VLADEMIRO LANDIM JUNIOR

INTRODUÇÃO A PROGRAMAÇÃO USANDO A LINGUAGEM HARBOUR.

FORTALEZA, CEARÁ

2016

VLADEMIRO LANDIM JUNIOR

INTRODUÇÃO A PROGRAMAÇÃO USANDO A LINGUAGEM HARBOUR.

Uma introdução aos aspectos básicos da programação de computadores usando a linguagem Harbour como ferramenta de aplicação dos conceitos aprendidos.

Área de concentração: Introdução a Programação, Linguagem Harbour, Computação, Informática, Algoritmos.

FORTALEZA, CEARÁ

2016

LANDIM, Vlademiro.
Introdução a programação usando a linguagem Harbour.
/ Vlademiro Landim Junior. 2016.
63p.;il. color. enc.

VLADEMIRO LANDIM JUNIOR

INTRODUÇÃO A PROGRAMAÇÃO USANDO A LINGUAGEM HARBOUR.

Área de concentração: Introdução a Programação, Linguagem Harbour, Computação, Informática, Algoritmos.

Aos meus Pais.

Agradecimentos

Agradeço a Paulo César Toledo e a todos que participam do fórum Clipper On Line (http://www.pctoledo.com.br/forum). A solidariedade e a atenção de todos vocês foi essencial para a conclusão desse trabalho. **Com certeza não vou poder citar todos** pois muitos me ajudaram mesmo sem saber, e na pressa do momento eu acabei esquecendo o nome da pessoa ou até mesmo nem lendo o nome. Segue abaixo uma lista muito parcial (a ordem não reflete a importância, fui me lembrando e digitando):

- Paulo César Toledo
- José Quintas
- "Maligno"
- Fladimir
- Rochinha
- "Asimoes"
- Claudio Soto
- Vailton
- Pablo Cesar
- janio
- Stanis Luksys
- Jairo Maia
- Itamar M. Lins Jr.
- "RobertoLinux"

- porter
- “alxsts”
- Eolo
- “paiva_dbdc”
- “rbonotto”
- “vagucs”
- “sygecon”
- “Imatech”

Caso alguém encontre algum erro nesse material envie um e-mail com as correções a serem feitas para vlad@altersoft.net com o título “E-book Harbour”. Eu me esforcei para que todas as informações contidas neste livro estejam corretas e possam ser utilizadas para qualquer finalidade, dentro dos limites legais. No entanto, os usuários deste livro, devem testar os programas e funções aqui apresentadas, por sua própria conta e risco, sem garantia explícita ou implícita de que o uso das informações deste livro, conduzirão sempre ao resultado desejado, sendo que há uma infinidade de fatores que poderão influir na execução de uma mesma rotina em ambientes diferentes.

“Suponham que um de vocês tenha um amigo e que recorra a ele à meia-noite e diga ‘Amigo, empreste-me três pães, porque um amigo meu chegou de viagem, e não tenho nada para lhe oferecer’. E o que estiver dentro responda: ‘Não me incomode. A porta já está fechada, e eu e meus filhos já estamos deitados. Não posso me levantar e lhe dar o que me pede’. Eu lhes digo: Embora ele não se levante para dar-lhe o pão por ser seu amigo, por causa da importunação se levantará e lhe dará tudo o que precisar. Por isso lhes digo: Peçam, e lhes será dado; busquem, e encontrarão; batam, e a porta lhes será aberta”

(Jesus Cristo, Filho do Deus vivo - Evangelho de Lucas 11:5-9)

Resumo

Esse livro busca ensinar os princípios de programação utilizando a linguagem Harbour. Ele aborda os conceitos básicos de qualquer linguagem de programação, variáveis, tipos de dados, estruturas sequenciais, estruturas de decisão, estruturas de controle de fluxo, tipos de dados complexos, funções, escopo e tempo de vida de variáveis. Como a linguagem utilizada é a linguagem Harbour, o presente estudo busca também apresentar algumas particularidades dessa linguagem como comparação de *strings* e macro substituição. Palavras-chave: Introdução a Programação, Linguagem de programação, Linguagem Harbour, Sistemas de Informação, xBase, Clipper.

Abstract

This book seeks to teach the principles of programming using the language Harbour . It covers the basics of any programming language , variables , data types , sequential structures , decision structures , flow control structures , complex data types , functions, scope and lifetime of variables. As the language is the language Harbour , this study also seeks to present some peculiarities of this language as a comparison textit string and macro replacement.

Keywords: . . .

Lista de Figuras

Lista de Tabelas

Sumário

1 Constantes e Variáveis

A fonte do saber não está nos livros, ela está na realidade e no pensamento. Os livros são placas de sinalização; o caminho é mais antigo, e inguém pode fazer por nós a viagem da verdade.

A.-D.Sertillanges - A vida intelectual

Objetivos do capítulo

- Entender o que é uma constante e uma variável bem como a diferença entre elas.
- Criar variáveis de memória de forma adequada.
- Compreender a importância de criar nomes claros para as suas variáveis e constantes.
- Atribuir valores as suas variáveis.
- Diferenciar declaração de variável de inicialização de variável.
- Receber dados do usuário.
- Realizar operações básicas com as variáveis.

1.1 Constantes

Constantes são valores que não sofrem modificação durante a execução do programa. No capítulo anterior nós trabalhamos, mesmo sem saber o nome formal, com dois tipos de constantes : as constantes numéricas e as constantes caracteres (*strings*). Veja novamente dois exemplos dessas constantes :

```
1 PROCEDURE Main
2
3    ? "Hello World" // ‘‘Hello World’’ é uma constante caractere
4    ? 2 // 2 é uma constante numérica
5
6 RETURN
```

Um programa de computador trabalha com várias constantes durante a sua execução, algumas delas possuem um valor muito especial, como por exemplo 3.1415 ou uma *string* que representa o código de uma cor (por exemplo : “FFFFFF”). Não é uma boa prática de programação simplesmente colocar esse símbolo dentro do seu código e usá-lo. Você pode replicar : “Tudo bem, eu posso criar um comentário para esse número ou string!”, mas e se esse valor se repetir em dez pontos diferentes do seu código ? Será que você vai comentar dez vezes ? E será que compensa comentar dez vezes ? E se houver uma mudança futura na constante, você saberá onde atualizar os comentários ?

Esse problema constuma acontecer mais com números do que com strings. Quando um número (que possui uma importância dentro do problema) é simplesmente colocado dentro do programa ele é chamado de “número mágico”. “Número mágico” é uma expressão irônica para indicar as constantes numéricas que simplesmente aparecem nos programas. Segundo Kernighan e Pike, “todo número pode ser mágico e, se for, deve receber seu próprio nome. Um número bruto no programa fonte não indica a sua importância ou derivação, tornando o programa mais difícil de entender e modificar.” (KERNIGHAN; PIKE, 2000, p. 21).

Uma forma de evitar o problema com números mágicos é dar-lhes um nome. Eles ainda serão números, mas serão “chamados” por um nome específico. O Harbour resolve esse problema através de um recurso chamado de “constante simbólica” ou“constante de pré-processador”. Esse recurso recebe esse nome porque as constantes recebem um nome especial (um símbolo) e, durante a fase anterior a geração do programa (conhecida também como “fase de processamento”), esse nome é convertido para o seu respectivo valor numérico. Como esse valor é atribuído em tempo de compilação, as constantes de pré-processador estão fora da procedure Main. No exemplo a seguir nós temos uma constante simbólica chamada VOLUME que vale 1000.

```
1 #define VOLUME 10000
2 PROCEDURE Main
3
```

```
4     ? VOLUME // Vai imprimir 10000
5
6  RETURN
```

Essa forma também é usada tradicionalmente por programadores da Linguagem C. Note que, no exemplo acima, caso o volume mude um dia. Eu terei que recompilar o programa mas a alteração do volume dela será feita em apenas um ponto. O valor de uma constante é sempre atribuída em tempo de compilação, por isso a mudança no valor de uma constante requer a recompilação do programa.

Dica 1

A expressão **"tempo de compilação"** refere-se ao momento em a instrução foi definida pelo computador. Nesse caso específico, as constantes simbólicas são definidas antes do programa principal ser gerado, ou seja, durante a compilação do programa.
Vamos acompanhar o processo que se esconde durante a definição de uma constante simbólica. Vamos tomar como ilustração a listagem abaixo :

```
1  #define VOLUME 12
2  PROCEDURE Main
3
4        ? VOLUME
5
6  RETURN
```

Quando você executa o hbmk2 para compilar e gerar o seu programa executável ele transforma a listagem acima em outra (sem que você veja) antes de compilar. A listagem que ele gera é a listagem abaixo :

```
1
2  PROCEDURE Main
3
4        ? 12
5
6  RETURN
```

Só agora ele irá compilar o programa. Note que ele "retirou" as definições e também substituiu VOLUME por 12. Não se preocupe, pois o seu código fonte permanecerá inalterado, ele não irá alterar o que você fez.

Dica 2

Você deve ter cuidado quando for criar constantes dentro de seu programa. Basicamente são dois os cuidados :

1. Escolha um símbolo que possa substituir esse valor. Evite os "números mágicos" e atribua nomes de fácil assimilação também para as constantes caracteres. Por exemplo, um código de cor pode receber o próprio nome da cor.

2. Decida sabiamente quando você quer que um valor realmente não mude dentro de seu programa. Geralmente códigos padronizados mundialmente, valores de constantes conhecidas (o valor de Pi) e notas de copyright. Siglas de estados ou percentuais de impostos não são bons candidatos porque podem vir a mudar um dia.

