscopo da pessoa.

4.22 – Solicite o nome e a idade de duas pessoas. Em seguida exiba o nome da pessoa mais velha e o nome da pessoa mais nova.

4.23 – Exiba o valor do empréstimo possível para um funcionário de uma empresa, cujo cargo e valor do salário é fornecido pelo usuário.

Sabe-se que:

Cargo	% do salário
Diretoria	30%
Gerência	25%

Operacional	20%

4.24 – Faça a verificação da validade de uma data completa (dia, mês e ano). Obs. um ano é bissexto, cujo mês de fevereiro possui 29 dias, se o resto da divisão do ano por 4 e também por 100 for zero, ou ainda se o resto da divisão por 400 for zero. Os meses com 30 dias são 4, 6, 9 e 11, os demais tem 31 dias.

4.25 – Receba 2 horários e exiba a diferença entre eles em segundos. A entrada destes horários pode ocorrer em qualquer ordem.
Dica: transforme os dois horários para segundos.

4.26 – Coloque em ordem crescente três números quaisquer. Como desafio, tente depois fazer uma solução com apenas 3 estruturas de decisão.

4.27 – Receba dois retângulos através dos seus quatro vértices. Cada vértice é um ponto e é formado por duas coordenadas x e y. Faça a crítica destes pontos, pois eles não podem se sobrepor. Observe que o usuário pode fornecê-los em qualquer ordem. Em seguida informe se os dois retângulos se interceptam em algum lugar.

(Estruturas de repetição)

4.28 – Exiba todos os números pares de 10 a 200.

4.29 – Exiba 50 números sorteados de 1 a 100 para o usuário.

4.30 – Exiba uma quantidade de números sorteados determinada pelo usuário. Peça também que o usuário determine a faixa do sorteio.

4.31 – Exiba todos os números ímpares existentes entre dois números informados pelo usuário. Dica: use o operador % para calcular o resto da divisão entre dois números.

4.32 – Solicite ao usuário a idade de cada um componente de um grupo de pessoas. A quantidade de pessoas também será determinada por ele. Após o término da entrada, apresente:

a) a média das idades,

b) a maior idade,

c) a menor idade,

d) a quantidade de pessoas maior de idade.

4.33 – Crie um jogo para o usuário descobrir um número sorteado de 1 a 100. A cada tentativa dele, forneça uma dica falando se o número é maior ou menor. Quando ele descobrir exiba uma mensagem de parabéns e mostre em quantas tentativas ele conseguiu.

4.34 – Aproveitando o algoritmo do exercício anterior, exiba uma lista de 40 grupos de números sorteados de 0 a 59. Cada grupo possui 3 números e deve

exibido em ordem crescente.

4.35 – Determine o maior valor de uma lista de 100 números fornecidos pelo usuário.

3.36 – Leia uma relação de pacientes de uma clínica, cada um com o nome, o sexo, o peso, a idade e a altura. Para sinalizar o fim da lista será fornecido "fim" no nome do último paciente. Exiba um relatório contendo:

a) a quantidade de pacientes.

b) a média de idade dos homens.

c) a quantidade de mulheres com altura entre 1,60 e 1,70 e peso acima de 70kg.

d) a quantidade de pessoas com idade entre 18 e 25.

e) o nome do paciente mais velho.

f) o nome da mulher mais baixa.

4.37 – A história do rei que e tornou pobre: após perder uma aposta com um súdito, ele teve que pagar uma quantia muito grande em sacos de arroz. A aposta feita anteriormente era que se o súdito ganhasse o rei teria que pagar um grão de arroz colocado na primeira casa de um tabuleiro de xadrez. Na segunda casa teria que pagar o dobro, ou seja, dois grãos de arroz, e assim sucessivamente até a casa número 64. Exiba quantos grãos de arroz este súdito teria que ganhar, somando todas as 64 casas. Depois tente exibir a quantia de sacos de arroz.

4.38 - Exiba a tabuada de um número fornecido pelo usuário. Por exemplo se ele digitar o número 5, então será mostrado:

```
5    x    1    =    5
5    x    2    =    10
5    x    3    =    15
5    x    4    =    20
5    x    5    =    25
5    x    6    =    30
5    x    7    =    35
5    x    8    =    40
5    x    9    =    45
5    x    10   =    50
```

4.39 – A operadora de celular Vai-Vai possui um plano com o valor mensal de 50,00 que permite 100 minutos por mês para qualquer número. Além disso, ela oferece 50 minutos a mais para ligações destinadas a um número da própria Vai-Vai. Ainda neste plano ela tem uma promoção onde cada minuto gasto para telefone fixo consome somente a metade. O valor do minuto excedente para outras operadoras é de 0.65, e para a própria Vai-Vai é 0.20. Faça um programa que permita ao usuário entrar com o tipo de ligação ('o' = outras operadoras, 'v' = a

própria Vai-Vai, ou ‘f’ = telefone fixo) e a quantidade de minutos. A cada entrada, deve-se informar o quanto que ele tem de saldo e o valor a pagar. Faça isto enquanto ele indicar que existem mais ligações a serem digitadas.

4.40 – Crie um programa para simular uma urna de votação para exatamente 3 candidatos. Logo no início deve-se perguntar ao usuário os nomes dos candidatos. Permita votos em branco. Ao término de toda a entrada, apresente o nome, a quantidade de votos e o percentual de cada candidato. Apresente também a quantidade e o percentual dos votos em branco e quem foi o ganhador da eleição.

4.41 – Leia uma data e uma quantidade de dias, em seguida exiba esta data somada pela quantidade de dias fornecida. Exemplo: 29/04/2007 + 3 = 02/05/2007.

(Estruturas de repetição aninhadas)

3.42 – Exiba as 10 tabuadas (de 1 a 10). Dica: use dois comandos while aninhados, ou seja um dentro do outro.

4.43 – Desenhe a seguinte pirâmide de asteriscos. O usuário determina a quantidade de linhas.

```
*
**
***
****
*****
******
*******
********
*********
**********
***********
************
...
```

4.44 – Desenhe a seguinte pirâmide de números. O usuário determina a quantidade de linhas.

01

01 02

01 02 03

01 02 03 04

01 02 03 04 05

01 02 03 04 05 06

01 02 03 04 05 06 07

01 02 03 04 05 06 07 08

…

4.45 – Desenhe a seguinte pirâmide de asteriscos. O usuário determina a

quantidade de linhas.