Lógica de Programação

Apostila de Lógica de Programação

Sumário

1 - Introdução

Para se construir um programa é necessário o domínio da linguagem de programação a ser usada (sua sintaxe e semântica), mas é necessário, principalmente, o domínio de uma lógica de programação que permita ao programador construir um programa eficiente e fácil de ser usado pelo operador e/ou usuário do mesmo. A *lógica de programação* é a técnica de encadear pensamentos para atingir determinado objetivo.

Esta lógica de programação pode ser ensinada a um aluno-programador juntamente com uma linguagem de programação. Por exemplo, o estudante pode aprender técnicas de programação e ao mesmo tempo usá-las em programas em C++, Pascal ou alguma outra linguagem, paralelamente. Outrossim, o aluno poderá aprender estas técnicas primeiramente, escrevendo seus programas em português mesmo (ou fluxograma), para depois aprender uma linguagem de programação.

Quando estamos escrevendo um programa, seja em qualquer linguagem de programação ou mesmo através de uma representação gráfica ou em português, estamos elaborando um *algoritmo*. Um algoritmo é, portanto, uma seqüência de instruções (passos ou etapas) ordenadas de forma lógica e bem definida para a resolução de uma determinada tarefa ou problema.

Estruturas de dados e algoritmos estão intimamente ligados. Não é possível estudar estruturas de dados sem considerar os algoritmos associados a elas, assim como a escolha dos algoritmos em geral depende da representação e da estrutura dos dados. As estruturas de dados são representações usadas nos programas para que possamos trabalhar com conjunto de dados de uma maneira mais eficiente. São exemplos de estruturas de dados: vetores, matrizes, registros, listas, árvores, etc.

Programar é, basicamente, estruturar dados e construir algoritmos. Os programas são formulações concretas de algoritmos abstratos, baseados em representações e estruturas específicas de dados. Uma linguagem de programação é uma técnica de notação para programar, com a intenção de servir de veículo tanto para a expressão do raciocínio quanto para a execução automática de um algoritmo por um computador.

2 - Algoritmos

Como mencionado anteriormente, um algoritmo é uma seqüência de instruções ordenadas de forma lógica e bem definida para a resolução de uma determinada tarefa ou problema. Como exemplos de tarefas, podemos citar: sacar dinheiro em um caixa eletrônico ou calcular a média de três números.

Até mesmo tarefas simples podem ser descritas como um seqüência lógica de instruções. Por exemplo:

Algoritmo para comer um chocolate

1. Pegar o chocolate
2. Retirar a embalagem
3. Comer o chocolate
4. Jogar a embalagem no lixo

Algoritmo para somar dois números

1. Escrever o primeiro número no retângulo A
2. Escrever o segundo número no retângulo B
3. Somar o número do retângulo A com o número do retângulo B e armazenar o resultado no retângulo C

Retângulo A Retângulo B Retângulo C

Um algoritmo é a definição do que se deve fazer para resolver a tarefa. O como fazer é a implementação do algoritmo em alguma linguagem de programação, quando estamos tratando dos algoritmos que podem gerar um programa de computador.

Técnicas para Escrever Algoritmos

Há várias maneiras de se escrever um algoritmo, antes de passá-lo para uma linguagem de programação como, por exemplo, linguagem natural (português), Portugol, Fluxograma e Diagrama de Chapin.

Linguagem Natural

Um algoritmo pode ser escrito como uma seqüência de passos em linguagem natural (português, por exemplo). Os algoritmos listados anteriormente foram descritos com essa técnica.

Exemplo:

1. Se A é maior que B
    a. Então faça C receber o valor de A
    b. Senão faça C receber o valor de B

Portugol

É uma técnica de se escrever programas em português, utilizando estruturas de dados. Você escreve o algoritmo e depois transcreve o que foi escrito para uma linguagem de programação (este processo de "tradução" do Portugol para uma linguagem chama-se mapeamento do programa). O Portugol é baseado nas linguagens de programação Pascal e Algol. Escrever um programa em Portugol é quase que como escrever a tradução de um programa em linguagem Pascal (onde os comandos são em inglês).

Exemplo:

```
se A > B
     então C <- A;
     senão C <- B;
fimse;
```

Fluxograma

Esta técnica consiste em escrever um programa utilizando símbolos gráficos, onde cada símbolo equivale a um comando de programa.

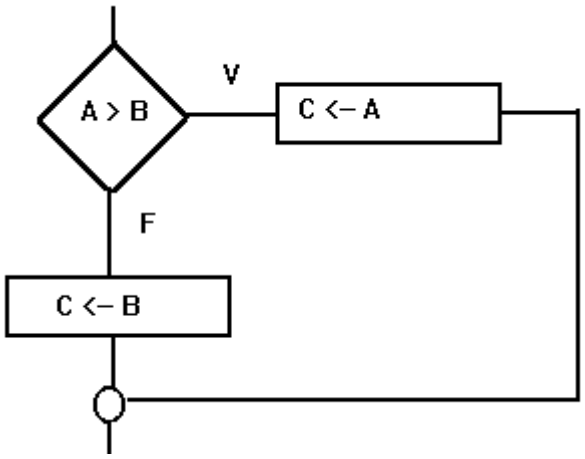
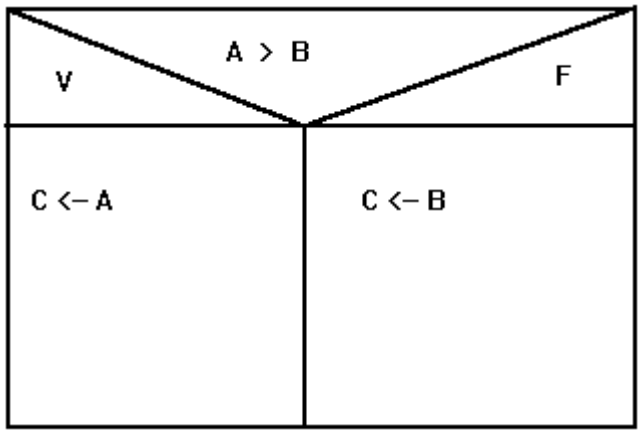


Diagrama de Chapin

O diagrama de CHAPIN, como o próprio nome indica, consiste em escrever o algoritmo dentro de um diagrama. As divisões deste diagrama simbolizam os comandos. É uma técnica ultrapassada, não mais utilizada por ser de interpretação complexa e por vezes confusa.



Regras para Escrever Algoritmos

Para escrever um algoritmo precisamos descrever a seqüência de instruções, de maneira simples e objetiva. Para isso utilizaremos algumas técnicas:

- Usar somente um verbo por frase
- Imaginar que você está desenvolvendo um algoritmo para pessoas que não trabalham com informática
- Usar frases curtas e simples
- Ser objetivo
- Procurar usar palavras que não tenham sentido dúbio

No desenvolvimento de um algoritmo, podemos dividir a tarefa (ou problema) sendo resolvida em três fases fundamentais:

- Entrada: dados de entrada do algoritmo.
- Processamento: passos realizados para se gerar a saída do algoritmo.
- Saída: dados resultantes da etapa de processamento.

Exemplo: calcular a média de três números N1, N2 e N3 quaisquer.

- Entrada: N1, N2 e N3.
- Processamento: somar os três números e dividir o resultado por três
  - (N1 + N2 + N3) / 3
- Saída: a média calculada no processamento.

Algoritmo

1. Receba o número N1
2. Receba o número N2
3. Receba o número N3
4. Some os três números e divida o resultado por três
5. Mostre o resultado da divisão.

Após desenvolver um algoritmo ele deverá sempre ser testado. Este teste é chamado de *teste de mesa* e significa seguir as instruções do algoritmo de maneira precisa para verificar se o procedimento utilizado está correto ou não.

Exemplo:

N1	**N2**	**N3**	**Soma**	**Média**
1	2	3	6	2
1	1	1	3	1

3 - Variáveis: Criação e Tipos

Variáveis

Sempre que estamos construindo programas, estamos envolvidos com dados, os quais devem ser tratados dentro dos programas de uma maneira especial. Como exemplo, podemos desenvolver um algoritmo para calcular a média de idade de 25 alunos de uma turma. Neste caso, os dados a serem manipulados são as idades dos mesmos. Sabemos que, para calcularmos uma média, necessitamos somar todas as idades e dividir pelo número de alunos. Neste caso, precisamos manipular estes dados (IDADE, SOMA, MEDIA) na memória do computador. Mas como o computador (ou melhor, o compilador da linguagem de programação utilizada) irá identificar qual valor corresponde à idade, à média ou à soma?

Nota-se, pois, que os valores a serem manipulados na memória do computador precisam ser identificados por nomes. Isto é uma variável.

Variável é, pois, uma posição de memória, identificada por um nome e que possui um valor dentro dela. Vejamos alguns exemplos através de "caixinhas" identificadas por nome, como se estas fossem as posições de memória.

IDADE	10

SALÁRIO	2500.52

NOME	Maria

No exemplo mostrado acima, vemos uma variável identificada (chamada) IDADE que possui o conteúdo igual a 10. Temos também a variável chamada SALÁRIO que possui o conteúdo igual a 2500.52 e a variável NOME com conteúdo igual a Maria. O nome de uma variável deve ser escrito em letras maiúsculas, podendo este nome conter números também, mas sempre começando com letra. Por exemplo: podemos ter variáveis chamadas de ENDEREÇO, TELEFONE, IDADE, TOTAL1, TOTAL5, mas NUNCA 1TOTAL, 5TOTAL, NOME-DO-FUNCIONARIO, porque não pode começar com número nem podemos utilizar símbolos no nome da variável, como -, *, /, $ ou outro símbolo qualquer.

Vimos, portanto, que uma variável deve receber um nome para que depois coloquemos nela um determinado conteúdo, que será usado no programa. No exemplo citado anteriormente, a respeito do programa para calcular média de idade da turma, podemos perceber que o mesmo precisaria de três variáveis: uma chamada IDADE, para armazenar a idade de cada aluno; outra chamada SOMA, para somar a IDADE de todos e outra identificada como MEDIA, que conterá a média de idade dos alunos. Mais tarde saberemos como construir este programa.

Constantes

O conceito de constante é similar ao de variável, na medida em que ambos possuem um identificador e um tipo. A diferença está no fato de que uma variável, como o próprio nome diz, pode ter seu conteúdo (valor) alterado durante a execução do programa. Já uma constante numa pode ser alterada. Uma vez que um dado valor é atribuído a uma constante, esta possuíra esse mesmo valor enquanto o programa estiver em execução.

Identificadores

Os identificadores são os nomes atribuídos a variáveis ou constantes de um programa. Deve-se buscar dar nomes claros às variáveis e constantes, de modo que elas reflitam os seus significados. Por exemplo, se uma variável irá armazenar o salário de um empregado, por que não escolher o identificador SALARIO para representa-la?

Os identificadores devem obedecer a duas regras de formação:

1) Devem começar com uma letra ou sublinhado (_).

2) Os caracteres seguintes devem ser letras, números ou sublinhado (_). Dito de outra forma, caracteres especiais, como **/*-+.%&^[{}]\!?'()** não são permitidos.

Tipos de Dados

Acabamos de ver que uma variável pode receber números, como no exemplo (10 e 2500.52) ou letras (MARIA). Em um programa, quando criamos uma variável na memória, é necessário determinarmos o tipo desta variável, ou seja, que tipo de conteúdo ela irá armazenar. Existem quatro tipos básicos de variáveis.

Portugol

- inteiro: uma variável do tipo inteiro, somente poderá armazenar valores inteiros, sem casas decimais, como por exemplo: -1, 5, -800, 0, 10.
- real: variáveis reais armazenam números reais e inteiros, como por exemplo -5, 3.14, 5.86, -4.6, 10. Note que o separador de casas decimais é o ***ponto*** e não a vírgula, como estamos normalmente acostumados a utilizar na matemática.
- caracter: variáveis deste tipo são para armazenar alfanuméricos (letras e números), como por exemplo, os valores: "Ana Maria", "a", "R. do Ouro", "123" etc.
- logico: variáveis deste tipo podem armazenar somente os valores VERDADEIRO (V) ou FALSO (F).

C++

- int: corresponde ao tipo inteiro de Portugol.
- float: corresponde ao real de Portugol
- string: corresponde ao tipo caracter de Portugol.
- bool: corresponde ao tipo logico de Portugol. Pode receber os valores *true* ou *false*.

Declaração de Variáveis

Portugol

Forma geral

```
tipo_de_dado: variaveis;
```

- Pode-se declarar uma única variável ou uma lista de variáveis separadas por vírgulas.
- Ao final da declaração, o ponto e vírgula é obrigatório.
- Os nomes de variáveis em Portugol **SEMPRE** são escritos em letras maiúsculas.

Exemplos:

- `inteiro: IDADE;`
- `real: PESO, ALTURA;`
- `caracter: NOME, SEXO;`
- `logico: POSSUI_ESTOQUE;`

C++

Forma geral

```
tipo_de_dado variaveis;
```

- Pode-se declarar uma única variável ou uma lista de variáveis separadas por vírgulas.
- Ao final da declaração, o ponto e vírgula é obrigatório.
- Os nomes de variáveis em C++ não precisam são escritos em letras maiúsculas.

Exemplos

- `int idade;`
- `float peso, altura;`
- `string nome, sexo;`
- `bool possui_estoque;`

4 – Estruturas Básicas de Controle

Operações Matemáticas Elementares

As operações mais básicas existentes são operações fundamentais (adição, subtração, multiplicação e divisão). Tanto em Portugol quanto em C++ elas são representadas pelos mesmos operados, listados na tabela a seguir:

Operação	Símbolo	Exemplo
Adição	+	1 + 2
Subtração	-	5 - 3
Multiplicação	*	-1 * 2
Divisão	/	10 / 4

Em uma expressão aritmética, assim como na matemática, primeiro será executada a multiplicação ou a divisão e, em seguida, a adição ou subtração. Para se alterar a prioridade das operações, deve-se utilizar parênteses.

Exemplo:

- 2 + 5 * 3 = 2 + 15 = 17
- (2 + 5) * 3 = 7 * 3 = 21

Início e Fim

São as estruturas que delimitam um programa, indicando o seu começo e o seu término.

Portugol

```
início
.
.
.
fim.
```

C++

```
#include <iostream.h>
using namespace std;
int main() {
 .
 .
 .
 return valor_do_tipo_inteiro;
}
```

Comando de Atribuição

É o comando utilizado para atribuir um valor a uma variável. O valor atribuído à variável deve ser do tipo de dado da variável. Em outras palavras, uma variável inteira deve receber valores inteiros, uma variável caracter deve receber cadeias de caracteres e assim por diante. Em Portugol, o comando de atribuição é representado por uma seta invertida (<-) e em C++, pelo sinal de igualdade (=).

Exemplo: atribuir o valor 30 à variável idade.

- Portugol (IDADE <- 30;
- C++ (idade = 30;

Em um algoritmo, o comando de atribuição faz parte da fase de processamento do problema.

Com os comandos apresentados, é possível construir um programa que calcule a média entre 10, 50 e 60.

- Entrada: nenhuma.
- Processamento: atribuir cada valor a uma variável, somar as variáveis e em seguida calcular a média.
- Saída: o valor calculado para a média.

Algoritmo:

1. Declarar as variáveis necessárias
2. Atribuir o primeiro valor à primeira variável
3. Atribuir o segundo valor à segunda variável
4. Atribuir o terceiro valor à terceira variável
5. Somar as três variáveis e armazenar o resultado na variável soma
6. Armazenar na variável média o resultado da soma dividido por três

Portugol

```
início
        inteiro: A, B, C, SOMA;
        real: MEDIA;

      A <- 10;
      B <- 50;
      C <- 60;
      SOMA <- A + B + C;
      MEDIA <- SOMA/3;
fim.
```

C++

```
#include <iostream.h>
using namespace std;
int main() {
```

```
  int a, b, c, soma;
  float media;
  a = 10;
  b = 50;
  c = 60;
  soma = a + b + c;
  media = soma / 3;
  return 0;
}
```

Comandos de Entrada e Saída

Reveja o programa exemplo anterior. Ele calcula a média, mas não exibe o resultado. Para mostrarmos o conteúdo de uma variável, constante ou de mensagens de texto fixo, temos que utilizar um comando de saída, que envia dados para o mundo exterior ao programa, geralmente para o monitor do computador. Este comando permite ao usuário ver a saída do processamento do programa.

Em Portugol, o comando de saída é o imprima e em C++, o *cout*.

Exemplos:

- `imprima ("A média é :", MEDIA);`
- `cout << "A média é: " << media;`

Os valores que estão entre aspas duplas são mensagens de texto fixo; quando o comando de saída for executado, ele irá exibir o texto da mensagem da mesma forma que este foi escrito. O que não estiver entre aspas duplas serão variáveis e constantes. Para estas, o comando de impressão exibirá o conteúdo armazenado. Por exemplo, considere que a variável média possua o valor 15. Se os comandos de saída acima fossem executados, eles exibiriam a mensagem: ***A média é 15***.

O comando de saída pode escrever vários dados de uma única vez, bastando para isso que estes dados sejam concatenados. A concatenação em Portugol é feita com a vírgula (**,**) e em C++, com o operador **<<**.

Para que o programa anterior mostre o valor da média, o programa deve ser acrescido do comando de impressão, ficando como abaixo:

Portugol

```
início
     inteiro: A, B, C, SOMA;
     real: MEDIA;

     A <- 10;
     B <- 50;
     C <- 60;
     SOMA <- A + B + C;
     MEDIA <- SOMA/3;
```

```
     imprima ("A média é:", MEDIA);
fim.
```

C++

```
#include <iostream.h>
using namespace std;
int main() {
  int a, b, c, soma;
  float media;
  a = 10;
  b = 50;
  c = 60;
  soma = a + b + c;
  media = soma / 3;
  cout << "A média é: " << media;
  return 0;
}
```

Para calcular a média entre 10, 5 e 15 basta alterar no programa os valores, ou seja, onde está 10, continua o mesmo valor, mas substituiremos 50 por 5 e 60 por 15. Assim sendo, o programa está pronto para ser executado novamente. Mas quem alterou os dados foi o programador e não o usuário. Imagine o trabalho que seria toda vez que o usuário do programa precisasse de executá-lo para valores diferentes.

Para se evitar isto, o ideal seria que o próprio usuário digitasse os valores na execução do programa. Neste caso, ao invés de atribuir valores às variáveis no corpo do programa pelo programador, estes valores serão atribuídos a elas de acordo com a digitação do usuário.

Para se fazer isso, lançamos mão do comando de entrada (leitura de dados). Quando um comando de leitura é encontrado no programa, o mesmo pára a execução e fica à espera da digitação do usuário e o que o for digitado será atribuído à variável que estiver no comando.

Em Portugol, o comando de entrada é o leia e em C++, o *cin*.

Exemplos:

- `leia (VALOR1, VALOR2);`
- `cin >> valor1, valor2;`

O comando de entrada recebe dados do mundo exterior para dentro do programa. Estes dados só podem ser armazenados em variáveis. Por essa razão, o comando de entrada só opera sobre variáveis, diferentemente do comando de saída, que pode trabalhar com variáveis, constantes e mensagens de texto fixo. Dessa forma, **NUNCA** haverá aspas duplas em comandos de entrada, mas apenas variáveis.

O comando de entrada pode ler várias variáveis de uma única vez, bastando para isso que estas sejam concatenadas. A concatenação em Portugol é feita com a vírgula (`,`) e em C++, com o operador **>>**.