Um detalhe importantíssimo é que uma constante simbólica **não é *case insensitive*** como os demais itens da linguagem. Se você criar uma constante com o nome de PI, e quando for usar escrever Pi, então o programa irá reportar um erro.

Dica 3

Use as constantes de pré-processador para evitar números mágicos. Esse recurso não se aplica somente em números. Posso definir strings também [a].
Exemplos :

```
#define PI_VALUE  3.141516
#define COLOR_WHITE "FFFFFF"

PROCEDURE Main

    ... Restante do código ...
```

Não esqueça de definir suas constantes antes da procedure Main, conforme o exemplo acima.

[a] Além de números e strings posso criar constantes simbólicas com outros tipos da linguagem Harbour que ainda não foram abordados. No momento certo nós daremos os exemplos correspondentes, o importante agora é entender o significado das constantes simbólicas.

Dica 4

Como as constantes são sensíveis a forma de escrita, habitue-se a escrever as constantes apenas com letras maiúsculas. Esse é um bom hábito adotado por todos os programadores de todas as linguagens de programação.

1.1.1 A hora em que você tem que aceitar os números mágicos

Nem todos os números mágicos devem ser convertidos em constantes simbólicas. Existe um caso muito especial onde você deve se conformar e manter aquele número misterioso

dentro do seu código. Isso acontece na hora de você aplicar determinadas fórmulas de terceiros dentro do seu código, como por exemplo a fórmula que converte uma temperatura medida em graus Celsius para Fahrenheit (F = C * 1.8 + 32). Você não precisa transformar 1.8 e 32 em constantes simbólicas, pois tais valores não serão encontrados isolados dentro do seu código. A mesma coisa também serve para conversões de valores inteiros para valores percentuais, quando durante o processo ocorre uma divisão por 100. Nesse caso, não faz sentido transformar esse 100 em uma constante simbólica. Coordenadas da tela também não devem ser convertidas em números mágicos (veremos o que é isso no capítulo 11). Fique atento, pois no processo de desenvolvimento de programas existem muitas regras de clareza mas sempre tem a maior de todas que é usar o bom senso.

1.1.2 Grupos de constantes

As vezes é importante você ter todas as suas constantes agrupadas em um único arquivo. Isso evita ter que ficar redigitando todas as constantes em vários lugares diferentes. Por exemplo, vamos supor que você criou uma constante para simbolizar o código da cor branca (“FFFFFF”), conforme abaixo :

```
1  #define BRANCO "FFFFFF"
2  PROCEDURE Main
3
4     ? BRANCO  // Imprime FFFFFF
5
6  RETURN
```

Suponha também que surgiu a necessidade de se criar constantes para as outras cores. Conforme o exemplo a seguir :

```
1  #define BRANCO "FFFFFF"
2  #define VERMELHO "FF0000"
3  #define AZUL "0000FF"
4  #define AMARELO "FFFF00"
5  PROCEDURE Main
6
7     ? BRANCO
8     ? VERMELHO
9     ? AZUL
10    ? AMARELO
11
12 RETURN
```

No futuro você terá várias constantes de cores. Mas, e se você resolver usar em outro programa ? Você teria que redefinir todas as constantes no novo programa. E se você

tiver que criar uma nova constante para incluir uma nova cor ? Você teria que alterar todos os programas.

Para evitar esse problema você pode criar um arquivo de constantes de cores e chamá-lo de “cores.ch”. Veja a seguir o conteúdo de um suposto arquivo chamado de “cores.ch”.

```
1 #define BRANCO "FFFFFF"
2 #define VERMELHO "FF0000"
3 #define AZUL "0000FF"
4 #define AMARELO "FFFF00"
```

Para usar essas constantes no seu programa basta salvar esse arquivo no mesmo local do seu programa e fazer assim :

```
1 #include "cores.ch"
2 PROCEDURE Main
3
4    ? BRANCO
5    ? VERMELHO
6    ? AZUL
7    ? AMARELO
8
9 RETURN
```

Pronto, agora você pode usar suas constantes de uma forma mais organizada.

Dica 5

Os arquivos include servem para organizar as suas constantes. Procure agrupar as suas constantes em arquivos includes, por exemplo : cores.ch, matematica.ch, financas.ch, etc. Sempre use a extensão “ch” para criar seus arquivos include. Se por acaso você não tem uma classificação exata para algumas constantes, não perca tempo, crie um arquivo chamado “geral.ch” ou “config.ch” (por exemplo) e coloque as suas constantes nesse arquivo.
O Harbour possui os seus próprios arquivos “include”. Você sempre pode utilizar esses arquivos caso deseje. Eles tem outras funções além de organizar as constantes em grupos, veremos mais adiante essas outras funções.

1.2 Variáveis

Variáveis são locais na memória do computador que recebem uma identificação e armazenam algum dado específico. O valor do dado pode mudar durante a execução do

programa. Por exemplo, a variável **nTotalDeItens** pode receber o valor 2,4, 12, etc. Daí o nome "variável". O criador da linguagem C++ introduz assim o termo :

> basicamente, não podemos fazer nada de interessante com um computador sem armazenar dados na memória [...]. Os "lugares" nos quais armazenamos dados são chamados de *objetos*. Para acessar um objeto necessitamos de um nome. Um objeto com um nome é chamado de variável (STROUSTRUP, 2012, p. 62).

1.2.1 Criação de variáveis de memória

Uma variável deve ser definida antes de ser usada. O Harbour permite que uma variável seja definida (o jargão técnico é "declarada") de várias formas, a primeira delas é simplesmente criar um nome válido e definir a ele um valor, conforme a listagem 1.1. Note que a variável fica no lado esquerdo e o seu valor fica no lado direito. Entre a variável e o seu valor existe um sinal (:=) que representa uma atribuição de valor. Mais adiante veremos as outras formas de atribuição, mas habitue-se a usar esse sinal (o nome técnico é "operador") pois ele confere ao seu código uma clareza maior.

Outro detalhe importante que gostaríamos de chamar a atenção é o seguinte : entre a linha 7 e a linha 8 existe uma linha não numerada. Não se preocupe pois isso significa que a linha 7 é muito grande e a sua continuação foi mostrada embaixo, mas no código fonte ela é apenas uma linha. Essa situação irá se repetir em algumas listagens apresentadas nesse livro.

Listing 1.1: Criação de variáveis de memória

```
1
2  /*
3     Criação de variáveis de memória
4  */
5  PROCEDURE Main
6
7     cNomeQueVoceEscolher := "Se nós nos julgássemos a nós mesmos,
          jamais seríamos condenados."
8     ? cNomeQueVoceEscolher
9
10 RETURN
```

.:Resultado:.

```
Se nós nos julgássemos a nós mesmos, jamais seríamos condenados.
```

Quando você gerou esse pequeno programa, e o fez na pasta "pratica" então você deve ter recebido do compilador algumas "mensagens estranhas", conforme abaixo :

.:Resultado:.

```
> hbmk2 var0
hbmk2: Processando script local: hbmk.hbm
hbmk2: Harbour: Compilando módulos...
Harbour 3.2.0dev (r1507030922)
Copyright (c) 1999-2015, http://harbour-project.org/
Compiling 'var0.prg'...
var0.prg(6) Warning W0001  Ambiguous reference
   'CNOMEQUEVOCEESCOLHER'
var0.prg(7) Warning W0001  Ambiguous reference
   'CNOMEQUEVOCEESCOLHER'
Lines 9, Functions/Procedures 1
Generating C source output to '.hbmk\win\mingw\var0.c'... Done.
hbmk2: Compilando...
hbmk2: Linkando... var0.exe
```

Estamos nos referindo as mensagens :

.:Resultado:.

```
var0.prg(6) Warning W0001  Ambiguous reference
   'CNOMEQUEVOCEESCOLHER'
var0.prg(7) Warning W0001  Ambiguous reference
   'CNOMEQUEVOCEESCOLHER'
```

Mensagens com a palavra "Warning" não impedem o seu programa de ser gerado, apenas chamam a atenção para alguma prática não recomendada ou um possível erro. Não se preocupe com isso, pois essas mensagens serão explicadas mais adiante ainda nesse capítulo. Por ora pode digitar normalmente os exemplos que virão e ignorar esses avisos estranhos.

1.2.2 Escolhendo nomes apropriados para as suas variáveis

Você não pode escolher arbitrariamente qualquer nome para nomear as suas variáveis de memória. Alguns critérios devem ser obedecidos, conforme a pequena lista logo abaixo :

1. O nome **deve** começar com uma letra ou o caracter "_" (*underscore* [1]).
2. O nome de uma variável **deve** ser formado por uma letra ou um número ou ainda por um *underscore*. Lembre-se apenas de não iniciar o nome da variável com um dígito (0...9), conforme preconiza o ítem 1.
3. Não utilize letras acentuadas ou espaços para compor o nome da sua variável.

O exemplo a seguir (listagem 1.2) mostra alguns nomes válidos para variáveis :

[1] Em algumas publicações esse caractere recebe o nome de *underline*

Listing 1.2: Nomes válidos para variáveis

```
1
2  /*
3    Nomes válidos
4  */
5  PROCEDURE Main
6
7     nOpc := 100
8     _iK := 200  // Válido, porém não aconselhado (Começa com
          underline).
9     nTotal32 := nOpc + _iK
10    ? "nOpc = " , nOpc
11    ? "_iK = " , _iK
12    ? "nTotal32 = " , nTotal32
13
14 RETURN
```

.:Resultado:.

```
nOpc =          100
_iK =          200
nTotal32 =          300
```

Dica 6

Apesar do caractere *underscore* ser permitido no início de uma variável essa prática é desaconselhada por vários autores.

É aconselhável que uma variável NÃO tenha o mesmo nome de uma "palavra reservada" da linguagem Harbour. Palavras reservadas são aquelas que pertencem a própria sintaxe de uma determinada linguagem de programação e o compilador "reserva" para si o direito exclusivo de usá-las. Nós utilizamos aspas para exprimir o termo "palavra reservada" da linguagem Harbour, porque o nome das instruções e comandos do Harbour não são "reservadas".

Dica 7

Em determinadas linguagens, como a Linguagem C, você não pode nomear uma variável com o mesmo nome de uma palavra reservada. O Clipper (ancestral do Harbour) também não aceita essa prática, de acordo com (RAMALHO, 1991, p. 68). O Harbour não reclama do uso de palavras reservadas, mas reforçamos que você **não** deve adotar essa prática.

Listing 1.3: Uso de palavras reservadas

```
1
2  /*
3    Evite usar palavras reservadas
```

```
4  */
5  PROCEDURE Main
6
7     PRIVATE := 100 // PRIVATE é uma palavra reservada
8     REPLACE := 12  // REPLACE também é
9     TOTAL := PRIVATE + REPLACE // Total também é
10    ? TOTAL
11
12 RETURN
```

.:Resultado:.

```
2        112
```

Dica 8

(SPENCE, 1994, p. 11) nos informa que o Clipper possui um arquivo (chamado "reserved.ch") na sua pasta include com a lista de palavras reservadas. O Harbour também possui esse arquivo, mas a lista está em branco. Você pode criar a sua própria lista de palavras reservadas no Harbour. Caso deseje fazer isso você precisa seguir esses passos :

1. Edite o arquivo reserved.ch que vem com o Harbour na sua pasta include.
2. Inclua em uma lista (um por linha) as palavras que você quer que sejam reservadas.
3. No seu arquivo de código (.prg) você deve incluir (antes de qualquer função) a linha : *#include "reserved.ch"*.

O Harbour irá verificar, em tempo de compilação, a existência de palavras que estejam nesse arquivo e irá barrar a compilação caso o seu código tenha essas palavras reservadas (por você). Você pode, inclusive, criar a sua própria lista de palavras reservadas independente de serem comandos ou não.

O programa a seguir (Listagem 1.4) exemplifica uma atribuição de nomes inválidos à uma variável de memória.

Listing 1.4: Nomes inválidos para variáveis

```
1  /*
2    Nomes inválidos
3  */
4  PROCEDURE Main
5
6     90pc := 100 // Inicializa com um número
7
```

```
8 RETURN
```

1.2.3 Seja claro ao nomear suas variáveis

Tenha cuidado quando for nomear suas variáveis. Utilize nomes indicativos daquilo que elas armazenam. Como uma variável não pode conter espaços em branco, utilize caracteres *underscore* para separar os nomes, ou então utilize uma combinação de letras maiúsculas e minúsculas [2].

Listing 1.5: Notações

```
1  /*
2  Notações utilizadas
3  */
4  PROCEDURE Main
5
6     Tot_Nota := 1200 // Variável numérica (Notação com underscores)
7                      // que representa o total da nota fiscal
8
9     NomCli := "Rob Pike" // Variável caracter (Notação hungara)
10                          // que representa o nome do cliente
11
12
13 RETURN
```

Mais adiante estudaremos os tipos de dados, mas já vamos adiantando : procure prefixar o nome de suas variáveis com o tipo de dado a que ela se refere. Por exemplo: **nTot** [3] representa uma variável numérica, por isso ela foi iniciada com a letra "n". Já **cNomCli** representa uma variável caractere (usada para armazenar *strings*), por isso ela foi iniciada com a letra "c". Mais adiante estudaremos com detalhes outros tipos de variáveis e aconselhamos que você siga essa nomenclatura: "n" para variáveis numéricas, "c" para variáveis caracteres (strings), "d" para variáveis do tipo data, "l" para variáveis do tipo lógico, etc. **Mas sempre é bom usar o bom senso** : se você vai criar um código pequeno para exemplificar uma situação, você pode abdicar dessa prática.

Dica 9

Essa parte é tão importante que vamos repetir : "apesar de ser simples criar um nome para uma variável, você deve priorizar a clareza do seu código. O nome de uma variável deve ser informativo, conciso, memorizável e, se possível, pronunciável" (KERNIGHAN; PIKE, 2000, p. 3). Procure usar nomes descritivos, além disso, procure manter esses nomes curtos, conforme a listagem 1.6.

Listing 1.6: Nomes curtos e concisos para variáveis

[2] Essa notação é conhecida como Notação Hungara.

[3] Essa notação é chamada por Rick Spence de "Notação Hungara Modificada"

```
/*
  NOMES CONSISOS

  Adaptado de (KERNIGHAN, p.3, 2000)
*/
PROCEDURE Main

   /*
   Use comentários "//" para acrescentar informações sobre
   a variável, em vez de deixar o nome da variável grande
      demais.
   */
   nElem := 1 // Elemento corrente
   nTotElem := 1200 // Total de elementos

RETURN
```

Evite detalhar demais conforme a listagem 1.7.

Listing 1.7: Nomes longos e detalhados demais

```
/*
  NOMES GRANDES DEMAIS PARA VARI?VEIS. EVITE ISSO!!!

  Adaptado de (KERNIGHAN, p.3, 2000)
*/
PROCEDURE Main

   nNumeroDoElementoCorrente := 1
   nNumeroTotalDeElementos := 1200

RETURN
```

Complementando : escolha nomes auto-explicativos para as suas variáveis, evite comentar variáveis com nomes esquisitos.

1.2.4 Atribuição

Já vimos como atribuir dados a uma variável através do operador *:=* . Agora iremos detalhar essa forma e ver outras formas de atribuição.

O código listado em 1.8 nos mostra a atribuição com o operador ‘‘=’’ e com o comando STORE. Conforme já dissemos, prefira o operador *:=*.

Listing 1.8: Outras forma de atribuição

```
1
```

```
2  /*
3  Atribuição
4  */
5  PROCEDURE Main
6
7     x = 1200  // Atribui 1200 ao valor x
8     STORE 100 TO y, k, z // Atribui 100 ao valor y, k e z
9
10    ? x + y + k + z
11
12 RETURN
```

.:Resultado:.

```
      1500
```

A atribuição de variáveis também pode ser feita de forma simultânea, conforme a listagem 1.9.

Listing 1.9: Atribuição simultânea

```
1  /*
2  Atribuição
3  */
4  PROCEDURE Main
5
6     x := y := z := k := 100
7     ? x + y + z + k
8
9  RETURN
```

.:Resultado:.

```
       400
```

Dica 10

Já foi dito que o Harbour é uma linguagem “Case insensitive”, ou seja, ela não faz distinção entre letras maiúsculas e minúsculas. Essa característica se aplica também as variáveis. Os seguintes nomes são equivalentes : **nNota, NNOTA, nnota, NnOta e NNota**. Note que algumas variações da variável **nNota** são difíceis de se ler, portanto você deve sempre nomear as suas variáveis obedecendo aos padrões já enfatizados nesse capítulo.

1.3 Variáveis: declaração e inicialização

Os operadores de atribuição possuem um triplo papel na linguagem Harbour : elas criam variáveis (caso elas não existam), atribuem valores as variáveis e podem mudar o tipo de dado de uma variável (os tipos de dados serão vistos no próximo capítulo).

1. O ato de criar uma variável chama-se *declaração*. Declarar é criar a variável, que é o mesmo que reservar um espaço na memória para ela.
2. O ato de inicializar uma variável chama-se *inicialização*. Inicializar uma variável é o mesmo que dar-lhe um valor.

Dica 11

Nos capítulos seguintes nós iremos nos aprofundar no estudo das variáveis, mas desde já é importante que você adquira bons hábitos de programação. Por isso iremos trabalhar com uma instrução chamada de *LOCAL*. Você não precisa se preocupar em entender o que isso significa, mas iremos nos acostumar a declarar as variáveis como *LOCAL* e **obrigatoriamente** no início do bloco de código, assim como feito no código anterior (Listagem 1.9).
Nós deixamos claro, no início do livro, que não iremos usar conceitos sem a devida explicação, mas como esse conceito é importante demais resolvemos abrir essa pequena exceção[a]. Nós digitaremos muitos códigos até o final do livro, e em todos usaremos essa palavra ainda não explicada para declarar as nossas variáveis.
Note também que a palavra reservada *LOCAL* não recebe indentação, pois nós a consideramos parte do início do bloco e também que nós colocamos uma linha em branco após a sua declaração para destacá-las do restante do código. Nós também colocamos um espaço em branco após a vírgula que separa os nomes das variáveis, pois isso ajuda na visualização do código.
A existência de linhas e espaços em branco não deixa o seu programa "maior" e nem consome memória porque o compilador simplesmente os ignora. Assim, você deve usar os espaços e as linhas em branco para dividir o seu programa em "mini-pedaços" facilmente visualizáveis.
Podemos sintetizar o que foi dito através da seguinte "equação" :

```
Variáveis agrupadas de acordo com o comentário +
Indentação no início do bloco +
Espaço em branco após as vírgulas +
Linhas em branco dividindo os ‘‘mini-pedaços’’  =
--------------------------------------------------
Código mais fácil de se entender.
```

[a] Você só poderá entender completamente esse conceito no final do livro

Dica 12

Procure adquirir o bom hábito de **não** declarar as suas variáveis através de um operador. Use a instrução *LOCAL* para isso.
Você pode, preferencialmente, fazer conforme o exemplo abaixo (declaro e inicializo ao mesmo tempo) :

```
1 PROCEDURE Main
2 LOCAL nValor := 12
3
4     ? nValor
5     // O restante das instruções ...
6
7 RETURN
```

se não souber o valor inicial, então faça conforme o exemplo a seguir (declaro e só depois inicializo) :

```
1 PROCEDURE Main
2 LOCAL nValor
3
4     nValor := 12
5     ? nValor
6     // O restante das instruções ...
7
8 RETURN
```

mas nunca faça como o exemplo abaixo (sem a instrução LOCAL)

```
1 PROCEDURE Main
2
3     nValor := 12
4     ? nValor
5     // O restante das instruções ...
6
7 RETURN
```

DICA IMPORTANTE!
Você pode designar o próprio Harbour para monitorar essa prática em tempo de compilação. Isso é muito bom pois evita que você possa declarar uma variável de uma forma não recomendada. Como fazer isso ?
É simples. Lembra do arquivo de configuração de opções de compilação do Harbour chamado de hbmk.hbm ? Ele foi citado no início do livro durante o aprendizado do processo de compilação (capítulo 2). Você pode acrescentar nesse arquivo a opção -w3.