Vejamos um exemplo para entendermos melhor:

Portugol

```
início

     real: VALOR1, VALOR2, VALOR3, SOMA, MEDIA;

     imprima (“Primeiro valor: ”);
     leia (VALOR1);

     imprima (“Segundo e terceiro valores: ”);
     leia (VALOR2, VALOR3);

     SOMA <- VALOR1 + VALOR2 + VALOR3;
     MEDIA <- SOMA/3;
     imprima ("Media = " , MEDIA, " Soma = " , SOMA);
fim.
```

C++

```
#include <iostream.h>
using namespace std;
int main() {
  float valor1, valor2, valor3, soma, media;

  cout << “Primeiro valor: ”:
  cin >> valor1;

  cout << “Segundo e terceiro valores: ”:
  cin >> valor2 >> valor3;

  soma = valor1 + valor2 + valor3;
  media = soma / 3;
  cout << “Media = ” << media << “ Soma = ” << SOMA;

  return 0;
}
```

5 – Outros Elementos Importantes

Comentários

Comentários são textos em linguagem natural (português, por exemplo) acrescentados aos programas para aumentar a compreensão e a legibilidade do mesmo.

Os comentários podem ser inseridos em qualquer ponto do programa e devem ser utilizados sempre que possível. A utilização de comentários não aumenta o tamanho do programa, nem o fazem ficar mais lento, pois eles não são transformados em instruções no momento de compilação do programa.

Como linhas comentadas são ignoradas pelo compilador, o comentário é útil não apenas para documentação de código, como também para depuração (debug), na medida em que é possível "comentar" certas partes do código para que elas não executem em determinado momento.

Os comentários devem ser inseridos para identificar o que está sendo feito no programa, mas não precisam parafrasear o código, pois nesse caso eles perdem sua utilidade. Por exemplo, suponha que a linha de código abaixo (em Portugol), calcule a nota final de um aluno:

```
nf <- n1 + n2;
```

Um bom exemplo de comentário para esta linha seria "cálculo da nota final" e um mau exemplo seria: "soma a variável n1 com a variável n2 e armazena o resultado em nf".

Uma boa utilização de comentários é para descrever o significado das variáveis do programa, no momento em que elas são declaradas, principalmente quando o nome da variável não permite a identificação imediata do seu significado.

Para se inserir um comentário em um programa em Portugol, basta que o texto a ser comentado esteja entre chaves: {comentário}.

Exemplo:
`nf <- n1 + n2;` **{Cálculo da nota final do aluno}**

Em C++, existem dois tipos de comentários: os de linha e os de bloco. Os comentários de linha servem para comentar linhas individuais dentro do programa. Os comentários de bloco servem para criar comentários que possuam mais de uma linha.

Os comentários de linha são criados inserindo-se os caracteres "//" na frente do texto a ser comentando. Os comentários de bloco iniciam-se com os caracteres "/*" e terminam com os caracteres "*/".

Exemplos:
`nf = n1 + n2;` **//Cálculo da nota final do aluno**

```
/* Exemplo de um
```

```
comentário
de várias linhas */
```

Identação

A identação (também chamada de endentação, edentação ou indentação) é um recurso aplicado a um programa para indicar a disposição hierárquica dos elementos (comandos) do código fonte.

Na maioria das linguagens, a identação possui um papel meramente estético, e um programa não identado funciona do mesmo jeito que um identado. Porém, a identação favorece a legibilidade de um programa, o que facilita a sua manutenção. Ou seja, identar, assim, comentar um programa, é uma boa prática de programação.

Um programa é sempre composto por um ou mais blocos de comando. Os programas mais simples, com apenas comandos de entrada e saída, possuem apenas o bloco principal. Programas mais complexos, com várias estruturas de controle, podem ter vários blocos, dispostos no mesmo nível hierárquico ou aninhados.

Os blocos de comandos contêm comandos ou outros blocos de comandos (aninhamento). Para indicar que um comando (ou bloco de comando) pertence a um bloco de comando, este comando (ou bloco) deve estar recuado de uma unidade de tabulação (TAB) à direita do bloco ao qual pertence.

Os comandos que fazem parte do mesmo bloco de comandos devem estar no mesmo nível de tabulação.

Exemplo em C++

```
#include <iostream.h>
int main() {
    int a, b, c;
    for (a = 0; a < 10; a++) {
        cin >> b;
        if (0 == a % 2) {
            c = b + a;
        }
        else {
            c = b - a;
        }
        cout << c;
    }
    return 0;
}
```

- Nível 1: primeiro bloco do programa. Sempre irá existir.
- Nível 2: comandos pertencentes ao nível 1.
- Nível 3: comandos pertencentes ao nível 2.
- Nível 4: comandos pertencentes ao nível 3.
- Podem existir N níveis em um programa.
- Para cada nível novo, um TAB a mais deve ser dado.
- Os blocos de comando começam e terminam sempre no mesmo nível

Exemplo em Portugol

```
início
   inteiro: A, B, C;
   para A de 0 até 9 passo 1 faça
      leia (B);
      se A mod 2 = 0
         então C <- B + A;
         senão C <- B – A;
      fimse;
   fimpara;
fim.
```

- Nível 1: primeiro bloco do programa. Sempre irá existir.
- Nível 2: comandos pertencentes ao nível 1.
- Nível 3: comandos pertencentes ao nível 2.
- Nível 4: comandos pertencentes ao nível 3.
- Podem existir N níveis em um programa.
- Para cada nível novo, um TAB a mais deve ser dado.
- Os blocos de comando começam e terminam sempre no mesmo nível

Regrais gerais para identação

- O programa sempre começa na margem esquerda do editor de textos.
- Sempre após a abertura de uma chave, que indica o começo de um novo bloco de comandos, escreveremos os comandos seguintes recuados uma tabulação do bloco a que pertencem.
- Os blocos são fechados no mesmo nível em que são abertos.
- Ao se fechar a chave, o bloco de comandos termina. Os comandos seguintes estarão no mesmo nível do bloco.

Outras operações matemáticas

Potenciação

A potenciação é uma operação aritmética que indica a multiplicação de uma dada base por ela mesma, tantas vezes quanto indicar o expoente: Exemplo $2^3 = 2 \times 2 \times 2 = 8$

Em Portugol, a operação é realizada com o operador **. Exemplo: 2**3 = 8. Em C++, utiliza-se a função pow, declarada na biblioteca math.h. Exemplo: pow(2, 3) = 8.

Quociente

O quociente é o resultado da divisão entre dois números. Exemplo: 13 / 4 = 3. O operador quociente retorna o quociente inteiro da divisão de dois números inteiros.

Em Portugol, utiliza-se o operador div. Exemplo: 13 div 4 = 3. Em C++, utiliza-se o operador /. Exemplo: 13 / 4 = 3. Note que este operador é o mesmo utilizado para a divisão em C++. Este operador é sobrecarregado, isto é, ele possui comportamentos distintos dependendo dos seus operadandos.

Se os dois operandos são do tipo inteiro, o operador calcula o quociente da divisão. Se um dos operandos for do tipo real, o operador irá calcular a divisão real (sem resto e possivelmente com casas decimais).

Exemplos:

- 13 / 4 = 3 → quociente da divisão
- (float)13 / 4 = 3.25 → divisão real

Como é possível observar no exemplo acima, quando queremos o quociente da divisão, basta utilizar o operador com os operandos normalmente. Mas, quando queremos a divisão real, precisamos explicitar este fato, tornando um dos operandos um número real. Isto é feito colocando-se a palavra-chave *float* entre parênteses na frente do primeiro operando.

Resto

O resto é o que sobra da divisão de dois números, quando o quociente da divisão é um número inteiro. Quando é feita a divisão real, não existe resto. Exemplo, o resto da divisão de 13 por 4 é igual a 1.

Em Portugol, o operador de resto é o mod. Exemplo: 13 mod 4 = 1. Em C++, o operador é o %. Exemplo: 13 % 4 = 1.

Operações Lógicas

Quando realizamos uma operação matemática sobre dois valores quaisquer, o resultado será um número inteiro ou real. Por exemplo, suponhamos que a variável A contém o valor 50 e a variável B contém o valor 10. Se fizermos A / B obteremos 5, A * B obteremos 500, A - B obteremos 40.

Mas, se ao invés de fazermos uma operação aritmética, fizermos uma comparação entre A e B, como por exemplo, A < B? Qual seria o resultado? Pois bem, A < B é uma comparação entre A e B, pois estamos perguntando se A é menor que B. Para esta pergunta (comparação) só há dois resultados possíveis: VERDADEIRO ou FALSO. No caso, sabemos que é falso, pois 50 não é menor que 10. Mas caso façamos A > B o resultado será verdadeiro.

Estes são exemplos de operações lógicas. Assim como para operações aritméticas usamos os símbolos aritméticos (** , *, / , + , -), para operações lógicas usaremos os símbolos e operadores lógicos.

Símbolos Lógicos (Portugol)

Símbolo Lógico	Significado

>	Maior que
<	Menor que
=	Igual a
<>	Diferente de
>=	Maior ou igual a
<=	Menor ou igual a

Símbolos Lógicos (C++)

Símbolo Lógico	Significado
>	Maior que
<	Menor que
==	Igual a
! =	Diferente de
>=	Maior ou igual a
<=	Menor ou igual a

Além dos símbolos lógicos existem os operadores lógicos, que conectam expressões do tipo lógico e também retornam um valor lógico, verdadeiro ou falso.

Exemplo: sejam duas variáveis A = 5 e B = 3. Neste caso teríamos:

- A = B → falso
- A <> B → verdadeiro
- A > B → verdadeiro
- A < B → falso
- A >= B → verdadeiro
- A <= B → falso

Operadores Lógicos

- E (and): utilizado para conjunção. Retorna verdadeiro se ambas as partes forem verdadeiras.
- Ou (or): utilizado para disjunção. Basta que uma parte seja verdadeira para o resultado ser verdadeiro.
- Não (not): utilizado para negação. Inverte o estado, de verdadeiro para falso e vice-versa.

Para que possamos entender estes operadores, vejamos um exemplo prático: Suponhamos que em um concurso para professor é exigida idade maior que 21 anos e tempo de experiência maior que dois anos. Para tal, é necessário que as duas comparações (IDADE > 21) e (EXPERIÊNCIA > 2) sejam verdadeiras, pois caso contrário o candidato não preenche os pré -requisitos. Estamos usando o operador ***e*** para testarmos duas condições e verificamos, portanto que ambas têm que ser verdadeiras para que o resultado final seja verdadeiro.

Suponhamos agora que o mesmo concurso exija que o candidato tenha apenas que ter mais de dois anos de experiência ou grau de instrução igual a três (que corresponde a terceiro grau). Neste caso a comparação seria (EXPERIÊNCIA > 2) OU (INSTRUÇÃO

= 3). Sabemos, é claro, que basta uma condição ser verdadeira para que ele seja aprovado. Logo, concluímos que ao usar o operador ou entre duas comparações, estamos considerando que apenas uma condição precisa ser verdadeira para o resultado final ser verdadeiro.

Quanto ao operador não, quando colocado antes de uma condição, ele nega o resultado, ou seja, não verdadeiro significa falso.

Portugol

A	B	A e B	A ou B	não (A)
V	V	V	V	F
V	F	F	V	F
F	V	F	V	V
F	F	F	F	V

C++

A	B	A && B	A || B	!A
V	V	V	V	F
V	F	F	V	F
F	V	F	V	V
F	F	F	F	V

Exemplo: sejam três variáveis A = 5, B = 8 e C = 1. Neste caso teríamos:

- (A = B) e (B > C) → F e V → F
- (A <> B) ou (B < C) → V ou F → V
- não (A < B) → não F → V
- (A < B) e (B > C) → V e V → V
- (A >= B) ou (B = C) → F ou F → F
- não (A <= B) → não V → F

6 – Estrutura Condicional Simples e Composta

Estrutura Condicional Simples

Os exemplos vistos até agora tratavam de programas em que todos os comandos eram executados seqüencialmente, sem que qualquer parte do programa deixasse de ser executada.

Imaginemos agora a seguinte situação: um usuário propõe que seja criado para ele um programa onde ele possa digitar um valor numérico inteiro e deseja que o programa responda se este valor é par através de uma mensagem.

Ora, sabemos que o programa deverá conter a mensagem "ESTE NÚMERO É PAR". Mas a mensagem só deverá aparecer no vídeo SE o número for realmente par.

E como verificar se o número é par? Quando o resto da divisão deste número por 2 for igual a 0, ou seja, se (NUMERO mod 2 = 0).

Nós iremos fazer esta comparação de igualdade e, caso ela seja verdadeira, imprimiremos a mensagem, caso contrário, não executaremos a impressão. Para tanto, utilizaremos a ESTRUTURA CONDICIONAL SIMPLES e o programa ficará assim:

Exemplo 6a:

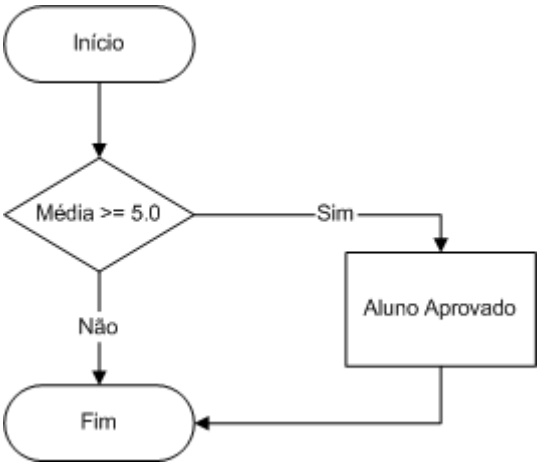
```
início
     inteiro: NUM;

     leia (NUM);
     se NUM mod 2 = 0
          então imprima (NUM, " é par");
     fimse;

fim.
```

As estruturas condicionais são também chamadas estruturas de decisão, pois o andamento do algoritmo depende de uma decisão que será tomada. Esta decisão vem na forma de uma expressão lógico-aritmética cujo resultado sempre será VERDADEIRO ou FALSO.

Sendo o resultado da expressão verdadeiro, o bloco de comandos dentro da estrutura condicional será executado. Caso o resultado seja falso, tal bloco não será executado e o programa continua sua execução. Graficamente, temos algo semelhante à figura abaixo:



Se a média é maior ou igual a 5.0, o aluno foi aprovado. Caso contrário nada acontece.

Portugol

Em Portugol, a estrutura condicional simples é representada pelo comando **se então**. A sintaxe é:

```
se condição_lógica
    então comandos;
fimse;
```

No exemplo 5a, a condição lógica é `NUM mod 2 = 0`, ou seja, se o resto da divisão do conteúdo da variável `NUM` por 2 é igual a zero. Caso essa expressão seja verdadeira, significa que o número é par e, nesse caso, o programa escreve que o número é par. Caso a condição seja falsa, nada acontece e o programa não imprime nada na tela.

C++

Em C++, a estrutura condicional simples é representada pelo comando **if**. A sintaxe é:

```
if (condição_lógica) {
    comandos;
}
```

Exemplo:

```
#include <iostream.h>
using namespace std;
int main() {
  int num;
  cin >> num;
  if (num % 2 == 0) {
    cout << num << " é par";
  }
  return 0;
}
```

Estrutura Condicional Composta

Imagine que o usuário queria incrementar o programa do exemplo 6a, modificando para dizer se um número é par ou se é impar. Uma das formas de se fazer isto é utilizando duas estruturas condicionais simples, como mostrado no exemplo a seguir:

Exemplo 6b:

```
início
     inteiro: NUM;

     leia (NUM);
     se NUM mod 2 = 0
          então imprima (NUM, " é par");
     fimse;
     se NUM mod 2 <> 0
          então imprima (NUM, " é ímpar");
     fimse;
fim.
```

No caso acima, a primeira estrutura condicional testa se o número é par e a segunda testa se ele é impar. Para cada número lido, haverá sempre dois testes, cada um feito por uma das estruturas condicionais simples.

No entanto, sabemos que um número inteiro qualquer ou é par ou é ímpar; ele não pode ser as duas coisas ao mesmo tempo. Portanto, se o primeiro teste do programa acima der verdadeiro, o segundo obrigatoriamente dará falso, e vice-versa.

Em casos como este podemos utilizar uma outra estrutura condicional, chamada estrutura condicional composta e que pode ser entendida como uma extensão da estrutura condicional simples.

Na estrutura condicional composta, realizamos um teste sobre uma condição lógica. Se a condição for verdadeira, um bloco de comandos será executado. Se a condição for falsa, um outro bloco de comandos será executado. Dessa forma, o exemplo 6b pode ser modificado para o seguinte programa:

Exemplo 6c:

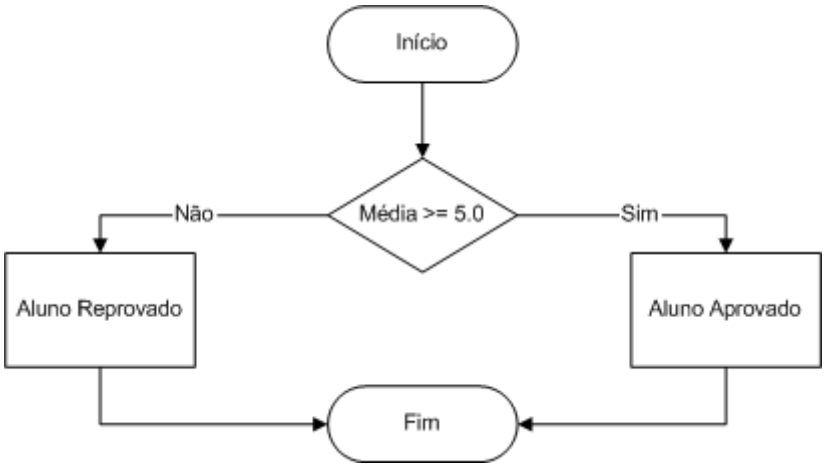
```
início
     inteiro: NUM;

     leia (NUM);
     se NUM mod 2 = 0
          então imprima (NUM, " é par");
          senão imprima (NUM, " é ímpar");
     fimse;
fim.
```

Se o resto da divisão do número por 2 for zero, o programa escreverá que o número é par; caso contrário, escreverá que ele é ímpar. Note que o efeito final dos programas 6b e 6c é o mesmo, isto é, os dois realizam a mesma tarefa, que é dizer se um número é par

ou ímpar. Entretanto, o programa 6c é mais eficiente, pois apenas um teste lógico é realizado; já no programa 6b, são realizados dois testes lógicos. Sendo assim, a solução 6c deve ser preferencialmente utilizada.

Graficamente, temos a seguinte representação para a estrutura condicional composta.



Se a média é maior ou igual a 5.0, o aluno foi aprovado. Caso contrário o aluno foi reprovado. Pode-se notar que é um pouco diferente da estrutura condicional simples, onde não havia comandos a serem executados caso o teste da condição lógica fosse falso.

Portugol

Em Portugol, a estrutura condicional composta é representada pelo comando **se então senão**. A sintaxe é:

```
se condição_lógica
    então comandos1;
    senão comandos2;
fimse;
```

Quando a condição lógica é verdadeira o bloco de comando após o então é executado e o bloco de comandos após o senão não é executado. Quando a condição é falsa, ocorre o contrário.

C++

Em C++, a estrutura condicional compost é representada pelo comando **if else**. A sintaxe é:

```
if (condição_lógica) {
    comandos1;
}
else {
    comandos2;
}
```

Exemplo:

```
#include <iostream.h>
using namespace std;
int main() {
  int num;
  cin >> num;
  if (num % 2 == 0) {
    cout << num << " é par";
  }
  else {
    cout << num << " é ímpar";
  }
  return 0;
}
```

Exercício Resolvido:

Em uma locadora de fitas de vídeo, o valor que se cobra do cliente obedece à seguinte fórmula:

Valor cobrado = quant. de fitas * quant. de dias * R$ 1,50.

Nesta locadora existe uma promoção oferecendo aos clientes que alugam 5 ou mais fitas de uma só vez, o desconto de 50% no valor cobrado. Faça um algoritmo que leia a quantidade de fitas alugadas, a quantidade de dias e calcule e imprima o valor a pagar.

Exemplo 6d:

```
início
     inteiro: FITAS, DIAS;

     leia (FITAS, DIAS);

     se FITAS >= 5
          então VALOR <- FITAS * DIAS * 1.50 / 2;
          senão VALOR <- FITAS * DIAS * 1.50;
     fimse;

     imprima (VALOR);
fim.
```

Aninhamento de Estruturas Condicionais

Quando necessitamos fazer um conjunto de testes de condições mutuamente exclusivas (se uma ocorrer as demais não ocorrem), devemos aninhar as estruturas condicionais, de modo a melhorar a eficiência do programa.

Por exemplo, considere a seguinte tabela, de percentual de imposto sobre o salário:

Salário	Imposto
Até R$500,00	7%
De R$500,01 até R$1000,00	9%
De R$1000,01 até R$1500,00	10%
Acima de R$1500,00	15%

Um dado salário corresponde a apenas um percentual de imposto. Logo, não é estritamente necessário testar se o salário corresponde a todos os percentuais. Veja nos trechos abaixo como fazer o aninhamento de estruturas condicionais tanto em Portugol quanto em C++.

Portugol

```
se SALARIO <= 500
  então IMPOSTO <- SALARIO * 0.07;
  senão
    se SALARIO > 500 e SALARIO <= 1000
      então IMPOSTO <- SALARIO * 0.09;
      senão
        se SALARIO > 1000 e SALARIO <= 1500
          então IMPOSTO <- SALARIO * 0.10;
          senão IMPOSTO <- SALARIO * 0.15;
        fimse;
    fimse;
fimse;
```

C++

```
if (SALARIO <= 500) {
  IMPOSTO = SALARIO * 0.07;
}
else {
  if (SALARIO > 500 && SALARIO <= 1000) {
    IMPOSTO = SALARIO * 0.09;
  }
  else {
    if (SALARIO > 1000 && SALARIO <= 1500) {
      IMPOSTO = SALARIO * 0.10;
    }
    else {
      IMPOSTO = SALARIO * 0.15;
    }
  }
}
```

Estrutura Condicional de Seleção Múltipla

Quando nos deparamos com situações onde condições mutuamente exclusivas precisam ser testadas, é possível utilizar a estrutura condicional de seleção múltipla, nos casos em que os valores a serem testados são valores individuais.

Por exemplo, considere a tabela abaixo, onde cada classe funcional corresponde a um salário:

Classe Funcional	Salário
1	R$500,00
2	R$750,00
3	R$1000,00
4	R$2000,00

Para esta tabela, podemos utilizar o aninhamento de estruturas condicionais, pois as condições são mutuamente exclusivas (uma dada pessoa só pertencerá a uma classe funcional).

Portugol

```
se CLASSE = 1
  então SALARIO <- 500;
  senão
    se CLASSE = 2
      então SALARIO <- 750;
      senão
        se CLASSE = 3
          então SALARIO <- 1000;
          senão SALARIO <- 2000;
        fimse;
    fimse;
fimse;
```

C++

```
if (1 == classe) {
  salario = 500;
}
else {
  if (2 == classe) {
    salario = 750;
  }
  else {
    if (3 == classe) {
      salario = 1000;
    }
    else {
      salario = 2000;
    }
  }
}
```

No entanto, uma maneira mais elegante e simples de se fazer a lógica acima é com a utilização da estrutura condicional de seleção múltipla. Nessa estrutura, um de vários

caminhos será selecionado para execução, sendo o caminho selecionado escolhido com base em alguma condição a ser testada.

Em Portugol, essa estrutura é representada pelo comando **escolha** possui a seguinte sintaxe:

```
escolha (valor_inteiro)
     caso constante_1:
          comando_1;
          comando_2;
          ...
          comando_n;
          pare;
     ...
     caso constante_n:
          comando_1;
          comando_2;
          ...
          comando_n;
          pare;
     padrão:
          comando_1;
          comando_2;
          ...
          comando_n;
fimescolha;
```

Já em C++, temos o comando **switch**, cuja sintaxe é mostrada abaixo:

```
switch (valor_inteiro) {
     case constante_1:
          comando_1;
          comando_2;
          ...
          comando_n;
          break;
     ...
     case constante_n:
          comando_1;
          comando_2;
          ...
          comando_n;
          break;
     default:
          comando_1;
          comando_2;
          ...
          comando_n;
}
```

Vejamos como está estrutura trabalha na prática, através da codificação da lógica anterior:

Portugol

```
escolha (CLASSE)
  caso 1:
    SALARIO <- 500;
    pare;

  caso 2:
    SALARIO <- 750;
    pare;

  caso 3:
    SALARIO <- 1000;
    pare;

  padrão:
    SALARIO <- 2000;
    pare;
fimescolha;
```

Na instrução escolha, o valor da variável CLASSE é testado. Se o valor for igual a um dos casos especificados (1, 2, ou 3), as instruções daquele caso serão executadas. Se o valor não for igual a nenhum caso, as instruções do caso padrão serão executadas. Por exemplo, se CLASSE é igual a 2, o caso 2 será executado e a variável SALARIO receberá o valor 750. Se CLASSE for igual a 4, o caso padrão será executado, uma vez que 4 não "casa" com os casos 1, 2 ou 3. Sendo assim, a variável SALARIO receberá o valor 2000.

C++

```
switch (classe) {
  case 1:
    salario = 500;
    break;

  case 2:
    salario = 750;
    break;

  case 3:
    salario = 1000;
    break;

  default:
    salario = 2000;
    break;
}
```

O funcionamento da estrutura em C++ é idêntico ao funcionamento em Portugol.

É importante observar que a estrutura condicional de seleção múltipla só funciona com valores do tipo inteiro, ou seja, o que vem entre parênteses nos comandos **escolha** ou **switch** deve ser do tipo inteiro, podendo ser um número, constante ou variável, por exemplo.

Outra restrição da estrutura condicional de seleção múltipla é que ela só funciona para testes de igualdade (ou diferença). Se for necessário testar faixas de valores (como mostrado no exemplo de aninhamento de estruturas condicionais), não é possível utilizar nem **escolha** em Portugol nem **switch** em C++. Nesses casos, será necessário utilizar o aninhamento de estruturas condicionais.

7 – Estruturas de Repetição

Até agora, vimos a estruturas seqüencial e condicional de programação. Na primeira, todos os comandos são executados um após o outro, do início até o final do programa. Na estrutura condicional, trechos do programa podem ou não ser executados, dependendo do resultado de um ou mais testes lógicos.

Estas estruturas permitem que se construam programas com relativa complexidade. No entanto, elas não são suficientes para abordar vários tipos de problema. Considere, por exemplo, o seguinte enunciado: construir um programa que leia o NOME e a IDADE de 20 pessoas e para imprima os NOMES das pessoas que possuam IDADE superior a 30 anos.

Com as estruturas que vimos até o momento, a única forma de ser ler o nome e a idade de 20 pessoas é criar 20 variáveis para o nome e 20 para a idade, e escrever vários comandos de entrada para ler estas variáveis. Esta não é melhor maneira de se resolver o problema, pois resultaria em um programa de difícil entendimento e manutenção.

Para resolver estes tipos de problemas, onde um certo conjunto de operações deve ser executado mais de uma vez, vamos introduzir novas estruturas de controle, chamadas estruturas de repetição.

As estruturas de repetição podem ser utilizadas para vários tipos de problemas como, por exemplo, teste de senha, contagem e soma de valores, recebimento de uma certa quantidade de dados de entrada, percorrer estruturas de dados (vetores).

Repetição com Variável de Controle

A estrutura de repetição com variável de controle serve para se executar um trecho de código *n* vezes, onde *n* é um número conhecido previamente. O problema anterior pode ser tratado por este tipo de estrutura pois, como vimos, deseja-se ler o nome e a idade de exatamente 20 pessoas.

O número de repetições executado por esta estrutura é controlado por uma variável, chamada variável de controle. Esta variável começa com um valor inicial estabelecido pelo programador e, a cada repetição (iteração ou loop), o valor é incrementado ou decrementado, até alcançar o valor final, também definido pelo programador.

Vejamos como ficaria o programa para ler 20 nomes e idades com a utilização da estrutura de repetição com variável de controle.

Portugol

```
início
  inteiro: IDADE, CONTADOR;
  caracter: NOME;

  para CONTADOR de 1 até 20 passo 1 faça
    leia (NOME, IDADE);
```

```
    se IDADE > 30
      então imprima (NOME);
    fimse;
  fimpara;
fim.
```

Em Portugol, a estrutura de repetição com variável de controle é representada pelo comando **para**. Este programa faz a variável de controle CONTADOR receber 1 no início da repetição. Quando o fluxo volta à estrutura de repetição, ele testa se a variável atingiu o limite superior (20). Caso não tenha atingido, ela é incrementada (acrescida) de 1 através do comando *passo 1* e entra na repetição novamente. O processo se repete até que a variável atinja o valor 20, quando parte para o fim do programa.

C++

```
#include <iostream.h>
using namespace std;

int main() {
  int idade, contador;
  string nome;

  for (contador = 0; contador < 20; contador++) {
    cin >> nome >> idade;

    if (idade > 30) {
      cout << nome;
    }
  }
  return 0;
}
```

Em C++, a estrutura de repetição com variável de controle é representada pelo comando **for**, que é dividido em três partes, separadas por ponto e vírgula (;). A primeira parte inicializa a variável de controle com o valor inicial (zero, no caso do exemplo). A segunda parte é o critério de parada (contador < 20). A terceira parte é o incremento ou decremento da variável de controle.

Repetição com Teste no Início

Uma estrutura de repetição permite que um mesmo trecho de programa possa ser executado por repetidas vezes, sem a necessidade de escrever o trecho de comando a quantidade de vezes que queremos repeti-lo.

A estrutura de repetição com teste no início é uma estrutura na qual o teste para se verificar se o loop executará a próxima vez ou não é realizado no início da estrutura, antes de qualquer outro comando. Dessa forma, se o teste der falso logo na primeira vez, os comandos dentro da estrutura não serão executados nenhuma única vez. Essa é a principal característica dessa estrutura de repetição. Vejamos como fica o exemplo quando se utiliza essa estrutura.

Portugol

O comando utilizado para estrutura de repetição com teste no início é:

```
enquanto <condição_lógica> faça
     comando1;
     comando2;
     ...
     comandoN;
fimenquanto;
```

No início da execução, a condição lógica é avaliada. Se ela for verdadeira, os comando dentro da estrutura de repetição são executados. Quando o programa chega na instrução `fimenquanto`, ele volta novamente ao início e avalia a condição novamente. Esse processo continua enquanto a condição for verdadeira.

Exemplo:

```
início
     inteiro: IDADE, CONTADOR;
     caracter: NOME;
     CONTADOR <- 0;
     enquanto CONTADOR < 20 faça
          leia (NOME, IDADE);
          se IDADE > 30
               então imprima (NOME);
          fimse;
          CONTADOR <- CONTADOR + 1;
     fimenquanto;
fim.
```

Note que esse programa realiza a mesma tarefa do programa anterior. Se a variável CONTADOR tivesse sido inicializada com o valor 20, o loop não executaria nenhuma única vez, pois 20 < 20 dá falso e o loop executa enquanto a condição lógica for verdadeira.

C++

O comando da estrutura de repetição com teste no início em C++ é o **while**, e sua sintaxe é a seguinte:

```
while (condição_lógica) {
    comando_1;
    comando_2;
    ...
    comando_n;
}
```

Exemplo:

```
#include <iostream.h>
```

```
using namespace std;

int main() {
  int idade, contador;
  string nome;

  contador = 0;
  while (contador < 20) {
    cin >> nome >> idade;

    if (idade > 30) {
      cout << nome;
    }
    contador++;
  }
  return 0;
}
```

Uso de Condição de Parada (Flag)

Nos exemplos vistos até o momento, as estruturas de repetição são executadas um número fixo de vezes (vinte), que foi definido pelo programador no momento da construção do programa.

No entanto, na grande maioria das vezes o programador não saberá quantas vezes a estrutura deverá ser executada, pois esta é uma decisão que geralmente cabe ao usuário do programa. O usuário pode, por exemplo, querer informar um, dois, dez ou até mesmo nenhum conjunto de dados. Sendo assim, o programa deverá realizar uma, duas, dez ou nenhuma iteração (loop), dependendo da vontade do usuário. Como fazer então para eventualmente parar as iterações, de modo que o programa chegue ao seu final?

Neste caso, precisamos criar uma condição de parada (também chamada de flag). A condição de parada geralmente é um valor específico de alguma variável pré-determinada. Quando a variável assume este valor (que é dado pelo usuário), significa que o usuário não deseja mais entrar com valores para o programa e este deve, então, sair da estrutura de repetição e continuar a executar as demais instruções, terminando o processamento e realizando as saídas de dados.

No exemplo dos nomes e das idades, se não quisermos obrigar o usuário a digitar exatamente 20 nomes e 20 idades, temos que utilizar um flag para que ele possa informar a quantidade de dados que desejar, quer sejam mais ou menos de 20. Um bom flag para este caso seria, por exemplo, idade menor que zero. Como não existem idades negativas, estes valores podem ser utilizados como critério de parada da estrutura de repetição. Vejamos como isto fica em Portugol e em C++:

Portugol

```
início
     inteiro: IDADE;
     caracter: NOME;
```

```
      leia (IDADE);
      enquanto IDADE >= 0 faça
            leia (NOME);
            se IDADE > 30
                  então imprima (NOME);
            fimse;
            leia (IDADE);
      fimenquanto;
fim.
```

Este programa funciona da seguinte maneira. Primeiro é lida uma idade fora da estrutura de repetição. Se ela for negativa, o programa não entra na estrutura, pois o usuário informou justamente um valor que satisfaz o critério de parada e, portanto, não quer informar nenhum dado. Se a idade for maior ou igual a zero, o programa entra na estrutura de repetição, onde acontecem o restante das leituras e o processamento. Como última instrução, temos novamente a leitura da idade. O usuário irá então informar um valor e o processo irá se repetir, até o critério de parada ser atingido.

C++

```
#include <iostream.h>
using namespace std;

int main() {
  int idade;
  string nome;

  cin >> idade;
  while (idade >= 0) {
    cin >> nome;
    if (idade > 30) {
      cout << nome;
    }
  }
  return 0;
}
```

O funcionamento em C++ é análogo ao funcionamento em Portugol.

Repetição com Teste no Final

A estrutura de repetição com teste no final possui um funcionamento semelhante ao da estrutura de repetição com teste no início. A principal diferença entre elas é que na com teste no final os comandos que estão dentro da estrutura de repetição serão obrigatoriamente executados pelo menos uma vez. Em outras palavras, nessa estrutura, os comandos são executados uma ou mais vezes. Na repetição com teste no início, os comandos são executados zero ou mais vezes.

Sendo assim, quando temos certeza que o loop deve ser executado pelo menos uma vez, mas não sabemos quantas vezes ele deverá ser executado, a estrutura de repetição com teste no final deve ser preferencialmente escolhida.

Os flags (condições de parada) também podem ser utilizados com a estrutura de repetição com teste no final.

Portugol

Em Portugol, o comando para a estrutura de repetição com teste no final é o **repita...até**. Neste comando, as instruções são repetidas até a condição lógica ser verdadeira. Em outras palavras, enquanto a condição lógica for **FALSA**, os comandos serão executados. A lógica aqui é um pouco diferente da repetição com teste no início, que executa enquanto a condição for verdadeira.

```
repita
    comando_1;
    comando_2;
    ...
    comando_n;
até condição_lógica;
```

Exemplo:

```
início
     inteiro: IDADE;
     caracter: NOME;

     repita
          leia (NOME, IDADE);
          se IDADE > 30
               então imprima (NOME);
          fimse;
     até IDADE < 0;
fim.
```

O programa continua fazendo o mesmo processamento, isto é, lendo nomes e idades e imprimindo o nome da pessoa, se ela possuir idade superior a 30 anos. No entanto, o teste da condição lógica é realizado no final da estrutura, o que garante que ela será executada pelo menos uma vez. Além disso, a condição lógica testada aqui (`IDADE < 0`) é o contrário daquela testa na repetição com teste no início (`IDADE >= 0`).

C++

Em C++, o comando da estrutura de repetição com teste no final é o **do...while**. Diferentemente de Portugol, essa estrutura executa enquanto a condição lógica for verdadeira, sendo então semelhante à lógica da estrutura de repetição com teste no final, com a única diferença do ponto onde o teste lógico é feito (início ou final da estrutura de repetição).

```
do {
    comando_1;
    comando_2;
    ...
    comando_n;
} while (condição_lógica);
```

Exemplo:

```
#include <iostream.h>
using namespace std;

int main() {
  int idade;
  string nome;

  do {
    cin >> nome >> idade;
    if (idade > 30) {
      cout << nome;
    }
  } while (idade >= 0);
  return 0;
}
```

Uso da Estrutura

Como exemplo de situação típica onde a estrutura de repetição com teste no final pode ser utilizada, temos a validação de dados de entrada informados pelo usuário. Suponha, por exemplo, que o usuário tenha que informar um valor para altura, e este valor deve ser maior que 0 e menor que 3 (em metros). A leitura da altura pode ser feita dentro da estrutura de repetição; ao final dela, teremos o teste da altura. Se a altura for válida, o programa prossegue; do contrário, ele volta a solicitar a altura ao usuário.

Portugol

```
repita
  leia (ALTURA);
até ALTURA > 0 e ALTURA < 3;
```

C++

```
do {
  cin >> altura;
} while (altura <= 0 || altura >= 3)
```

8 – Estruturas de Dados Homogêneas (Vetores)

Introdução

Para introduzirmos o assunto, vamos criar um algoritmo em Portugol que leia cinco valores e calcule a média destes valores lidos, imprimindo os resultados ao final da execução:

```
início
  real: VALOR1, VALOR2, VALOR3, VALOR4, VALOR5, SOMA, MEDIA;

  leia (VALOR1, VALOR2, VALOR3, VALOR4, VALOR5);
  SOMA <- VALOR1 + VALOR2 + VALOR3 + VALOR4 + VALOR5;
  MEDIA <- SOMA / 5;
  imprima (SOMA, MEDIA);

fim.
```

Este algoritmo é bastante simples. Mas imagine agora que fosse necessário desenvolver este mesmo algoritmo para 50, 100 ou 1000 números. Neste caso, teríamos que declarar 50, 100 ou 1000 variáveis VALOR, o que tornaria o programa impraticável, uma vez que seria muito difícil desenvolve-lo e corrigi-lo no caso de eventuais problemas. Até mesmo o entendimento do algoritmo ficaria prejudicado, devido à grande quantidade de variáveis.

Existem formas melhores para se trabalhar este problema. A primeira solução seria utilizar uma estrutura de repetição para ler os valores, onde teríamos o seguinte algoritmo:

```
início
  real: VALOR, SOMA, MEDIA;
  inteiro: CONTADOR;
  SOMA <- 0;
  para CONTADOR de 1 até 100 passo 1 faça
    leia (VALOR);
    SOMA <- SOMA + VALOR;
  fimpara;
  MEDIA <- SOMA / CONTADOR;
  imprima (SOMA, MEDIA);
fim.
```

Note que podemos facilmente modificar este programa para ler quantos valores nós desejemos, apenas modificando o valor final do contador na estrutura de repetição.