Essa opção eleva para três (o nível máximo) de "warnings" (avisos de cuidado) durante o processo de compilação. No nosso diretório de prática nós temos um arquivo hbmk.hbm já com a opção -w3, por isso se você compilar o terceiro exemplo dessa dica, então ele irá exibir um aviso, mas **não deixará de gerar o programa**.
As mensagens geradas durante a compilação do nosso exemplo acima são :

```
Warning W0001  Ambiguous reference 'NVALOR'
```

"Ambiguous reference" (Referência ambigua) quer dizer : "eu não sei exatamente o que você quer dizer com essa atribuição pois a variável não foi inicializada de forma segura." Voltaremos a esse assunto com detalhes no final do livro. Por enquanto, **certifique-se de ter um arquivo hbmk.hbm na sua pasta onde você compila o seu sistema e que dentro desse arquivo esteja definido o nível máximo de alerta (-w3)**.

1.4 Recebendo dados do usuário

No final desse capítulo iremos treinar algumas rotinas básicas com as variáveis que aprendemos, mas boa parte dessas rotinas precisam que você saiba receber dados digitados pelo usuário. O Harbour possui várias maneiras de receber dados digitados pelo usuário, como ainda estamos iniciando iremos aprender uma maneira simples de receber dados numéricos: o comando *INPUT*[4]. O seu uso está ilustrado no código 1.10.

Listing 1.10: Recebendo dados externos digitados pelo usuário

```
1
2  /*
3  Uso do comando INPUT
4  */
5  PROCEDURE Main
6  LOCAL nVal1 := 0 // Declara a variável parcela de pagamento
7  LOCAL nVal2 := 0 // Declara a variável parcela de pagamento
8
9     INPUT "Informe o primeiro valor : " TO nVal1
10    INPUT "Informa o segundo valor : " TO nVal2
11
12    ? "A soma dos dois valores é : ", nVal1 + nVal2
13
14
15 RETURN
```

[4]O comando INPUT recebe também dados caracteres, mas aconselhamos o seu uso para receber apenas dados numéricos.

Prática número 1

Abra o arquivo basico.prg na pasta “pratica” e salve-o com o nome pratica_input.prg. Feito isso digite o conteúdo acima.

Quando for executar o programa, após digitar o valor sempre tecle *ENTER*.

.:Resultado:.

```
Informe o primeiro valor : 15
Informa o segundo valor : 30
A soma dos dois valores é :          45
```

Dica 13

Note que, na listagem 1.10 as variáveis foram declaradas em uma linha separada, enquanto que em outras listagens elas foram declaradas em uma única linha. Não existe uma regra fixa com relação a isso, mas aconselhamos você a organizar as variáveis de acordo com o comentário (*//*) que provavelmente você fará após a declaração. Por exemplo, se existem duas variáveis que representam coisas que podem ser comentadas em conjunto, então elas devem ser declaradas em apenas uma linha. A listagem 1.10 ficaria mais clara se as duas linhas de declaração fossem aglutinadas em apenas uma linha. Isso porque as duas variáveis (*nVal1* e *nVal2*) possuem uma forte relação entre si. O ideal seria :

```
LOCAL nVal1 := 0, nVal2 := 0 // Parcelas do pagamento.
```

ou ainda recorrendo a atribuição simultânea de valor :

```
LOCAL nVal1 :=  nVal2 := 0 // Parcelas do pagamento.
```

Note também que nós devemos atribuir um valor as variáveis para informar que elas são números (no caso do exemplo acima, o zero informa que as variáveis devem ser tratadas como numéricas).

Se os dados digitados forem do tipo caractere você deve usar o comando *ACCEPT*. (Veja um exemplo na listagem 1.11). Observe que o *ACCEPT* funciona da mesma forma que o *INPUT*.

Listing 1.11: Recebendo dados externos digitados pelo usuário (Tipo caractere)

```
1
2 /*
3 Uso do comando ACCEPT
```

```
4  */
5  PROCEDURE Main
6  LOCAL cNome // Seu nome
7
8     /* Pede e exibe o nome do usuário */
9     ACCEPT "Informe o seu nome : " TO cNome
10    ? "O seu nome é : ", cNome
11
12 RETURN
```

Prática número 2

Abra o arquivo basico.prg na pasta "pratica" e salve-o com o nome pratica_accept.prg. Feito isso digite o conteúdo acima.

Após digitar a *string* tecle *ENTER*.

.:Resultado:.

```
Informe o seu nome : Vlademiro Landim Junior
O seu nome é :  Vlademiro Landim Junior
```

Dica 14

Você pode questionar porque temos um comando para receber do usuário dados numéricos e outro para receber dados caracteres. Esses questionamentos são pertinentes, mas não se preocupe pois eles (*ACCEPT* e *INPUT*) serão usados apenas no início do nosso aprendizado. Existem formas mais eficientes para a entrada de dados, mas o seu aprendizado iria tornar o processo inicial de aprendizado da linguagem mais demorado.
Apenas aprenda o "mantra" abaixo e você não terá problemas :

```
Para receber dados do tipo numérico = use INPUT.
Para receber dados do tipo caractere = use ACCEPT.
```

Se mesmo assim você quer saber o porque dessa diferença continue lendo, senão pode pular o trecho a seguir e ir para a seção de exemplos.
Na primeira metade da década de 1980 o dBase II foi lançado. Naquela época não tínhamos os recursos de hardware e de software que temos hoje, tudo era muito simples. Os primeiros comandos de entrada de dados usados pelo dBase foram o *INPUT* e o *ACCEPT*. O *ACCEPT* serve para receber apenas variáveis do tipo caracter. Você não precisa nem declarar a variável, pois se ela não existir o comando irá criá-la para você. O *INPUT* funciona da mesma forma, mas serve também para receber dados numéricos, caracteres, data e lógico. Porém você deve formatar a entrada de dados convenientemente, e é isso torna o seu uso confuso para outros tipos de dados que não sejam os numéricos.

Exemplos de uso do comando INPUT :

```
Usando com variáveis numéricas :

    nVal := 0
    INPUT "Digite o valor : " TO nVal

Usando com variáveis caracteres :

    cNome := " "
    INPUT "Digite o seu nome : " TO "cNome"
    // Note que  variável cNome deve estar entre parênteses
```

Vidal acrescenta que "o comando *INPUT* deve preferivelmente ser utilizado para a entrada de dados numéricos. Para a entrada de dados caracteres use o comando *ACCEPT*" (VIDAL, 1989, p. 107). Para usar *INPUT* com dados do tipo data utilize a função CTOD() e com dados do tipo lógico apenas use o dado diretamente (.t. ou .f.)[a].

[a] Veremos os dados lógico e data mais adiante.

1.5 Exemplos

A seguir veremos alguns exemplos com os conceitos aprendidos. Fique atento pois nós usaremos os sinais referentes as quatro operações com números.

Dica 15

Antes de prosseguirmos note que, nas listagens dos códigos, nós acrescentamos um cabeçalho de comentários conforme abaixo :

```
/*
Um pequeno texto com a descrição do que a rotina faz.
Entrada : Dados de entrada.
Saída : Dados de saída.
*/
```

Acostume-se a documentar o seu código dessa forma: uma descrição breve, os dados de entrada e os dados de saída.

1.5.1 Realizando as quatro operações

O exemplo da listagem 1.14 gera um pequeno programa que executa as quatro operações com dois números que o usuário deve digitar. Note que o sinal de multiplicação é um asterisco (“*”) e o sinal de divisão é uma barra utilizada na escrita de datas (“/”) [5].

Listing 1.12: As quatro operações

```
/*
As quatro operações
Entrada : dois números
Saída : As quatro operações realizadas com esses dois números
*/
PROCEDURE Main
LOCAL nValor1, nValor2 // Valores a serem calculados

   // Recebendo os dados
   ? "Introduza dois números para que eu realize as quatro oper.: "
   INPUT "Introduza o primeiro valor : " TO nValor1
   INPUT "Introduza o segundo valor : " TO nValor2

   // Calculando e exibindo
   ? "Soma.............. : " , nValor1 + nValor2
   ? "Subtraçäo........ : " , nValor1 - nValor2
   ? "Multiplicaçäo.... : " , nValor1 * nValor2
   ? "Divisäo.......... : " , nValor1 / nValor2

RETURN
```

Prática número 3

Vá para a pasta “pratica” e abra o arquivo pratica_quatro.prg e complete o que falta. O resultado deve se parecer com o quadro abaixo.