Esta solução já melhora em muito a legibilidade do código, tornando-o bem mais simples e fácil de se manter. No entanto, ela apresenta uma desvantagem em relação à solução anterior.

Observe que a leitura dos 100 valores sempre é feita na variável VALOR. Dessa forma, não temos como recuperar todos os valores lidos (só teríamos o último), algo que acontecia na primeira solução.

Para resolver este problema, vamos introduzir uma nova estrutura de dados, chamada vetor.

O vetor é uma estrutura de dados capaz de armazenar vários elementos de um mesmo tipo de dados (por exemplo, inteiros ou reais). Essa é uma de suas diferenças em relação a uma variável comum, onde é possível armazenar apenas um único valor de um determinado tipo.

Na memória do computador, teríamos as seguintes representações para um vetor e uma variável comum:

Vetor de Reais

-9.6	10.5	0	1000	-5000

Variável do Tipo Real

19.2

O vetor é uma variável composta unidimensional homogênea

- Composta, porque possui vários elementos.
- Unidimensional, porque possui apenas uma dimensão, diferente de outras variável com mais dimensões (matrizes, por exemplo, que são bidimensionais).
- Homogênea, porque nas posições do vetor devem ser armazenados elementos do mesmo tipo; ou seja, não é possível armazenar um nome na primeira posição e na segunda armazenar um salário (nome é do tipo caracter e salário é do tipo real).

O vetor é então uma variável composta de várias posições, onde cada posição pode ser considerada como uma variável simples. Sendo assim, Por exemplo, cada posição de um vetor de inteiros representa uma variável do tipo inteiro, e todas elas possuem o mesmo nome do vetor.

Sendo o nome o mesmo para todos os elementos de um vetor, como fazer para diferencia-los, ou seja, para acessar individualmente cada um dos elementos? Neste caso, o acesso é feito pelo índice, um valor do tipo inteiro que indica a posição do elemento dentro do vetor.

O índice vem logo o nome da variável do tipo vetor, sempre entre colchetes. Então, se tivermos, por exemplo, uma variável chamada VALOR e quisermos atribuir o valor 95 a sua quarta posição e ler o valor armazenado na sua primeira posição, teríamos as seguintes instruções:

Operação	**Portugol**	**C++**
Atribuição	`VALOR[3] <- 95;`	`valor[3] = 95;`
Leitura	`leia (VALOR[0]);`	`cin >> valor[0];`

VALOR

10.9			95	
0	1	2	3	4

Observe que os índices do vetor começam a partir do zero, e não a partir do um, tanto em Portugol como em C++.

Nas próximas seções veremos como utilizar os vetores em nessas duas linguagens.

Vetores em Portugol

Declaração

A declaração de um vetor em Portugol é feita da seguinte forma:

```
vetor [li:ls] <tipo_dado><nome_da_variavel>;
```

Onde:

- vetor: palavra reservada da linguagem
- li: limite inferior do vetor (seu menor índice)
- ls: limite superior do vetor (seu maior índice)
- tipo_dado: o tipo de dados dos elementos do vetor
- nome_da_variavel: nome da variável sendo criada

Exemplo:

- `vetor [0:4] real: VALOR;` Cria uma variável chamada VALOR que é um vetor de número reais cujo menor índice é 0 e maior índice é 4, tendo assim cinco posições.

Note que os valores para **li** e **ls** são determinados pelo programador desenvolvendo o algoritmo. Sendo assim, nada impede que eles sejam 1 e 5, 2 e 6, -1 e 3, por exemplo. No entanto, para termos uma definição semelhante à utilizada em C++, iremos convencionar que o limite inferior (li) sempre será 0.

Acesso aos Elementos

O acesso aos elementos de um vetor sempre se dá de maneira individual, não sendo possível acessar (ler ou escrever) todos os elementos de uma única vez.

Para acessar um determinado elemento, precisamos utilizar o índice do mesmo, de modo a indicar com qual posição do vetor estamos trabalhando.

Podemos então acessar os elementos do vetor como se fossem variáveis isoladas, como mostrado a seguir:

```
leia (VALOR[0], VALOR[1], VALOR[2], VALOR[3], VALOR[4]);
```

Esta abordagem não é indicada, pois se o vetor possuir muitos elementos, teremos novamente um código grande e de difícil manutenção. A solução ideal para o acesso ao vetor é utilizar uma estrutura de repetição, como mostrado a seguir:

```
para CONTADOR de 0 até 99 passo 1 faça
  leia (VALOR[CONTADOR]);
fimpara;
```

Dessa forma, podemos ler quantos elementos quisermos, apenas mudando o valor final da variável CONTADOR.

Observe que a variável CONTADOR é utilizada para indexar o vetor. Isto é perfeitamente possível, pois dentro dos colchetes pode ser colocada qualquer coisa do tipo inteiro e CONTADOR é uma variável deste tipo (apesar de isso não estar expresso no trecho de código acima).

O processo funciona da seguinte forma: na primeira iteração do loop, CONTADOR terá o valor 0 e estaremos acessando VALOR[0], isto é, a primeira posição do vetor. Depois CONTADOR passa para 1, e acessaremos VALOR[1], a segunda posição. Esta repetição prossegue até CONTADOR atingir o valor 99, onde estaremos acessando a última posição do vetor (VALOR[99]).

Exemplo

Vejamos como fica o exemplo do início deste capítulo com a utilização de vetores:

```
início
  vetor [0:99] real: VALOR;
  inteiro: CONTADOR;
  real: SOMA, MEDIA;
  SOMA <- 0;
  para CONTADOR de 0 até 99 passo 1 faça
    leia (VALOR[CONTADOR]);
    SOMA <- SOMA + VALOR[CONTADOR];
  fimpara;
  MEDIA <- SOMA / CONTADOR;
  imprima (SOMA, MEDIA);
fim.
```

Vetores em C++

Declaração

A declaração de um vetor em C++ é feita especificando-se:

- O tipo de dados do vetor;
- O nome da variável
- O número de elementos do vetor

Sintaxe

```
tipo nome[x];
```

- Tipo especifica o tipo da variável vetor.

- Nome especifica o nome da variável vetor.
- O valor entre colchetes especifica o número de elementos da variável vetor

Exemplos

```
int num[100];  /* Declara um vetor de inteiros de 100
posições */

float valor[5];  // Declara um vetor de reais de 5 posições
```

Acesso aos Elementos

O acesso aos elementos é idêntico ao acesso em Portugol. Devemos sempre acessar cada um dos elementos do vetor, e nunca podemos acessa-lo como um todo.

Podemos acessar os elementos como variáveis comuns, como mostrado a seguir:

```
cin >> valor[0] >> valor[1] >> valor[2] >> valor[3] >>
valor[4];
```
Mas a forma preferível para se fazer o acesso é através de uma estrutura de repetição:

```
for (contador = 0; contador < 100; contador++) {
  cin >> valor[contador];
}
```

Exemplo

Vejamos como fica o exemplo do início deste capítulo com a utilização de vetores:

```
int main() {
  float valor[100], media, soma = 0;
  int contador;
  for (contador = 0; contador < 100; contador++) {
    cin >> valor[contador];
    soma = soma + valor[contador];
  }
  media = soma / contador;
  cout << soma << media;
  return 0;
}
```

Usos do Vetor

Vetor como Área de Armazenamento de Dados

Uma das utilizações do vetor é como área de armazenamento de dados. Quando precisamos ler um conjunto de dados, fazer algum processamento e depois utilizar

novamente os dados, precisamos utilizar o vetor para fazer o armazenamento dos dados lidos, de modo que possamos utiliza-los durante o programa.

Considere o seguinte problema: ler 10 números inteiros, calcular sua média e imprimir os números maiores ou iguais à média. Só é possível calcular a média após a leitura dos 10 números. E para mostrar os que são maiores ou iguais à média, é necessário que todos sejam armazenados para serem comparados com a média calculada. Vejamos como fica a solução em Portugol e em C++:

Portugol

```
início
  vetor [0:9] inteiro: VALOR;
  inteiro: CONTADOR;
  real: SOMA, MEDIA;
  SOMA <- 0;

  para CONTADOR de 0 até 9 passo 1 faça
    leia (VALOR[CONTADOR]);
    SOMA <- SOMA + VALOR[CONTADOR];
  fimpara;

  MEDIA <- SOMA / CONTADOR;  imprima (SOMA, MEDIA);

  para CONTADOR de 0 até 9 passo 1 faça
    se VALOR[CONTADOR] >= MEDIA
      então imprima (VALOR[CONTADOR]);
    fimse;
  fimpara;fim.
```

Na primeira estrutura de repetição, os 10 números são lidos e somados. Logo após a media é calculada. Em seguida, na segunda estrutura de repetição, percorremos o vetor, comparando cada número lido com a média e imprimindo os que são maiores ou iguais à media.

C++

```
int main() {
  int valor[10], contador, soma = 0;
  float media;

  for (contador = 0; contador < 10; contador++) {
    cin >> valor[contador];
    soma = soma + valor[contador];
  }

  media = soma / contador;
  cout << soma << media;

  for (contador = 0; contador < 10; contador++) {
    if (valor[contador] >= media) {
```

```
      cout << valor[contador];
    }
  }

  return 0;
}
```

Vetor como Tabela de Dados

Uma outra utilização do vetor é como tabela para armazenamento de dados. Com a utilização do vetor desta forma, podemos simplificar consideravelmente alguns tipos de programas.

Considere o seguinte enunciado: ler o nome e a classe funcional de 10 empregados, calcular e imprimir o nome e o salário de cada empregado, de acordo com a tabela abaixo.

Classe Funcional	Salário (em reais)
1	350,00
2	800,00
3	1.500,00
4	4.000,00

Até o momento, conhecemos duas formas de se resolver este problema: com aninhamento de estruturas condicionais ou com a estrutura condicional de seleção múltipla, como mostrado abaixo:

Aninhamento de Estruturas Condicionais em Portugol

```
inicio
  caractere: NOME;
  inteiro: CLASSE, CONTADOR;
  real: SALARIO;

  para CONTADOR de 1 até 10 passo 1 faça
    leia (NOME, CLASSE);

    se CLASSE = 1
      então SALARIO <- 350;
      senão
        se CLASSE = 2
          então SALARIO <- 800;
          senão
            se CLASSE = 3
              então SALARIO <- 1500;
              senão SALARIO <- 4000;
            fimse;
        fimse;
    fimse;
```

```
    imprima (NOME, SALARIO);
  fimpara;
fim.
```

Aninhamento de Estruturas Condicionais em C++

```
#include <iostream.h>
using namespace std;

int main() {
  string nome;
  int classe, contador;
  float salario;

  for (contador = 0; contador < 10; contador++) {
    cin >> nome >> classe;

    if (1 == classe) {
      salario = 350;
    }
    else {
      if (2 == classe) {
        salario = 800;
      }
      else {
        if (3 == classe) {
          salario = 1500;
        }
        else {
          salario = 4000;
        }
      }
    }
    cout << nome << salario;
  }
  return 0;
}
```

Estrutura Condicional de Seleção Múltipla em Portugol

```
inicio
  caractere: NOME;
  inteiro: CLASSE, CONTADOR;
  real: SALARIO;

  para CONTADOR de 1 até 10 passo 1 faça
    leia (NOME, CLASSE);

    escolha (CLASSE)
      caso 1:
```

```
        SALARIO <- 350;
        pare;
      caso 2:
        SALARIO <- 800;
        pare;
      caso 3:
        SALARIO <- 1500;
        pare;
      caso 4:
        SALARIO <- 4000;
        pare;
    fimescolha;

    imprima (NOME, SALARIO);
  fimpara;
fim.
```

Estrutura Condicional de Seleção Múltipla em C++

```
#include <iostream.h>
using namespace std;

int main() {
  string nome;
  int classe, contador;
  float salario;

  for (contador = 0; contador < 10; contador++) {
    cin >> nome >> classe;
    switch (classe)
    {
      case 1:
        salario = 350;
        break;
      case 2:
        salario = 800;
        break;
      case 3:
        salario = 1500;
        break;
      case 4:
        salario = 4000;
        break;
    }
    cout << nome << salario;
  }
  return 0;
}
```

As duas soluções funcionam, mas apresentam a seguinte desvantagem. Se o número de classes funcionais aumentar, temos que fazer mais aninhamentos ou acrescentar mais

casos ao nosso programa, o que vai torna-lo maior e, conseqüentemente, mais difícil de ser lido, entendido e corrigido em caso de problemas.

A utilização de vetores permite que possamos acrescentar mais classes funcionais sem haver grandes impactos no código desenvolvido. Considere novamente a tabela de classes e sálarios

Classe Funcional	Salário (em reais)	Vetor Salários
1	350,00	0
2	800,00	1
3	1.500,00	2
4	4.000,00	3

Acrescentamos uma terceira coluna chamada “Vetor Salários”, que representa um vetor de salários; os números de 0 a 3 são os índices desse vetor (0 para a primeira posição, 1 para a segunda e assim por diante).

Note que há uma correspondência entre a classe funcional e o índice do vetor. Quando a classe é 1, o índice é 0; quando a classe é 2, o índice é 1 e assim por diante. Ou seja, dada a classe funcional, basta subtrair 1 que encontramos o índice correspondente no vetor.

Dessa forma podemos armazenar os valores de salários em um vetor e utilizar a classe funcional para indexar o vetor. Assim, se necessitarmos aumentar o número de classes funcionais, basta acrescentar os salários correspondentes no vetor, que será o único ponto do programa que precisaria de alteração. A solução em Portugol e em C++ é mostrada a seguir:

```
inicio
  caractere: NOME;
  inteiro: CLASSE, CONTADOR;
  vetor [0:3] real: SALARIOS;
  SALARIOS[0] <- 350;
  SALARIOS[1] <- 800;
  SALARIOS[2] <- 1500;
  SALARIOS[3] <- 4000;

  para CONTADOR de 1 até 10 passo 1 faça
    leia (NOME, CLASSE);
    imprima (NOME, SALARIOS[CLASSE - 1]);
  fimpara;
fim.
```

```
#include <iostream.h>
using namespace std;

int main() {
  string nome;
  int classe, contador;
  float salarios[4];
```

```
  salarios[0] = 350;
  salarios[1] = 800;
  salarios[2] = 1500;
  salarios[3] = 4000;

  for (contador = 0; contador < 10; contador++) {
    cin >> nome >> classe;
    cout << nome << salarios[classe - 1];
  }
  return 0;
}
```

Note que a solução do problema fica bem menor e mais simples de ser visualizada e entendida. Se quisermos aumentar o número de classes, basta aumentar o tamanho do vetor de salários e armazenar o valor de salário correspondente às novas classes.

Observe também que utilizamos **classe - 1** para indexar o vetor. Isso é perfeitamente possível, pois esta operação resulta em um número inteiro, e este número é utilizado para indexar o vetor, conforme a correspondência entre classe funcional e índice do vetor mostrada anteriormente.

9 – Estruturas de Dados Heterogêneas (Registros)

No capítulo anterior vimos as estruturas de dados homogêneas (vetores), nas quais todos os elementos da estrutura possuem o mesmo tipo.

Neste capítulo trabalharemos como uma outra estrutura de dados, chamada estrutura de dados heterogênea ou registro. Nesta estrutura, como o nome sugere, os elementos componentes poderão ser de tipos distintos, diferentemente do vetor.

A principal vantagem de um registro é a de agrupar dados logicamente relacionados, de modo a se obter uma melhor representação de dados complexos do mundo real. Por exemplo, o endereço de uma empresa pode ser representado por uma variável do tipo cadeia de caracteres, como mostrado abaixo:

```
ENDERECO <- "Rua Conselheiro Lafaiete, nº 2003 - lj. 14 –
Bairro Cidade Nova – CEP: 31170-000 – Belo Horizonte – MG";
```

Podemos observar, no entanto, que este endereço é um dado complexo, pois pode ser dividido em partes menores que facilitariam a sua manipulação:

- Tipo do Logradouro (caracter): Rua
- Logradoutro (caracter): Conselheiro Lafaiete
- Número do Logradouro (inteiro): 2003
- Complemento do Logradouro (caracter): lj. 14
- Bairro (caracter): Cidade Nova
- CEP (caracter): 31170-000
- Cidade (caracter): Belo Horizonte
- UF (caracter): MG

Sendo assim, seria mais interessante tratar este endereço como uma variável do tipo registro, onde teríamos elementos representando cada um dos componentes do tipo complexo.

Vejamos um outro exemplo. Considere que uma passagem de ônibus interestadual seja formada pelo seguinte conjunto de informações logicamente relacionadas:

- Nome da empresa prestadora do serviço
- Número da passagem
- Origem
- Destino
- Distância a ser percorrida
- Data
- Hora
- Poltrona
- Nome do passageiro
- Indicação se passageiro é menor de idade
- Número da carteira de identidade do passageiro

Viação Expressa Ltda. Bilhete Nº: 046765

Origem: Belo Horizonte Destino: Barbacena Distância: 170 km
Data: 03/09/2007 Hora: 08:00 Poltrona: 21

Passageiro: José da Silva Menor de idade: ___ Sim ___ Não
Carteira de Identidade: MG-15.647.643 – SSP/MG

A maneira mais fácil de se lidar com estas informações não seria ter uma única variável do tipo caracter armazenando toda a informação, mas dividi-la em seus componentes e armazena-los em campos de um registro, cada um com o tipo mais apropriado para tratar a informação.

Uma outra importância das estruturas de dados heterogêneas é que elas são precursoras do conceito de classe, um importante elemento do paradigma de programação orientado por objetos. Sendo assim, o entendimento dos registros facilita o posterior entendimento do que são classes e objetos.

Registros em Portugol

Declaração

Um registro pode ser considerado como um novo tipo de dados, que irá se juntar àqueles pré-definidos pela linguagem (inteiro, real, caracter e logico). Sendo assim, a primeira coisa a se fazer para utilizar um registro é cria-lo, definindo seu nome e seus elementos.

Em Portugol, a criação de um tipo registro é feita da seguinte forma:

```
tipo nome_da_estrutura = registro
  lista_de_campos;
fimregistro;
```

Onde:

- tipo: palavra-chave da linguagem; utilizada para se criar novos tipos.
- nome_da_estrutura: nome do registro.
- Registro: palavra-chave da linguagem; indica o começo da definição da estrutura de dados heterogênea.
- lista_de_campos: a lista de elementos (membros ou campos) que fazem parte do registro; são especificados de maneira semelhante às variáveis.
- fimregistro: palavra-chave da linguagem; indica o final da definição da estrutura de dados heterogênea.

Exemplo:

```
tipo passagem = registro
  caracter: EMPRESA, ORIGEM, DESTINO, DATA, HORA,
    PASSAGEIRO, IDENTIDADE;
```

```
  real: DISTANCIA;
  inteiro: NUM_PASSAGEM, POLTRONA;
  logico: EH_MENOR;
fimregistro;
```

Utilização

Após a criação do tipo, é necessário declarar pelo menos uma variável para sua utilização, pois os dados em um programa apenas são manipulados através de variáveis, e não de tipos. Da mesma forma que é só possível ler e escrever em variáveis do tipo inteiro (e não no tipo inteiro em si), para ler e escrever em variáveis do tipo registro precisamos obrigatoriamente criar estas variáveis.

Em outras palavras, o tipo registro é apenas uma forma para a criação de novas variáveis, e estas devem ser criadas para se poder trabalhar com os registros. A criação de variáveis do tipo registro segue a sintaxe costumeira de Portugol, como mostrado a seguir, onde são criadas duas variáveis do tipo passagem.

```
passagem: PASSAGEM1, PASSAGEM2;
```

Assim como em um vetor, só podemos acessar os elementos de um registro de forma individual, ou seja, não é possível ler ou escrever todos os campos do registro de uma única vez.

Para acessar os membros do registro, utilizamos o operador de seleção (representado por .), como mostrado a seguir:

```
leia (PASSAGEM1.DISTANCIA);
escreva (PASSAGEM2.PASSAGEIRO);
```

A primeira instrução armazena uma informação de distância no campo correspondente de PASSAGEM1 e a segunda lê o nome do passageiro armazenado no registro PASSAGEM2.

Note que os campos são tratados como variáveis comuns. A única diferença é que eles são prefixados pelo nome do registro.

Exemplo

Faça um algoritmo que receba o nome, altura e sexo 10 pessoas, calcule e imprima o nome e o peso ideal da pessoa, com base nas seguintes fórmulas:

1. Para homens: (72.7 * h) – 58
2. Para mulheres: (62.1 * h) – 44.7 (h = altura)

```
inicio
  tipo dados_pessoa = registro
    caracter: NOME, SEXO;
    real: ALTURA, PESO_IDEAL;
  fimregistro;
```

```
  dados_pessoa: PESSOA;
  inteiro: CONTADOR;

  para CONTADOR de 1 até 10 passo 1 faça

    leia (PESSOA.NOME, PESSOA.ALTURA, PESSOA.SEXO);
    se PESSOA.SEXO = "M"
      então PESSOA.PESO_IDEAL <- 72.7*PESSOA.ALTURA – 58;
      senão PESSOA.PESO_IDEAL <- 62.1*PESSOA.ALTURA - 44.7;
    fimse;

    escreva (PESSOA.NOME, PESSOA.PESO_IDEAL);
  fimpara;
fim.
```

Registros em C++

A utilização de registros em C++ é bastante semelhante à utilização em Portugol. Sendo assim, nesta seção apenas mostramos como faze-lo, sem aprofundar na discussão, como foi feito em Portugol.

Declaração

Para utilizar um registro, primeiro o definimos e depois declaramos variáveis que seguem a definir. A sintaxe para a definição de um registro em C++ é mostrada a seguir:

```
struct nome_da_estrutura {
    lista_de_campos;
};
```

Onde:

- struct: palavra-chave da linguagem para se definir um tipo registro
- nome_da_estrutura: nome do registro.
- lista_de_campos: a lista de elementos (membros ou campos) que fazem parte do registro; são especificados de maneira semelhante às variáveis.

Note que ao final da definição do registro, temos que obrigatoriamente utilizar um ponto e vírgula.

Exemplo:

```
struct passagem {
  string empresa, origem, destino, data, hora,
    passageiro, identidade;
  float distancia;
  int num_passagem, poltrona;
  bool eh_menor;
};
```

Utilização

Após a definição do tipo registro, precisamos declarar variáveis desse tipo, para poder utiliza-lo. A declaração segue a sintaxe habitual de C++.

```
passagem passagem1, passagem2;
```

O acesso é idêntico a Portugol, com a utilização do operador de seleção (.).

```
cin >> passagem1.distancia;
cout << passagem2.passageiro;
```

Exemplo

Vejamos como fica o exemplo anterior em C++:

```
#include <iostream.h>
using namespace std;

int main() {
  struct dados_pessoa {
    string nome, sexo;
    float altura, peso_ideal;
  };

  dados_pessoa pessoa;
  int contador;

  for (contador = 0; contador < 10; contador++) {
    cout << "Informe nome, altura e sexo: ";
    cin >> pessoa.nome >> pessoa.altura >> pessoa.sexo;

    if ("M" == pessoa.sexo) {
      pessoa.peso_ideal = 72.7 * pessoa.altura - 58;
    }
    else {
      pessoa.peso_ideal = 62.1 * pessoa.altura - 44.7;
    }

    cout << "Nome: " << pessoa.nome << "\tPeso ideal: "
      << pessoa.peso_ideal << "\n\n";
  }
  system("pause");
  return 0;
}
```

Utilização Conjunta de Registros e Vetores

É possível a utilização conjunta de vetores e registros, quer seja um registro com um dos campos do tipo vetor, quer seja um vetor cujos elementos são registros. É ainda possível que se tenha as duas combinações, ou seja, um vetor de registros sendo que o registro possui um vetor como um de seus campos.

Registro com um Campo do Tipo Vetor

Pode haver a necessidade de construirmos um tipo registro que tenha um campo que seja do tipo vetor. Por exemplo, considere os seguintes dados de aluno em uma disciplina qualquer:

- Nome
- Matrícula
- Notas (5 notas de 0 a 20 pontos, perfazendo 100 pontos)
- Nota Final

Para representar estes dados, precisamos utilizar um registro. O último campo (notas), porém, será um tipo composto, uma vez que é um conjunto de cinco notas, que serão representadas com um vetor.

Vejamos na prática através de um exemplo: construa um algoritmo que leia os dados de um aluno conforme especificado acima e calcule e imprima sua nota final.

Portugol

```
inicio
  tipo dados_aluno = registro
    caracter: NOME;
    inteiro: MATRICULA;
    vetor [0:4] real: NOTAS;
    real: NOTA_FINAL;
  fimregistro;

  dados_aluno: ALUNO;
  inteiro: CONTADOR;

  ALUNO.NOTA_FINAL <- 0;

  leia (ALUNO.NOME, ALUNO.MATRICULA);
  para CONTADOR de 0 até 4 passo 1 faça
    leia (ALUNO.NOTAS[CONTADOR]);
    ALUNO.NOTA_FINAL <- ALUNO.NOTA_FINAL +
      ALUNO.NOTAS[CONTADOR]);
  fimpara;

  imprima (ALUNO.NOTA_FINAL);
fim.
```

C++

```
#include <iostream.h>
using namespace std;

int main() {
  struct dados_aluno {
    string nome;
    int matricula;
    float notas[5], nota_final;
  };

  dados_aluno aluno;
  int contador;

  aluno.nota_final = 0;
  cin >> aluno.nome >> aluno.matricula;

  for (contador = 0; contador < 5; contador++) {
    cin >> aluno.notas[contador];
    aluno.nota_final = aluno.nota_final +
      aluno.notas[contador];
  }
  cout << aluno.nota_final;
  system("pause");
  return 0;
}
```

Vetor de Registros

Durante um programa, também pode haver a necessidade de armazenarmos todas as informações lidas em um registro para um processamento posterior. Nesse caso, precisamos utilizar um vetor onde cada posição será um registro, de modo que todos os dados lidos sejam preservados no programa.

Veja o exemplo: construa um algoritmo que leia os dados (especificados anteriormente) de 10 alunos, calcule e imprima a nota final de cada um, a nota média da turma e o nome e matrícula dos alunos com nota final abaixo da média.

Portugol

```
inicio
  tipo dados_aluno = registro
    caracter: NOME;
    inteiro: MATRICULA;
    vetor [0:4] real: NOTAS;
    real: NOTA_FINAL;
  fimregistro;

  vetor [0:9] dados_aluno: ALUNO;
  inteiro: CONT1, CONT2;
```

```
  real: SOMA, MEDIA;
  SOMA <-0;

  para CONT1 de 0 até 9 passo 1 faça
    ALUNO[CONT1].NOTA_FINAL <- 0;

    leia (ALUNO[CONT1].NOME, ALUNO[CONT1].MATRICULA);
    para CONT2 de 0 até 4 passo 1 faça
      leia (ALUNO[CONT1].NOTAS[CONT2]);
      ALUNO[CONT1].NOTA_FINAL <- ALUNO[CONT1].NOTA_FINAL +
        ALUNO[CONT1].NOTAS[CONT2]);
    fimpara;

    imprima (ALUNO[CONT1].NOTA_FINAL);
    SOMA <- SOMA + ALUNO[CONT1].NOTA_FINAL;
  fimpara;

  MEDIA <- SOMA / 10;

  para CONT1 de 0 até 9 passo 1 faça
    se ALUNO[CONT1].NOTA_FINAL < MEDIA
      então imprima (ALUNO[CONT1].NOME,
        ALUNO[CONT1].MATRICULA);
    fimse;
  fimpara;
fim.
```

C++

```
#include <iostream.h>
using namespace std;

int main() {
  const int NUM_ALUNOS = 10;

  struct dados_aluno {
    string nome;
    int matricula;
    float notas[5], nota_final;
  };

  dados_aluno aluno[NUM_ALUNOS];
  int cont1, cont2;
  float soma, media;
  soma = 0;

  for (cont1 = 0; cont1 < NUM_ALUNOS; cont1++) {
    aluno[cont1].nota_final = 0;
    cin >> aluno[cont1].nome >> aluno[cont1].matricula;

    for (cont2 = 0; cont2 < 5; cont2++) {
```

```
      cin >> aluno[cont1].notas[cont2];
      aluno[cont1].nota_final = aluno[cont1].nota_final +
        aluno[cont1].notas[cont2];
    }
    cout << aluno[cont1].nota_final;
    soma = soma + aluno[cont1].nota_final;
  }
  media = soma / NUM_ALUNOS;

  for (cont1 = 0; cont1 < NUM_ALUNOS; cont1++) {
    if (aluno[cont1].nota_final < media) {
      cout << aluno[cont1].nome << aluno[cont1].matricula;
    }
  }
  system("pause");
  return 0;
}
```

10 – Procedimentos e Funções

Procedimentos

Um procedimento é um instrumento de programação que serve basicamente a dois objetivos:

1. Evitar que uma certa seqüência de comando que ocorra em vários locais de um algoritmo tenha que ser escrita repetidamente nestes locais.
2. Dividir e estruturar um algoritmo em partes fechadas e logicamente coerentes.

Em alguns casos, os trechos de comandos repetidos utilizam operandos (variáveis) diferentes. Neste caso, a utilização de procedimentos permite também uma única escrita dos comandos que, ao serem executados, utilizam os parâmetros ou operandos adequados a cada ocorrência.

Trata-se, pois, de um bloco de comandos que poderá ser solicitado em qualquer parte do programa.

Um procedimento permite a declaração de variável internamente. Sendo assim, podemos definir as variáveis em dois tipos, com relação ao "lugar" onde elas são declaradas:

a) Variáveis Locais: são variáveis criadas (declaradas) dentro de um procedimento e, neste caso, são reconhecidas apenas por este procedimento. Fora do mesmo, não se pode fazer referência a elas.

b) Variáveis Globais: são variáveis declaradas no início do programa e, neste caso, podem ser utilizadas tanto pelo programa quanto por qualquer procedimento declarado dentro do mesmo.

Como criar um procedimento

```
procedimento <NOME_DO_PROCEDIMENTO>;
início
     <declarações de variáveis e/ou tipos>

     comando1;
     comando2;
     ...
     comandoN;
fim;
```

Exemplo 8a:

```
início
     procedimento TROCA;
     início
          inteiro: AUX;
```

```
          AUX <- X;
          X <- Y;
          Y <- AUX;
     fim;

     leia (X,Y);
     TROCA;
     imprima (X,Y);
fim.
```

No programa acima, serão lidos valores para X e Y. Quando chamado o procedimento TROCA, o programa entra no procedimento e faz a troca de seus conteúdos, retornando o controle do programa para a próxima linha após a chamada, sendo que em seguida é executado o comando imprima, com os conteúdos das variáveis trocados, terminando o programa.

Neste exemplo, o algoritmo troca apenas o conteúdo das variáveis X e Y. Se quisermos ler as variáveis A e B e chamarmos o procedimento, não adianta nada, pois ele não vai trocá-las. Para que o procedimento execute a troca, independente da variável que o programa utiliza, temos que usar a passagem de parâmetros.

Os parâmetros de um procedimento são variáveis usadas apenas como "escopo" para a execução do mesmo. Quando o procedimento é chamado, ele substitui os parâmetros pelas variáveis que o programa utilizou para chamar o procedimento.

Vejamos como esta técnica funciona:

```
início

     inteiro: A, B;

     procedimento TROCA(inteiro: X, Y);
     início
          inteiro: AUX;

          AUX <- X;
          X <- Y;
          Y <- AUX;
     fim;

     leia (A, B);
     TROCA(A, B);
     imprima (A, B);
fim.
```

Funções

As funções têm comportamento semelhante aos procedimentos, com a diferença que, no caso das funções, elas devem retornar um valor ao programa. Como ela retorna um valor, deve ser declarado um tipo para a função também.

Vejamos um exemplo para uma função onde, sendo fornecido pelo programa o lado de um quadrado, a função devolve para a sua área:

Exemplo 9a:

```
início
     inteiro: LADO, QUADRADO;

     função CALCULA_AREA(inteiro: L): inteiro:
     início
          CALCULA_AREA <- L * L;
     fim;

     imprima ("Informe o lado do quadrado: ");
     leia (LADO);
     QUADRADO <- CALCULA_AREA(LADO);
     imprima ("A área é: ", QUADRADO);
fim.
```

C++

Criação de Métodos (Procedimentos e Funções)

Quando você cria um método, você precisa especificar o nome, a lista de parâmetros e o corpo do método.

Sintaxe para criação de métodos

Para criar um método você utiliza a seguinte sintaxe:

```
void NOME (<lista_de_parametros>)
{
     COMANDOS
}
```

Chamando Métodos

Para chamar um método utilize o nome do método seguido pela lista de parâmetros em parênteses.

Os parênteses são obrigatórios mesmo para chamar um método sem parâmetros como mostrado no seguinte exemplo:

```
NOME (<lista_de_parametros>);
```

Utilizando Variáveis Locais

Você pode incluir variáveis locais no corpo de um método, como mostrado no exemplo a seguir:

```
void EXEMPLO()
{
     int x = 1;
     long a;
     char endereco;
}
```

Conflitos de Escopo

No C++ você pode declarar uma variável local com o mesmo nome que uma variável global, mas isto pode produzir resultados inesperados. No programa abaixo, as variáveis possuem o mesmo nome (x). No entanto, como estão em escopos diferentes, a atribuição de uma não altera o valor da outra. Na medida do possível, deve-se evitar a repetição de nomes de variáveis, para aumentar a legibilidade do programa.

```
#include <iostream.h>
using namespace std;

void teste() {
     int x = -3;

     cout << x << "\n";
}

int main() {
     int x = 5;

     cout << x << "\n";

     //Criacao de um bloco de codigo
     {
          int x = 10;
          cout << x << "\n";
     }

     teste();
     return 0;
}
```

Declarando e Chamando Parâmetros

Os parâmetros permitem que informações sejam passadas de fora para dentro de um método. Quando você define um método, você pode incluir uma lista de parâmetros, entre parênteses na frente do nome do método.

Declarando Parâmetros

O seguinte exemplo mostra como declarar um método com parâmetros:

```
void MetodoComParametros (int n, char y) {
     // ...
}
```

Chamando Métodos com Parâmetros

O seguinte código mostra dois exemplos de como chamar um método com parâmetros. Em cada caso os valores dos parâmetros são colocados em parênteses.

```
MetodoComParametros (2, 'a');

int p = 7;
char c = 'c';

MetodoComParametros (p, c);
```

Passando Parâmetro por Valor

O mecanismo de passagem de parâmetro por valor é padrão. Na passagem por valor, existe uma cópia dos valores das variáveis utilizadas na chamada da função para os parâmetros formais da função. No exemplo abaixo, as variáveis x e n não ocupam a mesma posição de memória. Modificações feitas em n não alteram o valor de x, da mesma forma que modificações feitas em x não alteram o valor de n.

```
#include <iostream.h>
using namespace std;

int testePar (int n) {
     n = n + 5;
     return n;
}

int main() {
     int r, x = 7;

     cout << "Main: Valor de x antes: " << x << "\n";
     r = testePar(x);
     cout << "Main: Valor de x depois: " << x << "\n";
     cout << "Main: Valor de r: " << r << "\n";
     return 0;
}
```

Passando Parâmetro por Referência

Na passagem de parâmetros por referência (endereço), as variáveis x e n, do exemplo, ocupam a mesma posição de memória. É passado para a função o endereço da variável

utilizada na chamada da função. Assim sendo, ao se modificar x também se estará modificando n, da mesma forma que ao se modificar n também se estará modificando x.

```
#include <iostream.h>
using namespace std;

int testePar (int &n) {
     n = n + 5;
     return n;
}

void main() {
     int r, x = 7;

     cout << "Main: Valor de x antes: " << x << "\n";
     r = testePar(x);
     cout << "Main: Valor de x depois: " << x << "\n";
     cout << "Main: Valor de r: " << r << "\n";
     return 0;
}
```

11 – Exercícios

Exercícios envolvendo conceitos básicos

1. Crie uma seqüência lógica para tomar banho.

2. Faça um algoritmo para somar dois números e multiplicar o resultado pelo primeiro número.

3. Descreva com detalhes a seqüência lógica para trocar um pneu de um carro.

4. Faça um algoritmo para trocar uma lâmpada. Descreva com detalhes.

5. Identifique os dados de entrada, processamento e saída no algoritmo abaixo:

    a) Receba código da peça
    b) Receba valor unitário da peça
    c) Receba quantidade de peças
    d) Calcule o valor total da peça (quantidade x valor unitário da peça)
    e) Mostre o código da peça e seu valor total

6. Faça um algoritmo para "Calcular o estoque médio de uma peça", sendo que Estoque Médio = (Quantidade Mínima + Quantidade Máxima) / 2.

7. Teste o algoritmo anterior com dados definidos por você, utilizando a técnica do teste de mesa.

Exercícios envolvendo conceitos de variáveis

8. Declare 08 variáveis utilizando a correta sintaxe de Portugol, sendo duas de cada tipo.

9. Declare 08 variáveis utilizando a correta sintaxe de C++, sendo duas de cada tipo.

10. Na lista abaixo, assinale os nomes de variáveis inválidos e justifique por que são inválidos.

```
a) CONTA            d) DIA-MÊS-ANO
b) 1o. ALUNO        e) CANDIDATO777
c) TOTAL1           f) OLHAR43%
```

11. Num algoritmo devem ser usadas variáveis que representem o número do CPF, número de dependentes, renda anual, desconto e nome de contribuintes do Imposto de Renda. Usando nomes significativos, crie essas variáveis e declare-as de acordo com as regras de declaração de variáveis do Portugol.

12. Veja a lista de variáveis abaixo:

```
inteiro: B, TOTAL;
real: SOMA, DIFERENÇA;
caracter: NOME;
lógico: X;
```

Num algoritmo envolvendo essas variáveis seriam válidos os seguintes comandos de atribuição abaixo? (justifique)

```
a) B <- SOMA;            d) SOMA <- TOTAL;
b) NOME <- "X";          e) X <- "falso";
c) NOME <- X;            f) X <- falso;
```

13. Dê os resultados de:

```
a) A <- (25 div 4) * 6 - 3 ** 2;

A = ?
```

b) X é uma variável do tipo lógico e A, B e C são do tipo inteiro.

```
A <- 5;
B <- 3;
C <- A * B;
X <- (A > B) e (C > B) ou (C <= A**2);

X = ?
```

14. Considerando que:

```
A <- 127;
B <- 10;
C <- 5;
D <- falso;
E <- verdadeiro;
```

Qual é o valor produzido em cada uma das sentenças abaixo?

```
a) não D;
b) D e E;
c) (A > B) ou (B < C);
d) não (A < B)
e) (D e E) ou (A = B)
f) (D ou E) e (A < B)
g) A + B < C e D ou E e não D;
h) A + B * C / B = 3 e não (D ou E);
```

15. Qual é a primeira operação executada em cada um dos comandos abaixo:

```
a) X + Y - Z;
b) A + B/C**2;
```

```
c) JOAO + JOSE/JOEL;
d) X + Y + B**2 / R;
e) MARIA + JOAO + BETE * JUNIA;
f) X + Y + B**2 + R*33;
g) A * B / C * D;
```

16. Transforme estas equações algébricas em comandos em Portugol:

```
a)  E + B                  b)  A + K + D
    -----                      ----------
      C                            C

c)  A + M + X              d)  X² + 5 x A - 3
    ---------
      C + D
```

17. Admita que L, G são variáveis lógicas e A, B e C são do tipo real:

```
A <- 4;
B <- 3;
C <- 0;
G <- não L;
L <- (A > B) e (C > B) ou (C <= A ** 3);
G = ?
```

Qual é o erro encontrado no trecho de algoritmo acima? Fazendo-se a devida alteração, qual será o valor de G?