.:Resultado:.

```
Introduza dois números para que eu realize as quatro oper.:
Introduza o primeiro valor : 10
Introduza o segundo valor : 20
Adição.......... :          30
Subtração........ :         -10
Multiplicação.... :         200
Divisão.......... :           0.50
```

[5] O sinal de uma operação entre variáveis recebe o nome de “operador”. Veremos mais sobre os operadores nos capítulos seguintes.

1.5.2 Calculando o antecessor e o sucessor de um número

O exemplo da listagem 1.13 gera um pequeno programa que descobre qual é o antecessor e o sucessor de um número qualquer inserido pelo usuário.

Listing 1.13: Descobrindo o antecessor e o sucessor

```
1
2  /*
3  Descobre o  antecessor e o sucessor
4  */
5  PROCEDURE Main
6  LOCAL nValor // Número a ser inserido
7
8     // Recebendo os dados
9     ? ""
10    ? "**** Descobrindo o antecessor e o sucessor ****"
11    ? ""
12    INPUT "Introduza o número : " TO nValor
13
14    // Calculando e exibindo
15    ? "Antecessor........ : " , nValor - 1
16    ? "Sucessor.......... : " , nValor + 1
17
18 RETURN
```

Prática número 4

Abra o arquivo basico.prg na pasta “pratica” e salve-o com o nome pratica_antsuc.prg. Feito isso digite o conteúdo acima.

.:Resultado:.

```
**** Descobrindo o antecessor e o sucessor ****

Introduza o número : 10
Antecessor........ :           9
Sucessor.......... :          11
```

Prática número 5

Faça um programa que calcule quantos números existem entre dois números pré-determinados, inclusive esses números. Abra o arquivo pratica_var.prg e complete os itens necessários para resolução desse problema.

Listing 1.14: Calculando quantos números existem entre dois números

```
1
```

```
2  /*
3  Descrição: Calcula quantos números existem entre dois
       intervalos
4                (incluídos os números extremos)
5  Entrada: Limite inferior (número inicial) e limite superior
       (número final)
6  Saída: Quantidade de número (incluídos os extremos)
7  */
8  #define UNIDADE_COMPLEMENTAR 1 // Deve ser adicionada ao
       resultado final
9  PROCEDURE Main
10 LOCAL nIni, nFim // Limite inferior e superior
11 LOCAL nQtd // Quantidade
12
13    ? "Informa quantos números existem entre dois intervalos"
14    ? "(Incluídos os números extremos)"
15    INPUT "Informe o número inicial : " TO nIni
16    INPUT "Informe o número final : " TO nFim
17    nQtd := nFim - nIni + UNIDADE_COMPLEMENTAR
18    ? "Entre os números " , nIni, " e " , nFim, " existem ",
          nQtd, " números"
19
20 RETURN
```

.:Resultado:.

```
Informa quantos números existem entre dois intervalos
(Incluídos os números extremos)
Informe o número inicial : 5
Informe o número final : 10
Entre os números           5  e          10  existem
             6  números
```

1.6 Exercícios de fixação

As respostas encontram-se no apêndice H.

1. Escreva um programa que declare 3 variáveis e atribua a elas valores numéricos através do comando INPUT; depois mais 3 variáveis e atribua a elas valores caracteres através do comando ACCEPT; finalmente imprima na tela os valores.

2. (HORSTMAN, 2005, p. 47) Escreva um programa que exibe a mensagem “Oi, meu nome é Hal!”. Então, em uma nova linha, o programa deve imprimir a mensagem “O que você

gostaria que eu fizesse ?". Então é a vez do usuário digitar uma entrada. [...] Finalmente, o programa deve ignorar a entrada do usuário e imprimir uma mensagem "Sinto muito, eu não posso fazer isto.". Aqui está uma execução típica :

.:Resultado:.

```
Oi, meu nome é Hal!
O que você gostaria que eu fizesse ?
A limpeza do meu quarto.
Sinto muito, eu não posso fazer isto.
```

3. (HORSTMAN, 2005, p. 47) Escreva um programa que imprima uma mensagem "Oi, meu nome é Hal!". Então, em uma nova linha, o programa deve imprimir a mensagem "Qual o seu nome ?" [...] Finalmente, o programa deve imprimir a mensagem "Oi, *nome do usuário*. Prazer em conhecê-lo!" Aqui está uma execução típica :

.:Resultado:.

```
Oi, meu nome é Hal!
Qual é o seu nome ?
Dave
Oi, Dave. Prazer em conhecê-lo.
```

4. Escreva um programa que receba quatro números e exiba a soma desses números.

5. Escreva um programa que receba três notas e exiba a média dessas notas.

6. Escreva um programa que receba três números, três pesos e mostre a média ponderada desses números.

7. Escreva um programa que receba o salário de um funcionário, receba o percentual de aumento (por exemplo 15 se for 15%) e exiba o novo salário do funcionário.

8. Modifique o programa anterior para exibir também o valor do aumento que o funcionário recebeu.

9. Modifique o programa anterior para receber também o percentual do imposto a ser descontado do salário novo do funcionário, e quando for exibir os dados (salário novo e valor do aumento), mostre também o valor do imposto que foi descontado.

10. Escreva um programa que receba um valor a ser investido e o valor da taxa de juros. O programa deve calcular e mostrar o rendimento e o valor total depois do rendimento.

11. Calcule a área de um triangulo. O usuário deve informar a base e a altura e o programa deve retornar a área.

    Dica : $nArea = \frac{nBase * nAltura}{2}$

12. Escreva um programa que receba o ano do nascimento do usuário e retorne a sua idade e quantos anos essa pessoa terá em 2045.

    Dica : YEAR(DATE()) é uma combinação de duas funções que retorna o ano corrente.

13. Escreva um programa que receba uma medida em pés e converta essa medida para polegadas, jardas e milhas.

    Dicas

    - 1 pé = 12 polegadas
    - 1 jarda = 3 pés
    - 1 milha = 1,76 jardas

14. Escreva um programa que receba dois números (nBase e nExpoente) e mostre o valor da potência do primeiro elevado ao segundo.

15. Escreva um programa que receba do usuário o nome e a idade dele . Depois de receber esses dados o programa deve exibir o nome e a idade do usuário convertida em meses. Use o exemplo abaixo como modelo :

    .:Resultado:.

```
Digite o seu nome : Paulo
Digite quantos anos você tem : 20

Seu nome é Paulo e você tem aproximadamente 240 meses de
   vida.
```

    Dica : Lembre-se que o sinal de multiplicação é um “*”.

16. Escreva um programa que receba do usuário um valor em horas e exiba esse valor convertido em segundos. Conforme o exemplo abaixo :

    .:Resultado:.

```
Digite um valor em horas : 3
3    horas tem       10800      segundos
```

17. Faça um programa que informe o consumo em quilômetros por litro de um veículo. O usuário deve entrar com os seguintes dados : o valor da quilometragem inicial, o valor da quilometragem final e a quantidade de combustível consumida.

1.7 Desafios

1.7.1 Identifique o erro de compilação no programa abaixo.

Listing 1.15: Erro 1

```
/*
Onde está o erro ?
*/
PROCEDURE Main
LOCAL x, y, x // Número a ser inserido

   x := 5
   y := 10
   x := 20

RETURN
```

1.7.2 Identifique o erro de lógica no programa abaixo.

Um erro de lógica é quando o programa consegue ser compilado mas ele não funciona como o esperado. Esse tipo de erro é muito perigoso pois ele não impede que o programa seja gerado. Dessa forma, os erros de lógica só podem ser descoberto durante a seção de testes ou (pior ainda) pelo cliente durante a execução. Um erro de lógica quase sempre é chamado de *bug*.

Listing 1.16: Erro de lógica

```
/*
Onde está o erro ?
*/
PROCEDURE Main
LOCAL x, y // Número a ser inserido

   x := 5
   y := 10
   ACCEPT "Informe o primeiro número : " TO x
   ACCEPT "Informe o segundo número : " TO y
   ? "A soma  é ", x + y

RETURN
```

1.7.3 Valor total das moedas

(HORSTMAN, 2005, p. 50) Eu tenho 8 moedas de 1 centavo, 4 de 10 centavos e 3 de 25 centavos em minha carteira. Qual o valor total de moedas ? Faça um programa que calcule o valor total para qualquer quantidade de moedas informadas.

Siga o modelo :

.:Resultado:.

```
Informe quantas moedas você tem :

Quantidade de moedas de 1 centavo :  10
Quantidade de moedas de 10 centavos : 3
Quantidade de moedas de 25 centavos : 4

Você tem     1.40 em moedas.
```

1.7.4 O comerciante maluco

Adaptado de (FORBELLONE; EBERSPACHER, 2005, p. 62). Um dado comerciante maluco cobra 10% de acréscimo para cada prestação em atraso e depois dá um desconto de 10% sobre esse valor. Faça um programa que solicite o valor da prestação em atraso e apresente o valor final a pagar, assim como o prejuízo do comerciante na operação.

Referências Bibliográficas

ASCENCIO, A. F. G.; CAMPOS, E. A. V. de. **Fundamentos da programação de computadores: algoritmos, PASCAL, C/C++ (padrão ANSI) e JAVA**. Pearson, São Paulo, 2014.

DEITEL, H. M.; DEITEL, P. J. **C++ Como programar**. Bookman, Porto Alegre, 2001.

FORBELLONE, A. L.; EBERSPACHER, H. F. **Lógica de programação: a construção de algoritmos e estrutura de dados**. Prentice Hall, São Paulo, 2005.

HORSTMAN, C. **Conceitos de computação com o essencial de C++ .** Bookman, Porto Alegre, 2005.

KERNIGHAN, B. W.; PIKE, R. **A prática da programação**. Campus, Rio de Janeiro, 2000.

RAMALHO, J. A. A. **Clipper avançado**. McGraw-Hill, São paulo, 1991.

SPENCE, R. **Clipper 5.2**. Makron, Rio de Janeiro, 1994.

STROUSTRUP, B. **Princípios e práticas de programação com C++ .** Bookman, Porto Alegre, 2012.

VIDAL, A. G. d. R. **Clipper**. LTC - Livros Técnicos e Científicos, Rio de Janeiro, 1989.

A

Exercícios : constantes e variáveis

Somos o que repetidamente fazemos. A excelência, portanto, não é um feito, mas um hábito.

Aristóteles

Objetivos do capítulo

- Praticar o que foi visto até agora.

A.1 Resposta aos exercícios de fixação sobre variáveis - Capítulo 4

1.Escreva um programa que declare 3 variáveis e atribua a elas valores numéricos através do comando INPUT; depois mais 3 variáveis e atribua a elas valores caracteres através do comando ACCEPT; finalmente imprima na tela os valores.

Listing A.1: Resposta

```
1  /*
2  Entrada : 3 variáveis numéricas e 3 variáveis caracteres
3  Saída : As variáveis mostradas
4  */
5  PROCEDURE Main
6  LOCAL nA, nB, nC // Variáveis numéricas
7  LOCAL cA, cB, cC // Variáveis caracteres
8
9     INPUT "Insira o primeiro valor numérico : " TO nA
10    INPUT "Insira o segundo valor numérico : " TO nB
11    INPUT "Insira o terceiro valor numérico : " TO nC
12
13    ACCEPT "Insira o primeiro valor caractere : " TO cA
14    ACCEPT "Insira o segundo valor caractere : " TO cB
15    ACCEPT "Insira o terceiro valor caractere : " TO cC
16
17    ? "Os valores numéricos : " , nA, nB, nC
18    ? "Os valores caracteres : " , cA, cB, cC
19
20 RETURN
```

2.(HORSTMAN, 2005, p. 47) Escreva um programa que exibe a mensagem “Oi, meu nome é Hal!”. Então, em uma nova linha, o programa deve imprimir a mensagem “O que você gostaria que eu fizesse ?”. Então é a vez do usuário digitar uma entrada. [...] Finalmente, o programa deve ignorar a entrada do usuário e imprimir uma mensagem “Sinto muito, eu não posso fazer isto.”. Aqui está uma execução típica :

.:Resultado:.

```
Oi, meu nome é Hal!
O que você gostaria que eu fizesse ?
A limpeza do meu quarto.
Sinto muito, eu não posso fazer isto.
```

Comentário : nesse exemplo você deve ter recebido o aviso de warning :

.:Resultado:.

```
r0402.prg(16) Warning W0032  Variable 'CENTRADA' is assigned
   but not used in function 'MAIN(11)'
```

Esse aviso, nós já vimos, é um “warning”. Ele não impedirá o seu programa de ser gerado, mas avisa que algo possivelmente está errado. Você consegue identificar o “possível erro” ? Pense um pouco a respeito. A resposta está nessa nota de rodapé[1].

Listing A.2: Resposta

```
1  /*
2  Entrada : 3 variáveis numéricas e 3 variáveis caracteres
3  Saída : As variáveis mostradas
4  */
5  PROCEDURE Main
6  LOCAL cEntrada // Valor a ser digitado
7
8     ? "Oi, meu nome é Hal!"
9     ? "O que você gostaria que eu fizesse ?"
10    ?
11    ACCEPT TO cEntrada
12
13    ? "Sinto muito, eu näo posso fazer isso"
14
15 RETURN
```

3.(HORSTMAN, 2005, p. 47) Escreva um programa que imprima uma mensagem “Oi, meu nome é Hal!”. Então, em uma nova linha, o programa deve imprimir a mensagem “Qual o seu nome ?” [...] Finalmente, o programa deve imprimir a mensagem “Oi, *nome do usuário*. Prazer em conhecê-lo!” Aqui está uma execução típica :

.:Resultado:.

```
Oi, meu nome é Hal!
Qual é o seu nome ?
Dave
Oi, Dave. Prazer em conhecê-lo.
```

Listing A.3: Resposta

```
1  /*
2  Entrada : 3 variáveis numéricas e 3 variáveis caracteres
3  Saída : As variáveis mostradas
4  */
5  PROCEDURE Main
6  LOCAL cEntrada  // Valor a ser digitado
7
```

[1] O “possível erro” é o seguinte : você declarou uma variável (cEntrada), depois a inicializou com o comando ACCEPT, mas não fez nada com ela. Isso é muito estranho. Por que alguém iria criar uma variável e não fazer nada com ela ? Por isso o compilador emitiu um aviso informando que ela (a variável) recebeu (assigned) um valor mas ele não foi usado. Mas, como o nosso exercício não previa uso para a variável cEntrada, podemos ignorar esse aviso.

```
8     ? "Oi, meu nome é Hal!"
9     ? "Qual é o seu nome ?"
10    ? // Pula uma linha (fica melhor para o usuário)
11    ACCEPT TO cEntrada
12
13    ? "Oi," , cEntrada, "prazer em conhecê-lo."
14
15  RETURN
```

4.Escreva um programa que receba quatro números e exiba a soma desses números.

Listing A.4: Resposta

```
1   /*
2   Descr. : Programa que recebe quatro números,
3            calcula e mostra a soma desses números.
4   Entrada: quatro números.
5   Saída  : a soma desses números exibida.
6   */
7   PROCEDURE Main
8   LOCAL n1, n2, n3, n4 // Parcelas
9   LOCAL nSoma // Soma
10
11     INPUT "Entre com valor 1 : " TO n1
12     INPUT "Entre com valor 2 : " TO n2
13     INPUT "Entre com valor 3 : " TO n3
14     INPUT "Entre com valor 4 : " TO n4
15
16     nSoma := n1 + n2 + n3 + n4
17     ? "A soma desses números é " , nSoma
18
19  RETURN
```

5.Escreva um programa que receba três notas e exiba a média dessas notas.

Listing A.5: Resposta

```
1   /*
2   Descr. : Programa que recebe três notas,
3            calcula e mostra a média desses números.
4   Entrada: três números.
5   Saída  : a média desses números exibida.
6   */
7   PROCEDURE Main
8   LOCAL n1, n2, n3 // Valores
9   LOCAL nMedia // Média dos valores
10
11     INPUT "Entre com valor 1 : " TO n1
```

```
12    INPUT "Entre com valor 2 : " TO n2
13    INPUT "Entre com valor 3 : " TO n3
14
15    nMedia := (( n1 + n2 + n3 ) / 3 )
16    ? "A média desses números é " , nMedia
17
18 RETURN
```

6.Escreva um programa que receba três números, três pesos e mostre a média ponderada desses números.

Listing A.6: Resposta

```
1  /*
2  Descr. : Programa que recebe três notas e seus pesos,
3           calcula e mostra a média ponderada desses números.
4  Entrada: três números e três pesos.
5  Saída  : a média ponderada desses números exibida.
6  */
7  PROCEDURE Main
8  LOCAL n1, n2, n3 // Valores
9  LOCAL p1, p2, p3 // Pesos
10 LOCAL nMedia // Média
11
12    INPUT "Entre com valor 1 : " TO n1
13    INPUT "Entre com peso 1 : " TO p1
14    INPUT "Entre com valor 2 : " TO n2
15    INPUT "Entre com peso 2 : " TO p2
16    INPUT "Entre com valor 3 : " TO n3
17    INPUT "Entre com peso 3 : " TO p3
18
19    nMedia := ( (( n1 * p1 ) + ( n2 * p2 ) + ( n3 * p3 ) ) / (
         p1 + p2 + p3 ) )
20    ? "A média ponderada desses números é " , nMedia
21
22 RETURN
```

7.Escreva um programa que receba o salário de um funcionário, receba o percentual de aumento (por exemplo 15 se for 15%) e exiba o novo salário do funcionário.

Listing A.7: Resposta

```
1  /*
2  Descr. : Programa que recebe o salário do funcionário, o
     percentual de reajuste
3           calcule e mostre o novo salário.
4  Entrada: Salário.
5  Saída  : O salário reajustado.
```

```
6  */
7  PROCEDURE Main
8  LOCAL nSalario
9  LOCAL nAumento
10 LOCAL nSalarioNovo
11
12    INPUT "Entre com valor do salário : " TO nSalario
13    INPUT "Entre com o percentual de aumento (Ex: 15 se for 15%)
          : " TO nAumento
14
15    nSalarioNovo := nSalario + ( nSalario * nAumento ) / 100
16    ? "O novo salário é " , nSalarioNovo
17
18 RETURN
```

8.Modifique o programa anterior para exibir também o valor do aumento que o funcionário recebeu.