18. Determine os resultados que serão impressos depois de executado o algoritmo abaixo:

```
inicio

inteiro: NUMERO, D1, D2, D3, D4;

     NUMERO <- 1352;
     D4 <- NUMERO mod 10;
     D3 <- (NUMERO div 10) mod 10;
     D2 <- (NUMERO div 100) mod 10;
     D1 <- (NUMERO div 1000) mod 10;
     imprima (D4, D3, D2, D1);

fim.
```

19. Determine o que faz o algoritmo abaixo.

```
inicio
     inteiro: X, Y;
     leia (X, Y);
     Y <- X + Y;
     X <- Y - X;
```

```
      Y <- Y - X;
      imprima (X, Y);
fim.
```

20. Escreva um algoritmo que calcule e imprima a soma e a média entre 10, 20 e 30.

Exercícios envolvendo comandos de entrada e saída (leia/imprima)

21. Faça um programa que receba uma velocidade em metros por segundo (m/s) e converta-a para quilômetros por hora (km/h). Para realizar a conversão, basta multiplicar a velocidade em m/s por 3,6.

22. Nos Estados Unidos, a unidade de medida de temperatura é o Fahrenheit (F). A fórmula de conversão para Celsius (C) é a seguinte:

$$C = 5/9 * (F - 32)$$

Escreva um algoritmo que leia uma temperatura dada em Fahrenheit e imprima o resultado em graus Celsius.

23. Em um concurso público da prefeitura de Belo Horizonte, os candidatos fizeram provas de Língua Portuguesa, Matemática e Direito. Escreva um algoritmo que leia as notas de cada prova, calcule e imprima a média aritmética e média ponderada de um candidato, sendo que as provas possuem pesos distintos, de acordo com a tabela abaixo.

Matéria	**Peso**
Língua Portuguesa	2
Matemática	4
Direito	3

A média aritmética é a soma das notas de cada prova dividida pelo número de provas. A média ponderada é a soma de cada nota multiplicada pelo seu peso, dividida pela soma dos pesos, conforme ilustrado a seguir.

$$\frac{x_1 p_1 + x_2 p_2 + \ldots + x_n p_n}{p_1 + p_2 + \ldots + p_n}$$

24. Escreva um algoritmo que leia o saldo atual de um cliente do banco CITYBANK, leia os valores de duas retiradas e de um depósito. Calcular e imprimir:
    1. O saldo atual.
    2. O total das retiradas.
    3. O total de depósitos.
    4. O saldo final.

Exercícios envolvendo outros conceitos

25. Refaça os dois exercícios anteriores, inserindo comentários e realizando a identação.

26. Escreva um programa que leia dois números inteiros informados pelo usuário, calcule e imprima:
    - A soma dos números
    - A subtração dos números
    - A multiplicação dos números
    - O quociente inteiro da divisão
    - O resto da divisão
    - O resultado da divisão real
    - O resultado da potenciação do 1º número elevado ao 2º

Exercícios envolvendo estrutura condicional

27. Qual será o valor de L ao final da execução do trecho de algoritmo abaixo?

```
lógico: A, B, C;
real: X, Y;
inteiro: L;

A <- falso;              B <- verdadeiro;
C <- falso;              X <- 11.5;
Y <- 3.2;                X <- X + 1;

se C ou ((X + Y > 5) ou (não A e B))
     então L <- 0;
     senão L <- 1;
fimse;
```

28. Considere o seguinte algoritmo:

```
início
     lógico: B1, B2, B3;

     leia (B1, B2, B3);
     se B1
     ! então COMANDO1;
     ! senão se B2
     ! ! então se B3
     ! ! ! então COMANDO2;
     ! ! ! senão COMANDO3;
     ! ! fimse;
     ! fimse;
     fimse;
     COMANDO4;
fim.
```

Quais comandos serão executados se forem lidos os seguintes valores?

a) B1 = verdadeiro, B2 = verdadeiro, B3 = verdadeiro
b) B1 = falso, B2 = verdadeiro, B3 = falso
c) B1 = falso, B2 = verdadeiro, B3 = verdadeiro

d) Quais deverão ser os valores de B1, B2 e B3 para que somente o COMANDO4 seja executado?

29. Escreva um algoritmo que leia o nome, a idade, o salário e o sexo de uma pessoa. Imprimir a idade se for um homem e o salário se for uma mulher.

30. Tendo como dados de entrada a altura e o sexo de uma pessoa, construa um algoritmo que calcule e imprima seu peso ideal, utilizando as seguintes fórmulas:
1. Para homens: (72.7 * h) – 58
2. Para mulheres: (62.1 * h) – 44.7 (h = altura)

31. Escreva um algoritmo que leia o NOME, NÚMERO DE HORAS TRABALHADAS e CLASSE FUNCIONAL de um empregado da SNOB Confecções. Calcular e imprimir o salário líquido, sabendo que:

Classe Funcional	Salário/Hora
1	5,00
2	9,00

Salário Bruto = Horas Trabalhadas * Salário por Hora
O salário líquido é igual ao salário bruto menos 11% de INSS.

32. Escreva um algoritmo que leia o NOME e o SALÁRIO BRUTO de um funcionário. O programa deverá calcular e imprimir o salário líquido, sendo que:

Salário Bruto	Desconto
Até R$ 800,00	9% do salário bruto
De R$ 800,01 a R$ 1500,00	10% do salário bruto
Acima de R$ 1500,00	11% do salário bruto

SALÁRIO LÍQUIDO = SALÁRIO BRUTO – DESCONTO

33. Escreva um algoritmo que leia o nome, a idade e a renda familiar de um esportista do Clube Horse. Imprimir a categoria do esportista, com base na seguinte tabela:

Idade	Categoria
Até 15 anos	Infantil
De 16 a 18 anos	Juvenil
Acima de 18 anos	Adulto

Imprimir também a classe social do esportista, com base na tabela abaixo:

Renda Familiar	Classe Social
Até R$ 1.000,00	Média baixa
De R$ 1.000,01 a R$ 3.500,00	Média
Acima de R$ 3.500,00	Média alta

34. Escreva um algoritmo que leia o NOME e o TURNO de uma funcionária da MARIA DA PENHA CONFECÇÕES. Imprimir o nome e o salário da funcionária, sabendo que:

Turno	Salário
1	R$ 450,00
2	R$ 490,00
3	R$ 650,00

35. Escreva um algoritmo que leia o NOME, o CÓDIGO DO CARGO, o NÚMERO DE HORAS TRABALHADAS e o TURNO de um funcionário da PADARIA DA JUCRÉCIA. Calcular e imprimir o salário bruto, sabendo que:

Código do Cargo	Salário / Hora
1	5,00
2	8,00

Salário base = horas trabalhadas * salário por hora.

Há uma comissão por turno:

Turno	Comissão
1	3% do salário base
2	4% do salário base
3	5% do salário base

36. Escreva um algoritmo que leia o NOME do responsável e o NÚMERO DE FILHOS matriculados em uma escolinha de futebol, com mensalidade de R$ 120,00. Imprimir o valor que o responsável vai pagar, baseando-se na seguinte tabela de descontos:

Filhos Matriculados	Desconto
1	10%
2 a 3	15%
Acima de 3	20%

37. Escreva um algoritmo que leia o número do telefone, nome, número de impulsos registrados, valor total de interurbanos e tipo de telefone (1 = residencial, 2 = comercial) de um cliente da TELEMAR. Calcular e imprimir:

- Valor da tarifa básica.
  Telefone residencial: R$ 38,14
  Telefone comercial: R$ 64,69

- Valor do serviço local
  0,15 por impulso excedente, acima de 100 impulsos.

- Tarifa de interurbanos
  Valor total de interurbanos acrescidos de 5% para a Embratel.

- Valor da conta
  Soma de todos os serviços.

38. Escreva um algoritmo que leia o NOME de um aluno, o número de aulas freqüentadas e as quatro notas que ele obteve durante o curso. Calcular e imprimir:

    - A nota final do aluno.
    - A nota média do aluno.
    - O resultado (aprovado ou reprovado), sendo que, para ser aprovado, é necessário ter freqüentado 160 aulas ou mais e obter uma nota final de 60 pontos ou acima.

Exercícios envolvendo estruturas de repetição

39. Escreva um algoritmo que leia o NOME, NÚMERO DE HORAS TRABALHADAS e SALÁRIO POR HORA de 10 funcionários da NEW EMPIRE CONFECÇÕES. Calcular e imprimir o salário líquido de cada um, sendo que:

    SALÁRIO BRUTO = HORAS TRABALHADAS * SALÁRIO POR HORA
    INSS = 11% DO SALÁRIO BRUTO
    SALÁRIO LÍQUIDO = SALÁRIO BRUTO - INSS

40. Escreva um algoritmo que leia o NOME, a MODALIDADE ESPORTIVA (1 = Voley, 2 = Basquete, 3 = Futsal), a IDADE e o SEXO (M ou F) de 10 atletas do clube "LES ENFANTS". Calcular e imprimir:

    - Média de idade dos homens.
    - Média de idade das mulheres.
    - Porcentagem de mulheres matriculadas no basquete, em relação ao número de mulheres matriculadas.
    - Número de homens com idade entre 25 e 30 anos.

41. Quais os resultados impressos pelo algoritmo abaixo?

```
inicio
     lógico: X;
     inteiro: Y;

     Y <- 0;
     X <- falso;
     enquanto Y <> 6 faça
          X <- não X;
          Y <- Y + 1;
          se X
               então imprima (Y);
               senão imprima (-Y);
          fimse;
     fimenquanto;
fim.
```

42. Construir um algoritmo que leia NOME e IDADE de 300 pessoas e que calcule e imprima:

a) A soma de idade das pessoas.
b) Os nomes das pessoas com idade maior ou igual a 21 anos.

43. Construir um algoritmo que leia NOME, ALTURA e SEXO de um grupo indeterminado de pessoas e que calcule e imprima:

    a) A média de altura dos homens.
    b) A maior altura.
    c) A menor altura.

44. Elaborar um algoritmo que leia NOME, HORAS TRABALHADAS, SALÁRIO-HORA e SEXO de um grupo de operários e que calcule e imprima:

    a) Salário total dos funcionários.
    b) O maior salário, juntamente com o Nome de quem o recebeu.

45. Uma certa firma fez uma pesquisa de mercado para saber se as pessoas gostaram ou não de um certo produto lançado. Para isso, preencheu uma ficha com o SEXO do entrevistado e sua RESPOSTA (sim ou não). Sabendo-se que foram entrevistadas 2.000 pessoas, faça um algoritmo que calcule e imprima:

    a) o número de pessoas que responderam SIM.
    b) o número de pessoas que responderam NÃO.
    c) a porcentagem de pessoas do sexo feminino que responderam sim em relação ao total de mulheres entrevistadas.

46. O programador responsável pelo sistema do CEFET resolveu fazer um programa para processar as fichas com os dados dos alunos. Cada ficha contém o NOME, SEXO, IDADE, TURNO (1 para manhã, 2 para tarde e 3 para noite) e SÉRIE do aluno (1 para primeiro ano, 2 para segundo e 3 para terceiro). Construa um algoritmo que leia os dados dos alunos e calcule e imprima:

    a) O número de alunos matriculados no turno da noite.
    b) O número de alunos que estão na terceira série.
    c) O número de mulheres da segunda série.
    d) O número total de homens e mulheres.

47. A MARIA DA PENHA CONFECÇÕES mantém um cadastro de seus funcionários com os seguintes dados:

    NOME
    CLASSE FUNCIONAL
    IDADE
    HORAS TRABALHADAS
    SEXO

    Há também uma tabela de salário/hora para cada classe, conforme descrição abaixo:

Classe	Salário / Hora
1	10,00

2	15,00
3	18,00
4	25,00

Construa um algoritmo que leia as fichas dos 350 funcionários e que calcule e imprima:

a) O salário de cada funcionário, juntamente com seu nome, sabendo-se que:
SALÁRIO = HORAS TRABALHADAS * SALÁRIO/HORA
b) O maior salário e o nome do(a) funcionário(a) que o recebeu.
c) A idade da mulher mais velha e seu nome.
d) O total da folha de pagamento.

48. A MARIA DA PENHA CONFECÇÕES decidiu fazer um levantamento em relação aos 3823 candidatos que se apresentaram para preenchimento de vagas no seu quadro de funcionários. Supondo que você seja o programador responsável pelo programa deste levantamento, construa um algoritmo que:

1º) Leia as 3823 fichas contendo, cada uma, os seguintes dados:

a) número de inscrição do candidato
b) idade
c) sexo
d) experiência no serviço (SIM ou NÃO).

2º) Calcule:

a) O número de candidatos do sexo feminino.
b) O número de candidatos do sexo masculino.
c) A idade média dos homens que já tem experiência no serviço.
d) Porcentagem de homens que têm idade inferior a 35 anos e experiência no serviço.
e) Número de mulheres que têm idade superior a 35 anos e experiência no serviço.
f) A menor idade entre as mulheres que já tem experiência no serviço.

3º) Imprima:

a) O número de inscrição de cada mulher pertencente ao grupo descrito no item e.
b) O que foi calculado em cada item acima especificado.

49. Em um concurso para auxiliar de justiça foram recebidas inscrições de candidatos de ambos os sexos e a partir do segundo grau completo. Sabendo que cada candidato preencheu uma ficha que contém o seu GRAU DE INSTRUÇÃO (1 = segundo grau, 2 = terceiro grau) seu NOME e SEXO (F = feminino, M = masculino), construa um algoritmo que calcule e imprima:

a) O total de candidatos.
b) O número de candidatos do sexo masculino e do sexo feminino.
c) O número de mulheres com apenas o segundo grau.

d) A porcentagem de homens com terceiro grau em relação ao total de homens inscritos.

50. Construa um algoritmo que leia as fichas dos funcionários da MARIA DA PENHA CONFECÇÕES contendo os seguintes dados:

MATRICULA
NOME DO FUNCIONÁRIO
CLASSE (1 para quem ganha 16,00 por hora, 2 para quem ganha 20,00 por hora)
HORAS NORMAIS TRABALHADAS
HORAS EXTRAS TRABALHADAS

O programa deverá calcular e imprimir para cada funcionário:

MATRICULA
NOME
SALÁRIO DE HORAS NORMAIS
SALÁRIO DE HORAS EXTRAS
DEDUÇÃO DE INSS
SALÁRIO LÍQUIDO

Sabe-se que:

SALÁRIO DE HORAS NORMAIS = HORAS NORMAIS TRABALHADAS * SALÁRIO/HORA
SALÁRIO DE HORAS EXTRAS = HORAS EXTRAS TRABALHADAS * SALÁRIO/HORA * 1.30;

A dedução de INSS é de 8% do salário total (SALÁRIO DE HORAS NORMAIS + SALÁRIO DE HORAS EXTRAS). O salário líquido é encontrado subtraindo-se a dedução de INSS do salário total.

51. Havendo vários números de telefone de três cidades (SÃO PAULO, RIO DE JANEIRO E BELO HORIZONTE) conforme figura abaixo:

03134442633	Escolinha da Tia Mariêta
02124425544	Ana Maria do Nascimento
01133334433	Astrofôncia Raftopoulos de Paula
03133334400	Asdrúbal Costa
01134446688	Paula de Paula

Construa um algoritmo que leia o número do telefone (junto com prefixo, conforme acima) e o nome do proprietário. Sabendo-se que o prefixo 011 = São Paulo, 021 = Rio de Janeiro e 031 = Belo Horizonte, o algoritmo deverá imprimir o nome do proprietário, o telefone (sem prefixo) e o nome de sua cidade, conforme abaixo:

Escolinha da Tia Mariêta	34442633	Belo Horizonte
Ana Maria do Nascimento	24425544	Rio de Janeiro
Astrofôncia Raftopoulos de Paula	33334433	São Paulo

52. Escreva um algoritmo que leia um conjunto de 50 fichas contendo a ALTURA e o código do SEXO de uma pessoa (1 = masculino, 2 = feminino) e que calcule e imprima:

a) A maior e a menor altura da turma.
b) A média de altura das mulheres.
c) A média de altura da turma.

53. O sistema de avaliação de uma determinada disciplina obedece aos seguintes critérios: Durante o semestre são dadas três notas. A nota final é obtida pela média aritmética das notas dadas durante o curso. É considerado aprovado o aluno que obtiver nota final superior ou igual a 60 e que tiver comparecido a um mínimo de 40 aulas.

Faça um algoritmo que:

a) Leia um conjunto de dados contendo o número da matrícula, as três notas e a freqüência (número de aulas freqüentadas) de 100 alunos.

b) Calcule e imprima:
- A nota final de cada aluno, juntamente com seu número de matrícula e o resultado (aprovado ou reprovado).
- A maior nota da turma.
- A nota média da turma.
- O total de alunos reprovados.
- A porcentagem de alunos reprovados por infreqüência (dentre os alunos reprovados).

54. Uma universidade deseja fazer um levantamento a respeito de seu concurso vestibular. Para cada curso, foram considerados os seguintes dados:

- CÓDIGO DO CURSO
- NÚMERO DE VAGAS
- NÚMERO DE CANDIDATOS DO SEXO MASCULINO
- NÚMERO DE CANDIDATOS DO SEXO FEMININO

Faça um algoritmo que leia os dados de cada curso, considerando o FLAG sendo o código de curso menor ou igual a zero e que calcule e imprima:

a) O número de candidatos de cada curso, juntamente com o código do curso e também a porcentagem de candidatos do sexo feminino para o curso.
b) O código do curso com mais candidatos por vaga, juntamente com o número de candidatos por vaga do mesmo.
c) Total geral de candidatos.

55. Um comerciante deseja fazer o levantamento do lucro das mercadorias que ele comercializa. Para isso, montou uma ficha (uma para cada mercadoria), com o NOME, o PREÇO DE COMPRA e PREÇO DE VENDA das mesmas. Construa um algoritmo que calcule quantas mercadorias proporcionam lucro menor que 10%,

quantas proporcionam lucro entre 10% e 20%; e quantas proporcionam lucro maior que 20%. Calcular e imprimir também o total de compra e o total de venda.

56. Calcule o imposto de renda de um grupo de contribuintes considerando:

    a) Os dados de cada contribuinte: NÚMERO DE CPF, NÚMERO DE DEPENDENTES e RENDA ANUAL serão lidos.
    b) Para cada contribuinte será feito um desconto de R$ 600,00 por dependente.
    c) Os valores da alíquota para cálculo do imposto são:

Renda Líquida	Alíquota
Até R$ 20.000,00	isento
De R$ 20.000,01 até R$ 50.000,00	5%
De R$ 50.000,01 até R$ 100.000,00	10%
Acima de R$ 100.000,00	15%

    d) O flag será CPF igual a zero.
    e) O desconto por dependente deve ser abatido do valor da renda anual, antes de se calcular o imposto devido.
    RENDA LÍQUIDA = RENDA ANUAL - DESCONTOS

57. Uma loja deseja fazer o balancete do mês baseando-se no valor das notas fiscais de mercadorias vendidas. Fazer um algoritmo para ler um conjunto indeterminado de fichas contendo, cada uma, o valor da nota fiscal. Calcule e imprima:

    a) O número de notas fiscais abaixo de R$ 1.000,00
    b) O número de notas fiscais entre R$ 1.001,00 e R$ 5.000,00
    c) O número de notas fiscais acima de R$ 5.000,00.
    d) O número total de notas fiscais.
    e) O valor total de vendas no mês.

58. A TELEMAR mantém mensalmente os seguintes dados para cada cliente:

    - Número do telefone.
    - Tipo do telefone (1 = residencial, 2 = comercial).
    - Número de impulsos registrados para chamadas locais.
    - Valor total cobrado pelos interurbanos sem a taxa do FNT (Fundo Nacional de Telecomunicações)
    - Número de serviços despertador prestados
    - Número de telegramas passados
    - Valor total dos telegramas cobrados pelos CORREIOS.

    Escreva um algoritmo que:

    A) Leia estes dados e calcule e imprima para cada cliente:

    1 - Tarifa básica
    Telefone residencial = R$ 38,14
    Telefone comercial = R$ 64,69

2 - Serviço Local
R$ 0,15 por impulso excedente (acima de 100 impulsos).

3 - Serviço de Interurbano
Valor dos interurbanos acrescido de 8% para o FNT.

4 - Serviço Despertador
R$ 0,16 por serviço prestado

5 - Telegrama Fonado
Valor dos telegramas mais R$ 4,20 por telegrama passado.

B) Calcule e imprima também:
1 - Para cada assinante o número de seu telefone e o valor de sua conta mensal.
2 - O total arrecadado pela TELEMAR.
3 - O valor da maior conta e o telefone do assinante que a pagou.

59. Um clube recreativo oferece cursos em três modalidades esportivas: natação, voleyball e handball. Sabendo que cada usuário pode estar matriculado em um curso somente, construa um algoritmo que:

    1) Leia várias fichas contendo, cada uma:

    - Nome do usuário
    - Tipo do usuário (s = sócio, d = dependente)
    - Modalidade (1 = natação, 2 = voleyball, 3 = handball)
    - Sexo (f = feminino, m = masculino)

    2) Calcule e imprima:

    a) O número de inscritos em cada modalidade por sexo;
    b) A porcentagem de sócios matriculados na natação em relação ao total de sócios;
    c) A porcentagem de mulheres matriculadas no handball em relação ao total de mulheres matriculadas;
    d) O número de dependentes do sexo masculino matriculados no voleyball;
    e) A porcentagem de sócios do sexo masculino matriculados no voleyball em relação ao total de pessoas matriculadas nessa modalidade.
    f) O número total de pessoas matriculadas.

60. As costureiras da MARIA DA PENHA CONFECÇÕES recebem, mensalmente, além do salário normal, uma comissão por peças fabricadas. Elas estão divididas em três classes:

1 - para as que ganham R$ 450,00 por mês;
2 - para as que ganham R$ 720,00 por mês;
3 - para as que ganham R$ 980,00 por mês;

A tabela de comissões é dada a seguir:

PEÇAS FABRICADAS COMISSÃO

Até 60	5% do salário
De 61 até 120	7.5% do salário
Acima de 120	12% do salário

Construa um programa que leia a ficha de cada funcionária contendo, cada uma, seu nome, classe e número de peças fabricadas e calcule e imprima:
a) O salário total e o nome de cada funcionária.
b) O número de funcionárias de cada classe.
c) O total da folha de pagamento.
d) O total de peças fabricadas.

61. Faça um algoritmo, que leia várias fichas, contendo, cada uma, o nome e o peso de um atleta. A última ficha tem nome igual a "FIM" e não deve ser considerada. Calcule e imprima a média de peso de todos os atletas e o nome do atleta mais pesado.

62. Numa certa loja de eletrodomésticos, o comerciário encarregado da seção de televisores recebe, mensalmente, um salário fixo mais comissão. Essa comissão é calculada em relação ao tipo e ao número de televisores vendidos por mês, obedecendo à tabela abaixo:

Tipo do Aparelho de TV	Quantidade Vendida	Comissão
Em cores	Maior ou igual a 10	10,00 por aparelho
	Menor que 10	5,00 por aparelho
Preto e Branco	Maior ou igual a 20	4,00 por aparelho
	Menor que 20	2,00 por aparelho

Todo funcionário tem um desconto de 8% sobre seu salário fixo para o INSS, e o seu salário bruto equivale ao salário fixo mais comissões. O salário líquido é calculado da seguinte forma:
SALÁRIO LIQUIDO = (FIXO + COMISSÕES - INSS).

Sabendo-se que existem 20 empregados nesta seção, desenvolva um algoritmo que leia, para cada comerciário, o número de sua inscrição, o valor do salário fixo, o número de televisores em cores e o número de televisores preto e branco vendidos; calcule e imprima o número de inscrição de cada empregado, seu salário bruto e seu salário líquido.

63. Uma pesquisa sobre algumas características físicas da população em uma determinada região coletou os seguintes dados, referentes a cada habitante, para serem analisados:

    - Sexo
    - Cor dos olhos (azuis, verdes, castanhos)
    - Cor dos cabelos (louros, castanhos, pretos)
    - Idade

Cada habitante tem seus dados em uma ficha. Fazer um algoritmo que determine e imprima:

a) A maior idade dos habitantes;
b) A porcentagem de indivíduos do sexo feminino cuja idade está entre 18 e 35 anos e que tenha olhos verdes e cabelos louros em relação ao total de mulheres;
c) O total de indivíduos do sexo masculino que tenham idade superior a 21 anos.

Obs.: Considere como flag IDADE <= 0.

64. Uma empresa calcula os salários dos seus 250 empregados da seguinte maneira:

SALÁRIO BRUTO = HORAS TRABALHADAS * SALÁRIO POR HORA
SALÁRIO LIQUIDO = SALÁRIO BRUTO - INSS

SALÁRIO BRUTO	INSS
Até R$ 500,00	8,5%
De R$ 500,01 até R$ 1700,00	9,0%
Acima de R$ 1700,00	10,0%

Existem na empresa três faixas de salário:

A - para os que recebem R$ 8,00 por hora.
B - para os que recebem R$ 13,00 por hora.
C - para os que recebem R$ 19,00 por hora.

Construa um algoritmo que leia várias fichas contendo, cada uma, o nome do funcionário, a sua faixa de salário (A, B ou C) e o número de horas trabalhadas por mês.

Calcule e imprima:

a) Para cada funcionário, seu nome, o salário bruto e o salário líquido;
b) O número de funcionários de cada faixa.
c) O total da folha de pagamento.

65. Numa fábrica trabalham homens e mulheres divididos em três classes:

A - os que fazem até 30 peças por mês.
B - os que fazem de 31 a 35 peças por mês.
C - os que fazem mais de 35 peças por mês.

A classe A recebe salário mínimo. A classe B recebe salário mínimo e mais 3% do salário mínimo por peça, acima das 30 iniciais. A classe C recebe salário mínimo mais 5% do salário mínimo por peça acima das 30 iniciais.
Sabendo que o salário mínimo é de R$ 350,00, fazer um algoritmo que:

a) Leia várias fichas contendo, cada uma:

- O número do operário
- O número de peças fabricadas por ele no mês
- O sexo do operário

b) Calcule e imprima:

- O salário de cada operário
- O total da folha mensal de pagamento da fábrica
- O número total de peças fabricadas por mês
- A média de peças fabricadas pelos homens em cada classe
- A média de peças fabricadas pelas mulheres em cada classe
- O número e sexo do operário ou operária de maior salário (supondo que não existe empate)

Obs.: a última ficha, que servirá de flag, contém o número de operário igual a zero.

66. O IBGE (Instituto Brasileiro de Geografia e Estatística) fez uma pesquisa com 2.500 pessoas em Belo Horizonte, para avaliar o nível de instrução em função da renda familiar das pessoas. Para cada entrevistado, foi preenchida uma ficha com os seguintes dados:

NOME:	IDADE:	RENDA FAMILIAR:
GRAU DE INSTRUÇÃO:	0 para analfabeto 1 para primário	2 para secundário 3 para graduados

SEXO: F para feminino, M para masculino

Construir um algoritmo que leia os dados dos entrevistados, calcule e imprima:

a) A porcentagem de analfabetos em relação à amostra (total de pesquisados).
b) Média de idade dos homens analfabetos.
c) Porcentagem de homens graduados em relação ao total de homens.
d) Total de mulheres com renda familiar acima de R$ 100.000,00 e que não possuem graduação.
e) O nome e a idade da mulher mais nova com graduação.

67. Em uma eleição para Presidente, concorreram os candidatos:

1 – Lula
2 – Heloísa Helena
3 – Cristovam Buarque
4 – Geraldo Alckmin

Para cada voto foram computados o sexo do eleitor (M = masculino e F = feminino), o grau de instrução (1 = primário, 2 = secundário, 3 = superior) mais a identificação de voto que corresponde a:

1 – Lula	4 – Geraldo Alckmin
2 – Heloísa Helena	5 – voto nulo
3 – Cristovam Buarque	6 – voto em branco

Desenvolva um algoritmo que leia a identificação de voto, o sexo do eleitor e o grau de instrução. Calcule e imprima:

1) O número do candidato vencedor.

2) O número de votos de cada candidato.
3) A porcentagem de mulheres que votaram em Heloísa Helena, em relação ao total de pessoas que votaram nela.
4) A porcentagem de eleitores de nível superior em relação ao total de eleitores.
5) A porcentagem de eleitores de nível primário que votaram em Lula em relação ao total de eleitores.

68. Construa um algoritmo para um programa que:

1) Leia várias fichas contendo cada uma:
- O nome de um atleta
- O seu peso
- O sexo
- A idade
- Sua matrícula

2) Calcule e imprima:

a) O peso e a matrícula do atleta mais pesado;
b) A média de idade dos atletas do sexo feminino;
c) A matrícula e o peso do atleta masculino de menor peso.

69. O IBOPE fez uma pesquisa para avaliar a audiência das emissoras de televisão de Belo Horizonte. Para isto, foram preenchidas fichas com o sexo do telespectador e o numero do canal que ele mais assiste (2, 4, 7, 12). Faca um algoritmo para um programa que leia o sexo e o canal do telespectador e calcule e imprima:

a) O canal mais assistido e o número de pessoas que o escolheram.
b) A porcentagem de mulheres que assistem o canal 12, em relação ao total de pessoas que assistem esse canal.
c) O número de pessoas que assistem a cada canal.

Obs.: A última ficha, para indicar fim de dados, contém canal menor ou igual a zero.

70. Uma universidade deseja conhecer melhor os candidatos que se inscreveram para o seu concurso vestibular. Supondo que você seja o programador responsável por esta pesquisa, construa um algoritmo que:

1o.) Leia as fichas dos 15.000 candidatos, contendo, cada uma:

a) seu número de inscrição
b) idade
c) sexo
d) número de tentativas anteriores (A para nenhuma vez, B para uma vez, C para mais de uma vez).

2o.) Calcule e imprima:

a) O numero de candidatos do sexo masculino;
b) O número de candidatos do sexo feminino;

c) A idade média dos homens que já tentaram vestibular mais de uma vez;
d) A menor idade entre as mulheres que nunca tentaram vestibular;
e) Porcentagem de candidatos que estão fazendo vestibular pela primeira vez, em relação ao total de candidatos.

71. A MARIA DA PENHA CONFECÇÕES mantém o seguinte registro para cada funcionário:

    - nome
    - salário
    - idade
    - sexo
    - endereço

Construir um algoritmo que leia as fichas de todos os seus funcionários e em seguida calcule e imprima:

a) Maior e menor salário.
b) Nome e salário da mulher mais bem remunerada.
c) Nome e idade do funcionário mais novo.
d) Considerando que o salário mínimo é de R$ 350,00, determinar quantas pessoas ganham mais de 20 salários mínimos, bem como o total que a empresa gasta com estes funcionários (total de seus salários).

72. Deseja-se fazer uma pesquisa a respeito do consumo mensal de energia elétrica em uma determinada cidade. Para isto, são fornecidos os seguintes dados:

    - Preço do kWh consumido;
    - Número do consumidor;
    - Quantidade de kWh consumidos durante o mês;
    - Código do tipo de consumidor (residencial, comercial, industrial).

O número do consumidor menor ou igual a zero deve ser usado como flag. Fazer um programa que:

    - Leia os dados descritos acima
    - Calcule e imprima:
      a) para cada consumidor o seu número e total a pagar;
      b) o maior consumo verificado;
      c) o menor consumo INDUSTRIAL verificado;
      d) o total do consumo para cada um dos três tipos de consumidores;
      e) a média de consumo COMERCIAL.

73. Construa um algoritmo que calcule e imprima o SALÁRIO LÍQUIDO e SALÁRIO BRUTO dos empregados da MONSIEUR SILVA INFORMÁTICA, sabendo que:

    - Serão lidas as fichas dos empregados com o NOME, MATRÍCULA, NÚMERO DE HORAS TRABALHADAS e CLASSE.

- SALÁRIO BRUTO = HORAS TRABALHADAS * SALÁRIO/HORA
  INSS = 8.5% DO SALÁRIO BRUTO
  SALÁRIO LÍQUIDO = SALÁRIO BRUTO - INSS

- Há uma tabela de SALÁRIO/HORA para cada CLASSE:

CLASSE	SALÁRIO/HORA
1	15,00
2	22,00
3	35,00

Calcule e imprima também o número de funcionários de cada classe e o total de salário bruto.

Obs.: a última ficha, que servirá de flag, contém matricula menor ou igual a zero.

Exercícios envolvendo vetores

74. Faça um algoritmo que:

a) Preencha um vetor com 30 ocorrências de zeros;
b) Preencha este mesmo vetor com 1, 2, 3 ... até 30.

75. Dada a tabela de valores abaixo, calcule e imprima:

a) Soma total dos valores.
b) Média dos valores.

TABELA

10	20	30	40	50

76. Faça um algoritmo que:

a) Crie um vetor de tamanho 10;
b) Preencha cada posição do vetor com zero;
c) Preencha cada posição do vetor com um número que seja igual ao quadrado do índice dessa posição (por exemplo, o terceiro elemento seria 9, pois 9 é o quadrado do índice da 3a. posição).
d) Imprima os elementos do vetor que têm índice par.

77. Faça um algoritmo que:

Leia uma variável de 100 elementos (vetor) alfanuméricos (nomes de cidades) e verifique se existem elementos iguais a "BELO HORIZONTE". Se existirem, imprimir as posições que eles estão armazenados.

78. Faça um algoritmo que:

a) Leia dois vetores com o mesmo número de elementos (numéricos) de 50 posições.

b) Calcule e imprima outros dois vetores que são a soma e a diferença (feitas casa por casa) dos dois vetores lidos.

79. Faça um algoritmo que determine o maior e o menor elemento e a média dos elementos de um vetor de 100 posições.

80. Faça um algoritmo que leia um grupo de fichas contendo datas no formato DD MM AA e imprima estas datas por extenso. Considere como flag DIA menor ou igual a zero.

Exemplo:
10 08 06 (10 de agosto de 2006

Considere que todas as datas são do século XXI.

81. Uma empresa tem os salários dos seus funcionários padronizados por classe (veja tabela):

Classe	Salário
1	800,00
2	1.600,00
3	2.000,00
4	2.100,00
5	2.200,00
6	2.500,00
7	2.600,00
8	2.700,00
9	3.000,00
10	4.000,00

Supondo que a empresa tem 1500 funcionários, faça um algoritmo que leia a CLASSE e o NOME do funcionário e imprima seu nome juntamente com seu salário.

82. Construir um algoritmo que leia um vetor de 50 elementos inteiros e que determine o maior elemento deste vetor.

83. Os funcionários da MARIA DA PENHA CONFECÇÕES têm seus salários de acordo com a classe, conforme tabela:

Classe	Salário
1	800,00
2	1000,00
3	1100,00
4	1500,00
5	2500,00
6	4000,00
7	6000,00

Construir um algoritmo que leia o NOME e a CLASSE de cada um dos 350 funcionários e que calcule e imprima:

a) O nome de cada funcionário e seu salário.
b) O total de funcionários de cada classe.
c) Qual a classe tem mais funcionários e quantos funcionários esta classe tem.

84. Construir um algoritmo que leia as temperaturas registradas em BELO HORIZONTE de janeiro a julho de 2006 (212 dias) e calcular e imprimir:

a) a maior temperatura.
b) a menor temperatura.
c) a temperatura média.
d) o número de dias em que a temperatura esteve abaixo da média.

85. Sabendo-se que um tabuleiro de XADREZ tem 64 casas, fazer um algoritmo que calcule o somatório de grãos de trigo que caberiam neste tabuleiro casso fossem colocados 1 grão na primeira casa, 2 grãos na segunda, 4 na terceira, etc, ou seja, em cada casa será colocado o dobro de grãos colocado na casa anterior. Imprimir o número de grãos em cada casa na ordem inversa em que foram colocados.

86. A MARIA DA PENHA CONFECÇÕES mantém um cadastro de seus funcionários com as seguintes informações:

NOME: CLASSE: IDADE: SEXO:
HORAS NORMAIS TRABALHADAS: HORAS EXTRAS TRABALHADAS:

Há também uma tabela de salário/hora para cada classe de funcionários, conforme abaixo:

Classe	Salário / Hora
1	3,50
2	5,00
3	8,00
4	11,00
5	14,00
6	18,00
7	23,00
8	30,00

Construa um algoritmo para um programa que leia os registros dos 300 funcionários e que calcule e imprima:

a) O salário bruto de cada funcionário, juntamente com seu nome, sabendo que:

Salário Bruto = Salário de Horas Normais + Salário de Horas Extras
Salário de Horas Normais = Horas Trabalhadas * Salário/Hora
Salário de Horas Extras = Horas Extras Trabalhadas * Salário/Hora * 1.30

b) O maior salário bruto e o nome do funcionário que o recebeu.
c) O nome e a idade da mulher mais velha.
d) O maior salário feminino.

e) O total da folha de pagamento.

Obs.: utilize um vetor para armazenar os salários/hora.

87. Existem 7 indivíduos que são candidatos à presidência da república, conforme tabela. Cada um deles é designado por um número de ordem que servirá para a indicação de voto de cada eleitor.

1 - Cristovam Buarque
2 - Geraldo Alckmin
3 - Heloísa Helena
4 - José Maria Eymael
5 - Luciano Bivar
6 - Luiz Inácio Lula da Silva
7 - Rui Costa Pimenta

Fazer um algoritmo que:

Leia um número indeterminado de votos (cada voto contém o número de cada candidato).

Calcule e imprima:

- O número de votos de cada candidato, juntamente com seu nome.
- Imprima o NOME do vencedor.

88. Fazer um algoritmo que leia um vetor de 50 elementos inteiros e que calcule e imprima:

    a) Média dos elementos.
    b) Maior e menor elemento.
    c) Número de vezes que aparece o número 10 no vetor.

Exercícios envolvendo procedimentos e funções

89. O que será impresso ao ser executado o seguinte algoritmo?

```
início
        inteiro: X1, X2, X3;

        procedimento S(inteiro: A);
        início
                inteiro: T;

                T <- 1;
                T <- 1 + A;
                A <- T;
        fim;
```

```
X1 <- 2;
S(X1);
X2 <- 5;
X3 <- 3;
S(X3);

imprima (X1, X2, X3);
```

fim.

90. Construa um procedimento e um programa que o utilize, que tenha como objetivo receber uma data e informar se esta data é válida ou não.

91. Construa um procedimento que, a partir de uma data no formato DD, MM, AAAA, informe qual é a sua data juliana (dia do ano seguido do ano). Por exemplo, para a data 22/02/1968, a data juliana será 53/1968, porque este é o 53o dia do ano.