Listing A.8: Resposta

```
1  /*
2  Descr. : Programa que recebe o salário do funcionário,
3           o percentual de aumento,
4           calcule e mostre o novo salário, e o valor do aumento.
5  Entrada: Salário e percentual de aumento.
6  Saída  : O novo salário e o valor do aumento.
7  */
8  PROCEDURE Main
9  LOCAL nSalario
10 LOCAL nPercAumento
11 LOCAL nValorAumento
12 LOCAL nSalarioNovo
13
14    INPUT "Entre com valor do salário : " TO nSalario
15    INPUT "Entre com o percentual de aumento (Ex: 15) : " TO
          nPercAumento
16
17    nValorAumento := ( nSalario * nPercAumento ) / 100
18    nSalarioNovo := nSalario + nValorAumento
19    ? "O novo salário é " , nSalarioNovo
20    ? "O aumento foi de " , nValorAumento
21
22 RETURN
```

9.Modifique o programa anterior para receber também o percentual do imposto a ser descontado do salário novo do funcionário, e quando for exibir os dados (salário novo e valor do aumento), mostre também o valor do imposto que foi descontado.

Listing A.9: Resposta

```
1  /*
2  Descr. : Programa que recebe o salário do funcionário,
3           o percentual de aumento, o percentual do imposto
4           calcule e mostre o novo salário, o percentual do
               imposto
5           e o valor do aumento.
6  Entrada: Salário, percentual do imposto e percentual de aumento.
7  Saída  : O novo salário, o valor do aumento e o valor do
      imposto.
8  */
9  PROCEDURE Main
10 LOCAL nSalario
11 LOCAL nPercAumento
12 LOCAL nPercImposto
13 LOCAL nValorAumento
14 LOCAL nValorDoImposto
15 LOCAL nSalarioNovo
16
17    INPUT "Entre com valor do salário : " TO nSalario
18    INPUT "Entre com o percentual de aumento (Ex: 25) : " TO
         nPercAumento
19    INPUT "Entre com o percentual de imposto (Ex: 10) : " TO
         nPercImposto
20
21    nValorAumento := ( nSalario * nPercAumento ) / 100
22    nSalarioNovo := nSalario + nValorAumento
23    nValorDoImposto := ( nSalarioNovo * nPercImposto ) / 100
24    nSalarioNovo := nSalarioNovo - nValorDoImposto
25    ? "O novo salário é " , nSalarioNovo
26    ? "O aumento foi de " , nValorAumento
27    ? "O imposto foi de " , nValorDoImposto
28
29 RETURN
```

10. Escreva um programa que receba um valor a ser investido e o valor da taxa de juros. O programa deve calcular e mostrar o rendimento e o valor total depois do rendimento.

Listing A.10: Resposta

```
1  /*
2  Descr. : Recebe o valor de um depósito e o valor da taxa de
      juros,
3           calcule e mostre o valor do rendimento e o valor total
4           depois do rendimento.
5  Entrada: Depósito e taxa de juros.
6  Saída  : O valor do rendimento e o valor total depois do
```

```
   rendimento.
*/
PROCEDURE Main
LOCAL nDeposito // Valor depositado
LOCAL nTaxaJuros // Taxa de juros
LOCAL nRendimento // Rendimento
LOCAL nValorFinal // Valor final

   INPUT "Entre com valor do depósito : " TO nDeposito
   INPUT "Entre com a taxa de juros (Ex: 25) : " TO nTaxaJuros

   nRendimento := ( nDeposito * nTaxaJuros ) / 100
   nValorFinal := nDeposito + nRendimento
   ? "O valor do rendimento foi " , nRendimento
   ? "O valor total depois do rendimento foi " , nValorFinal

RETURN
```

11. Calcule a área de um triangulo. O usuário deve informar a base e a altura e o programa deve retornar a área.

    Dica : $nArea = \frac{nBase * nAltura}{2}$

Listing A.11: Resposta

```
/*
Descr. : Calcula a área de um triangulo sabendo que
          Area = ( Base * Altura ) / 2
Entrada: Base e altura.
Saída  : O valor da área de um triangulo.
*/
PROCEDURE Main
LOCAL nBase // Base
LOCAL nAltura // Altura
LOCAL nArea // Area

   INPUT "Entre com a base : " TO nBase
   INPUT "Entre com a altura : " TO nAltura

   nArea := ( nBase * nAltura ) / 2
   ? "O valor da área é  " , nArea

RETURN
```

12. Escreva um programa que receba o ano do nascimento do usuário e retorne a sua idade e quantos anos essa pessoa terá em 2045.

    Dica : YEAR(DATE()) é uma combinação de duas funções que retorna o ano corrente.

Listing A.12: Resposta

```
/*
Entrada: ano de nascimento e o ano atual
Saída  : a idade da pessoa, quantos anos ela terá em 2045

Dica : YEAR( DATE() ) retorna o ano corrente.
*/
#define ANO_FUTURO 2045
PROCEDURE Main
LOCAL nAnoNasc, nAno := YEAR( DATE() )
LOCAL nIdade, nIdadeFuturo

   INPUT "Entre com o ano do seu nascimento : " TO nAnoNasc

   nIdade := nAno - nAnoNasc
   nIdadeFuturo := ANO_FUTURO - nAnoNasc

   ? "Sua idade atual é " , nIdade
   ? "Em" , ANO_FUTURO, "você terá " , nIdadeFuturo , "anos"

RETURN
```

13.Escreva um programa que receba uma medida em pés e converta essa medida para polegadas, jardas e milhas.

Dicas

- 1 pé = 12 polegadas
- 1 jarda = 3 pés
- 1 milha = 1,76 jardas

Listing A.13: Resposta

```
/*
Descr. : Recebe uma medida em pés e converte a medida
         em polegada, jarda, milha
Entrada: O número em pés
Saída  : O valor da entrada em pés e converte a medida
         em polegada, jarda, milha


Nota : pé = 12 polegadas
       1 jarda = 3 pés
       1 milha = 1,760 jarda

*/
PROCEDURE Main
```

```
15 LOCAL nPe
16 LOCAL nPolegada, nJarda, nMilha
17
18    INPUT "Entre com a medida em pés : " TO nPe
19
20    nPolegada := nPe * 12
21    nJarda := nPe / 3
22    nMilha := nJarda / 1760
23
24    ? "O valor em polegadas é " , nPolegada
25    ? "O valor em jardas é " , nJarda
26    ? "O valor em milhas é " , nMilha
27
28 RETURN
```

14.Escreva um programa que receba dois números (nBase e nExpoente) e mostre o valor da potência do primeiro elevado ao segundo.

Listing A.14: Resposta

```
1  /*
2  Entrada: Base e expoente
3  Saída  : a potência
4  */
5  PROCEDURE Main
6  LOCAL nBase, nExpoente
7  LOCAL nPotencia
8
9     INPUT "Entre com a base : " TO nBase
10    INPUT "Entre com o expoente : " TO nExpoente
11
12    nPotencia := nBase ^ nExpoente
13
14    ? "O valor da potência é " , nPotencia
15
16 RETURN
```

15.Escreva um programa que receba do usuário o nome e a idade dele . Depois de receber esses dados o programa deve exibir o nome e a idade do usuário convertida em meses. Use o exemplo abaixo como modelo :

.:Resultado:.

```
Digite o seu nome : Paulo
Digite quantos anos você tem : 20
```

```
Seu nome é Paulo e você tem aproximadamente 240 meses de
   vida.
```

Dica : Lembre-se que o sinal de multiplicação é um “*”.

Listing A.15: Resposta

```
1  /*
2  Entrada: Nome e idade
3  Saída  : Meses de vida aproximados
4  */
5  #define MESES_POR_ANO 12
6  PROCEDURE Main
7  LOCAL cNome
8  LOCAL nIdade, nMeses
9
10    ACCEPT "Informe o seu nome : " TO cNome
11    INPUT "Entre com a idade : " TO nIdade
12
13    nMeses := nIdade * MESES_POR_ANO
14
15    ? "O seu nome é " , cNome, " e você tem aproximadamente " ,
         nMeses , " meses de vida"
16
17 RETURN
```

16.Escreva um programa que receba do usuário um valor em horas e exiba esse valor convertido em segundos. Conforme o exemplo abaixo :

.:Resultado:.

```
Digite um valor em horas : 3
3    horas tem       10800      segundos
```

Listing A.16: Resposta

```
1  /*
2  Entrada: Valor em horas
3  Saída  : Valor convertido para segundos
4  */
5  #define SEGUNDOS_POR_HORA 60*60 // 1 hora = 60 minutos * 60
      segundos
6  PROCEDURE Main
7  LOCAL nHora, nSegundos
8
9     INPUT "Entre com o valor em horas : " TO nHora
10
11    nSegundos := nHora * SEGUNDOS_POR_HORA
```

```
12
13    ? nHora, "horas tem" , nSegundos, "segundos."
14
15 RETURN
```

17.Faça um programa que informe o consumo em quilômetros por litro de um veículo. O usuário deve entrar com os seguintes dados : o valor da quilometragem inicial, o valor da quilometragem final e a quantidade de combustível consumida.

Listing A.17: Resposta

```
1  /*
2  Entrada: Quilometragem inicial e quilometragem final, e
       quantidade de combustivel
3  Saída  : Valor do consumo
4  */
5  PROCEDURE Main
6  LOCAL nKmIni, nKmFim // Quilometragem
7  LOCAL nQtd // Quantidade consumida
8  LOCAL nConsumo // Consumo
9
10    INPUT "Entre com o valor da quilometragem inicial : " TO
          nKmIni
11    INPUT "Entre com o valor da quilometragem final : " TO nKmFim
12    INPUT "Entre com a quantidade consumida : " TO nQtd
13
14    nConsumo := ( nKmFim - nKmIni ) / nQtd
15
16    ? "O consumo é de " , nConsumo , " Km/l"
17
18 RETURN
```

A.2 Respostas aos desafios - Capítulo 4

A.2.1 Identifique o erro de compilação no programa abaixo.