92. Construa um procedimento que, dadas duas datas, verifique qual é a data mais recente.

93. Construa um algoritmo com os seguintes procedimentos:

    - Verificar se uma data é válida.
    - Calcular a data juliana da mesma.
    - Verificar qual dentre duas datas é a mais recente.

    O algoritmo deverá ter um menu de opções para o usuário informar qual dos três procedimentos acima ele quer executar. Para terminar o mesmo, o usuário deverá informar a opção zero.

94. Uma empresa calcula o dígito verificador dos códigos de seus clientes da seguinte maneira:

    - Um número de cinco algarismos é separado em valores absolutos.
    - O primeiro algarismo é multiplicado por 6, o segundo por 5, o terceiro por 4, o quarto por 3 e o quinto por 2.
    - É realizada uma soma destes produtos.
    - A soma é dividia por 11 e o resto da divisão é analisado:
        - Caso o resto seja igual a 0, o dígito verificador será 1.
        - Caso o resto seja igual a 1, o dígito verificador será 0.
        - Sendo o resto maior que 1, o dígito verificador é igual a: 11 - Resto.

Exemplo:
O dígito verificador de 26131 será:

SOMA = 2*6 + 6*5 + 1*4 + 3*3 + 1*2 = 57
O resto será:
57 / 11 = 5 e sobram 2 (resto).
Neste caso o dígito será 11 - resto = 11 - 2 = 9.

Construa um algoritmo que leia vários números de cinco algarismos, considerando como flag número menor ou igual a 0 e que chame uma função para calcular o dígito verificador.

95. Supondo que não existam as funções pré-definidas MOD e DIV (que calculam, respectivamente o resto e quociente de uma divisão inteira entre dois números), construa duas funções que façam os trabalhos destas duas, elas deverão se chamar RESTO e QUOCIENTE. Mas, para tanto, você somente pode utilizar as quatro operações básicas da matemática: multiplicação, divisão, soma e subtração.

96. Construa uma função que receba como parâmetros: um vetor de 50 posições contendo caracteres e um caracter. A função dever retornar o número de ocorrências desse caracter informado dentro do vetor.

Anexos

Anexo I – Quadro Comparativo entre Portugol e C++

	Portugol	C++

Início e fim de programa	inicio
	.
	.
	.
	.
	fim.
		#include <iostream.h>
		using namespace std;
		
		int main() {
		.
		.
		.
		return <num_inteiro>;
		}

Declaração de variável do tipo inteiro	inteiro: NOME_DA_VARIÁVEL;	int nome_da_variável;
Declaração de variável do tipo real	real: NOME_DA_VARIÁVEL;	float nome_da_variável;
Declaração de variável do tipo string (cadeia de caracteres)	caracter: NOME_DA_VARIÁVEL;	string nome_da_variável;
Declaração de variável do tipo lógico	logico: NOME_DA_VARIÁVEL;	bool nome_da_variável;
Comando de atribuição	<-	=
Comentários	{ }	// ou /* */
Comparação de igualdade	=	==
Constantes de texto	"M", "f", "ab", "TESTE" etc.	"M", "f", "ab", "TESTE" etc.
Maior que	>	>
Menor que	<	<

Maior ou igual	>=	>=
Menor ou igual	<=	<=
Diferente	<>	!=
Operações matemáticas básicas	+, -, *, /	+, -, *, /
Quociente da divisão	div	/
Resto da divisão	mod	%
Potenciação	**	Função pow (math.h)
e (and) lógico	e	&&
ou (or) lógico	ou	||
não (not) lógico	não	!
Comando de saída	imprima (mensagem ou variáveis);	cout << mensagem ou variáveis;
Comando de entrada	leia (variável);	cin >> variável;

Estrutura condicional simples	se condição_lógica
	  então comandos;
	fimse;
		if (condição_lógica) {
		  comandos;
		}

Estrutura condicional composta	se condição_lógica
	  então comandos1;
	  senão comandos2;
	fimse;
		if (condição_lógica) {
		  comandos1;
		}
		else {
		  comandos2;
		}

Estrutura condicional múltipla	escolha (valor_inteiro)
	  caso valor_1:
	  comando_1;
	  comando_2;
	  ...
	  comando_n;
	  pare;
	  ...
	  caso valor_n:
		switch (valor_inteiro) {
		  case valor_1:
		  comando_1;
		  comando_2;
		  ...
		  comando_n;
		  break;
		  ...
		  case valor_n:

	comando_1; comando_2; ... comando_n; pare; padrão: comando_1; comando_2; ... comando_n; fimescolha;	comando_1; comando_2; ... comando_n; break; default: comando_1; comando_2; ... comando_n; }

Estrutura de repetição com variável de controle	para CONTADOR de 1 até 20 passo 1 faça
	comando_1;
	comando_2;
	...
	comando_n;
	fimpara;
		for (contador = 0; contador < 20; contador++) {
		comando_1;
		comando_2;
		...
		comando_n;
		}

Estrutura de repetição com teste no início	enquanto condição_lógica faça
	comando_1;
	comando_2;
	...
	comando_n;
	fimenquanto;
		while (condição_lógica) {
		comando_1;
		comando_2;
		...
		comando_n;
		}

Estrutura de repetição com teste no final	repita comando_1; comando_2; ... comando_n; até condição_lógica;	do { comando_1; comando_2; ... comando_n; } while (condição_lógica);

Declaração de Estrutura de Dados Homogênea (vetor)	vetor [li:ls] tipo_dado: nome_do_vetor; Ex.: vetor [1:10] real: SALARIOS; Obs.: Começa do índice 1	tipo_dado nome_do_vetor[numero_elementos]; Ex.: float salários[10]; Obs.: Começa do índice 0

Criação de Estruturas de Dados Heterogêneas (registro)	tipo nome_da_estrutura = registro
	  lista_de_campos;
	fimregistro;
		struct nome_da_estrutura {
		  lista_de_campos;
		};

Declaração de variável do tipo estrutura	nome_da_estrutura: NOME_DA_VARIÁVEL;	nome_da_estrutura nome_da_variável;
Utilização de variável do tipo estrutura	NOME_DA_VARIÁVEL.CAMPO_DA_ESTRUTURA;	nome_da_variável.campo_da_estrutura;

Anexo II – Debug de Programas no Dev-C++

Debug (ou depuração, em português) é o processo de encontrar bugs (defeitos) em programas de computador.

O debug serve para se encontrar erros de lógica no programa. Em outras palavras, o programa deve pelo menos compilar e começar a executar para que o processo de debug possa ser utilizado. Deste modo, apenas programas sintaticamente corretos podem ser "debugados".

Os programas que possuem erros sintáticos (como um comando digitado de forma incorreta ou faltando ponto e vírgulas obrigatórios) não serão compilados e a ferramenta (em nosso caso, o Dev-C++) dá um erro de compilação indicando qual foi o problema encontrado.

O processo de depuração varia em cada caso, mas existem vários passos que são normalmente executados, preferencialmente com a ajuda de uma ferramenta de depuração para analisar a causa do bug.

Os passos a serem seguidos são:

- Descobrir a existência do bug (é necessário saber que o programa está executando de forma errada para poder corrigi-lo. Por isso, é importante realizar testes no programa.

- Isolar o código que causa o bug (depois de descobrir que o programa possui um defeito, devemos localizar onde este defeito acontece.

- Identificar a causa do bug (encontrado o local onde ocorre o problema, devemos identificar sua(s) causas.

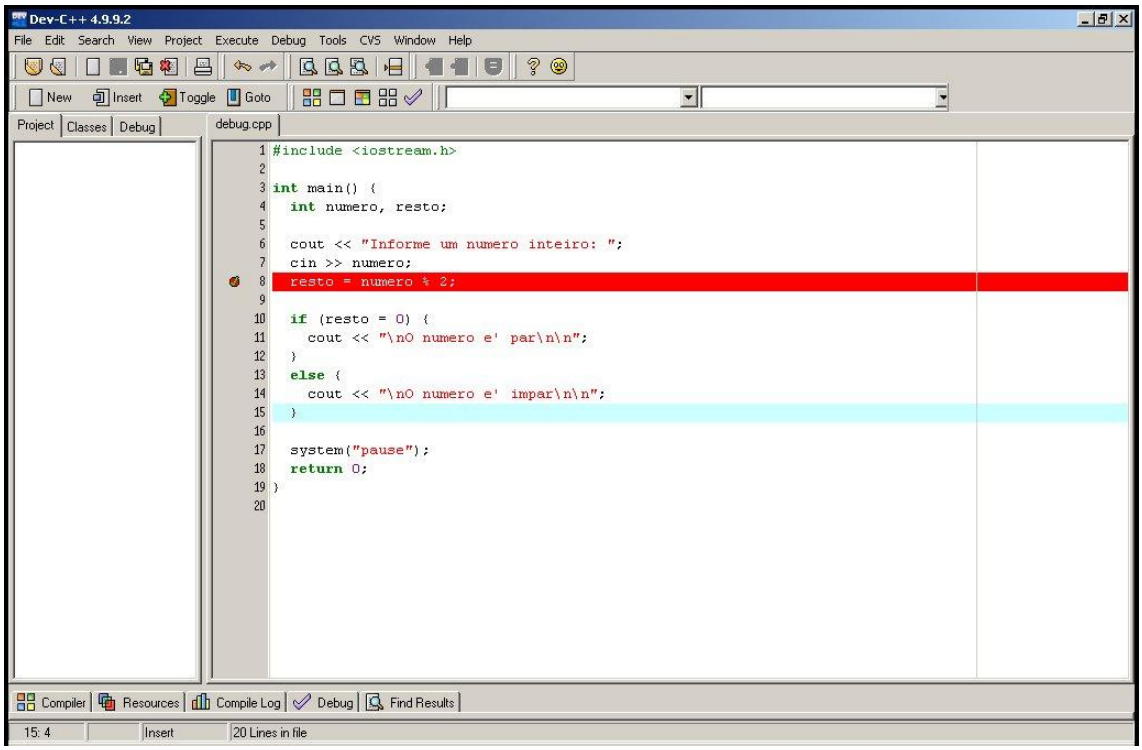
- Determinar uma correção para o bug (encontrar a melhor maneira de resolver o problema, atacando sua(s) causas.

- Aplicar a correção e testar a correção (resolver o problema de acordo com a solução encontrada e testar novamente o programa, para verificar se o erro foi realmente corrigido.

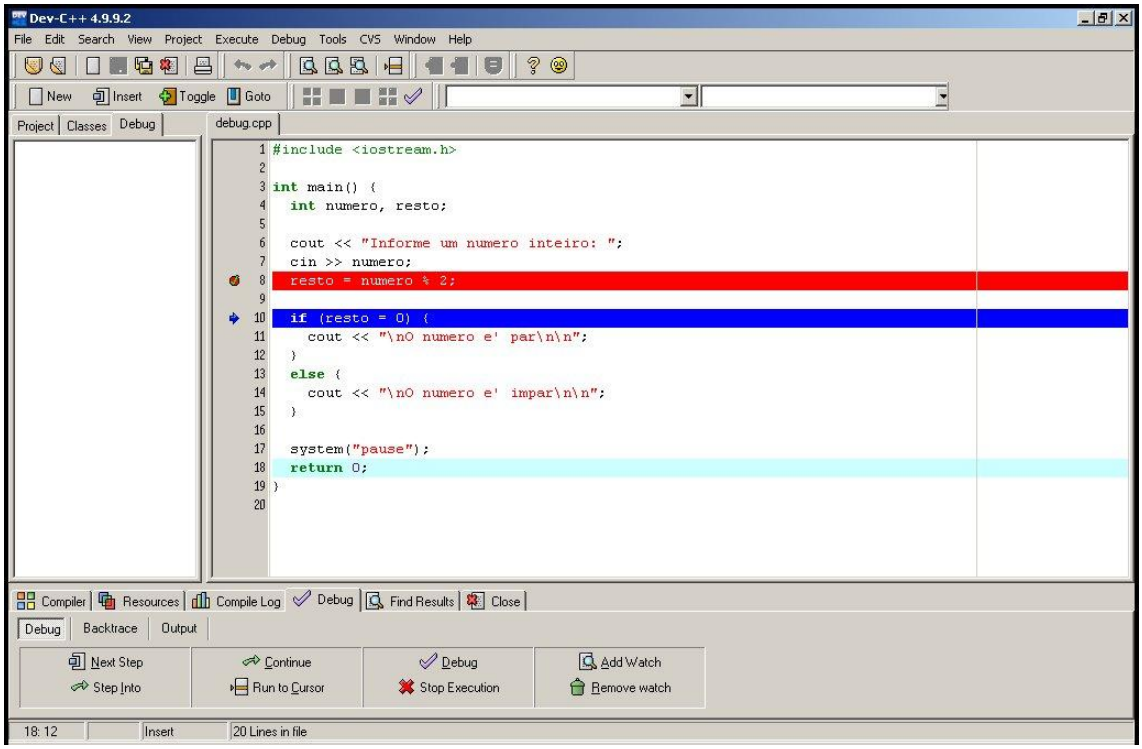
O Dev-C++ possui algumas facilidades para debug de programas:

- Breakpoint (o breakpoint serve para parar o fluxo de execução de um programa em um ponto específico do código. Por exemplo, ao se colocar um breakpoint em uma linha qualquer de um programa, este será executado até atingir tal linha. Ao alcança-la, o programa terá seu fluxo interrompido. Ele estará pausado e poderá ser executado passo a passo deste ponto em diante. Para colocar um breakpoint em um programa no Dev-C++, posicione o cursor em qualquer ponto da linha onde o breakpoint deve ser colocado e pressione as teclas Ctrl+F5. Para retirar um breakpoint, posicione o cursor na linha que o contém e pressione as

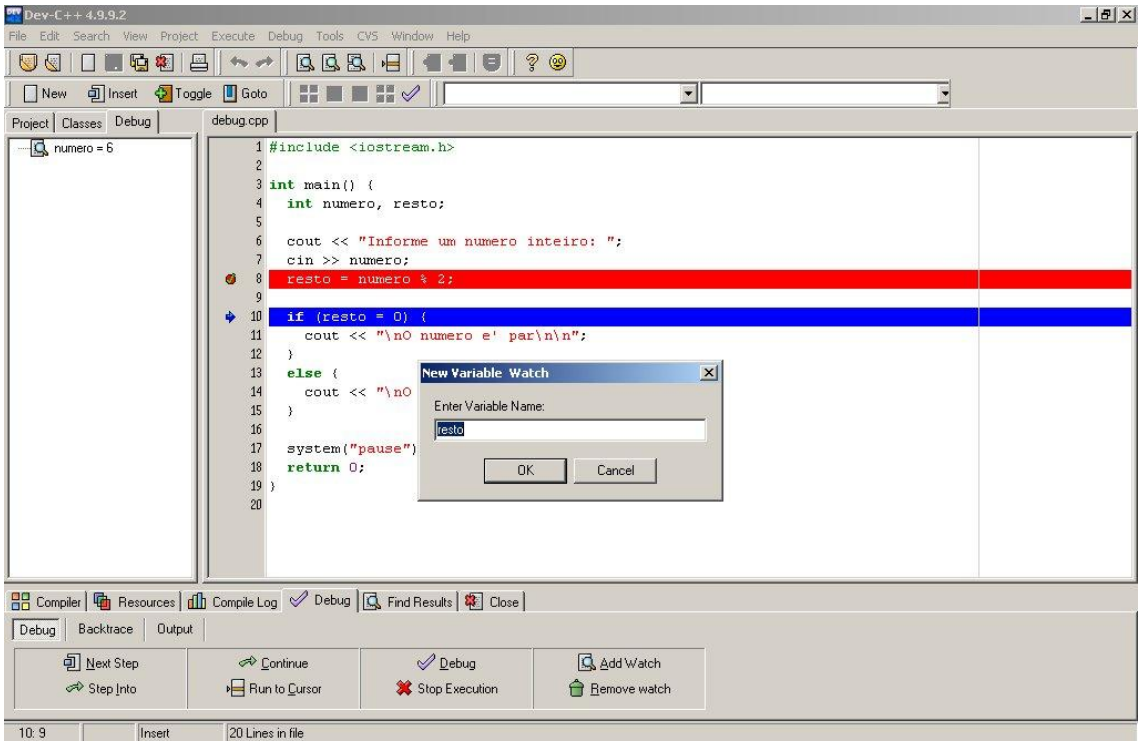
teclas Ctrl+F5 novamente. Na tela a seguir é mostrado um exemplo de programa com breakpoint.



- Executar passo a passo → o Dev-C++ permite que executemos um programa passo a passo, de modo que possamos ver exatamente quais instruções são executadas. Para isso, precisamos iniciar a execução do programa pressionando a tecla F8 (o programa deve ter sido previamente compilado). Quando fizermos isto, o programa irá executar até encontrar o primeiro breakpoint e, a partir desse ponto, poderá ser executado passo a passo pressionando-se a tecla F7. Toda vez que esta tecla for pressionada a próxima instrução do código do programa será executada. Abaixo, mostra-se uma tela com uma execução passo a passo. A linha com o comando que será executado fica em destaque na cor azul.



- Verificar valor de variável → é possível checar os valores que uma variável assume durante a execução passo a passo do programa. Para isso devemos colocar um "watch" na variável. No Dev-C++ isso é feito posicionando o cursor na variável e pressionando a tecla F4. Quando fizermos isso, apareça uma caixa de diálogo como mostrada abaixo (New Variable Watch), com o nome da variável já selecionado (caso o nome não apareça automaticamente, temos a opção de digita-lo). Após colocarmos o watch na variável, podemos monitora-la na aba Debug, que aparece à esquerda da tela. Caso a tela Watch não apareça, acesse o menu View, opção Project/Class Browser. Um watch só funciona enquanto o programa estiver em execução. Na tela a seguir é mostrado como adicionar um watch e a aba Debug com as variáveis sendo monitoradas.


Dev-C++ 4.9.9.2
File Edit Search View Project Execute Debug Tools CVS Window Help
New Insert Toggle Goto
Project Classes Debug
numero = 6
debug.cpp
#include <iostream.h>
int main() {
int numero, resto;
cout << "Informe um numero inteiro: ";
cin >> numero;
resto = numero % 2;
if (resto = 0) {
cout << "\nO numero e' par\n\n";
}
else {
cout << "\nO
}
system("pause")
return 0;
}
New Variable Watch
Enter Variable Name:
resto
OK
Cancel
Compiler Resources Compile Log Debug Find Results Close
Debug Backtrace Output
Next Step
Step Into
Continue
Run to Cursor
Debug
Stop Execution
Add Watch
Remove watch
10: 9
Insert
20 Lines in file

Bibliografia

GRILLO, Maria Célia A. Turbo Pascal 5.0/5.5. 2.ed. Rio de Janeiro: Livros Técnicos e Científicos, 1991.

GUIMARAES, Ângelo de Moura; LAGES, Newton A. De Castilho. Algoritmos e estruturas de dados. 11.ed. Rio de Janeiro: Livros Técnicos e científicos, 1985.

Forbellone, A. L. V., e Eberspacher, H. F., Lógica de Programação – A Construção de Algoritmos e Estruturas de Dados, 3ª ed., Makron Books, 2005.

MORAES, Paulo Sérgio de. Apostila de Lógica de Programação. Unicamp - Centro de Computação – DSC. 2000.

RIOS, Rosângela Silqueira Hickson. Aprenda a programar em C, C++ e C#. 1.ed. Rio de Janeiro: Campus, 2002.

ROBERTO, Pablo Alexandre. Apostila LED I. Faculdade Infórium de Tecnologia.

SILVA, Eli Lopes da. Apostila de Lógica de Programação. Faculdade Infórium de Tecnologia.