Listing A.18: Erro 1

```
1  /*
2  Onde está o erro ?
3  */
4  PROCEDURE Main
5  LOCAL x, y, x // Número a ser inserido
6
7     x := 5
8     y := 10
```

```
9     x := 20
10
11  RETURN
```

Resposta : A variável x foi declarada duas vezes.

A.2.2 Identifique o erro de lógica no programa abaixo.

Um erro de lógica é quando o programa consegue ser compilado mas ele não funciona como o esperado. Esse tipo de erro é muito perigoso pois ele não impede que o programa seja gerado. Dessa forma, os erros de lógica só podem ser descoberto durante a seção de testes ou (pior ainda) pelo cliente durante a execução. Um erro de lógica quase sempre é chamado de *bug*.

Listing A.19: Erro de lógica

```
1  /*
2  Onde está o erro ?
3  */
4  PROCEDURE Main
5  LOCAL x, y // Número a ser inserido
6
7     x := 5
8     y := 10
9     ACCEPT "Informe o primeiro número : " TO x
10    ACCEPT "Informe o segundo número : " TO y
11    ? "A soma  é ", x + y
12
13 RETURN
```

Resposta : O comando ACCEPT foi usado para receber valores numéricos.

A.2.3 Valor total das moedas

(HORSTMAN, 2005, p. 50) Eu tenho 8 moedas de 1 centavo, 4 de 10 centavos e 3 de 25 centavos em minha carteira. Qual o valor total de moedas ? Faça um programa que calcule o valor total para qualquer quantidade de moedas informadas.

Siga o modelo :

.:Resultado:.

```
Informe quantas moedas você tem :

Quantidade de moedas de 1 centavo :  10
Quantidade de moedas de 10 centavos : 3
Quantidade de moedas de 25 centavos : 4
```

```
Você tem     1.40 em moedas.
```

Listing A.20: Resposta

```
/*
Entrada: Quantidade de moedas (25, 10 e 1 centavos)
Saída  : Valor em moedas
*/
#define FATOR_25 0.25  // Fator de conversäo para 25 centavos
#define FATOR_10 0.1 // Fator de conversäo para 10 centavos
#define FATOR_01 0.01 // Fator de conversão para 1 centavo
PROCEDURE Main
LOCAL nQtd1Cent, nQtd10Cent, nQtd25Cent // Quantidades
LOCAL nValor // Valor em moedas

   INPUT "Quantidade de moedas de um centavo : " TO nQtd1Cent
   INPUT "Quantidade de moedas de dez centavos : " TO nQtd10Cent
   INPUT "Quantidade de moedas de vinte e cinco centavos : " TO
      nQtd25Cent

   nValor := ( nQtd25Cent * FATOR_25 ) + ( nQtd10Cent * FATOR_10 )
      + ( nQtd1Cent * FATOR_01 )

   ? "Você tem R$" , nValor, "em moedas"

RETURN
```

A.2.4 O comerciante maluco

Adaptado de (FORBELLONE; EBERSPACHER, 2005, p. 62). Um dado comerciante maluco cobra 10% de acréscimo para cada prestação em atraso e depois dá um desconto de 10% sobre esse valor. Faça um programa que solicite o valor da prestação em atraso e apresente o valor final a pagar, assim como o prejuízo do comerciante na operação.

Listing A.21: Resposta

```
/*
Entrada : O valor da parcela
Saída : O valor final após o acréscimo, o valor final após o
   desconto e o prejuízo.
*/
#define DESCONTO 0.10
#define ACRESCIMO 0.10
PROCEDURE Main
LOCAL nParcela // nValor inicial
LOCAL nValorAcr // nValor final com acréscimo
```

```
10 LOCAL nValorDesc // Valor final após o desconto
11
12    INPUT "Informe o valor da parcela em atraso : " TO nParcela
13
14    nValorAcr := nParcela + ( nParcela * ACRESCIMO )
15    ? "O valor da parcela acrescida de ", ACRESCIMO ,"por cento é "
          , nValorAcr
16    nValorDesc := nValorAcr - ( nValorAcr * DESCONTO )
17    ? "O valor descontado ", DESCONTO, "por cento é " , nValorDesc
18    ? "O prejuízo é de " , nParcela - nValorDesc
19
20 RETURN
```

Nota: Na resposta eu acrescentei, além do valor final a pagar (nValorDesc) e o valor do prejuízo, o valor da parcela acrescida. Isso torna o problema mais claro. Mas se você não mostrou o valor da parcela acrescida (nValorAcr) não tem problema.

A.3 Resposta aos exercícios de fixação sobre variáveis - Capítulo 5

1.Calcule a área de um círculo. O usuário deve informar o valor do raio.

Dica : $nArea = PI * nRaio^2$.

Lembrete : Não esqueça de criar a constante PI (Onde PI = 3.1415) .

Listing A.22: Resposta

```
1  /*
2  Descr. : Calcula a área de um circulo sabendo que
3             Area = PI * Raio ^ 2
4  Entrada: O raio.
5  Saída  : O valor da área de um círculo.
6  */
7  #define PI_VALOR 3.1415
8  PROCEDURE Main
9  LOCAL nRaio  // Raio da circunferência
10 LOCAL nArea // Área
11
12    INPUT "Entre com o raio : " TO nRaio
13
14    nArea := PI_VALOR * nRaio ^ 2
15    ? "O valor da área é  " , nArea
16
17 RETURN
```

2.Escreva um programa que receba um valor numérico e mostre :

- O valor do número ao quadrado.

- O valor do número ao ao cubo.
- O valor da raiz quadrada do número.
- O valor da raiz cúbica do número.

Nota : suponha que o usuário só irá informar números positivos.

Dica : lembre-se que $\sqrt[2]{100} = 100^{\frac{1}{2}}$

Listing A.23: Resposta

```
/*
Descr. : Calcula o número ao quadrado, ao cubo, a raiz quadrada
         e a raiz cúbica do número
Entrada: O número.
Saída  : o número ao quadrado, ao cubo, a raiz quadrada
         e a raiz cúbica do número

Nota : A raiz é o número elevado a 1 / potência
*/
PROCEDURE Main
LOCAL nNumero

   INPUT "Entre com o raio : " TO nNumero

   ? "O quadrado é  " , nNumero ^ 2
   ? "O cubo é  " , nNumero ^ 3
   ? "A raiz quadrada é  " , nNumero ^ ( 1 / 2 )
   ? "A raiz cúbica é  " , nNumero ^ ( 1 / 3 )

RETURN
```

3.Construa um programa para calcular o volume de uma esfera de raio R, em que R é um dado fornecido pelo usuário.

Dica : o volume V de uma esfera é dado por $V = \frac{4*PI*Raio^3}{3}$.

Listing A.24: Resposta

```
/*
Resposta
*/
#define PI 3.14
PROCEDURE Main
LOCAL nRaio

   CLS
   ? "Cálculo do volume de uma esfera"
   INPUT "Informe o valor do raio da esfera " TO nRaio
   ? "O volume da esfera é : " , ( ( 4 * PI * nRaio ) ^ 3 ) / 3
```

```
12
13 RETURN
```

Comentário: note o uso de parênteses para evitar que a fórmula seja aplicada erroneamente.

4.Construa programas para reproduzir as equações abaixo. Considere que o lado esquerdo da equação seja a variável que queremos saber o valor. As variáveis do lado direito devem ser informadas pelo usuário. Siga o exemplo no modelo abaixo :

IMPORTANTE: Estamos pressupondo que o usuário é um ser humano perfeito e que não irá inserir letras nem irá forçar uma divisão por zero.

- $z = \frac{1}{x+y}$

Modelo :

```
1  PROCEDURE Main
2  LOCAL x,y,z
3
4       INPUT "Insira o valor de x : " TO x
5       INPUT "Insira o valor de y : " TO y
6       z = ( 1 / ( x + y ) )
7       ? "O valor de z é : " , z
8
9
10 RETURN
```

.:Resultado:.

```
Insira o valor de x : 10
Insira o valor de y : 20
O valor de z é :           0.03
```

Agora faça o mesmo com as fórmulas seguintes.

- $x = \frac{y-3}{z}$

Listing A.25: Resposta

```
1  /*
2  Resposta
3  */
4  //$ x = \frac{y-3}{z}$
5  PROCEDURE Main
6  LOCAL nY, nZ
7
8     CLS
9
```

```
10     INPUT "Informe o valor de y : " TO nY
11     INPUT "Informe o valor de z : " TO nZ
12     ? ( nY - 3 ) / nZ
13
14 RETURN
```

- $k = \frac{x^3+z/5}{y^2+8}$

Listing A.26: Resposta

```
1  /*
2  Resposta
3  */
4  PROCEDURE Main
5  LOCAL nY, nZ, nX
6
7     CLS
8
9     INPUT "Informe o valor de x : " TO nX
10    INPUT "Informe o valor de y : " TO nY
11    INPUT "Informe o valor de z : " TO nZ
12    ? ( nX^3 + nZ/5 ) / ( nY^2 + 8 )
13
14 RETURN
```

- $y = \frac{x^3 z}{r} - \frac{4n}{h}$

Listing A.27: Resposta

```
1  /*
2  Resposta
3  */
4  PROCEDURE Main
5  LOCAL nH, nZ, nX, nN, nR
6
7     CLS
8
9     INPUT "Informe o valor de x : " TO nX
10    INPUT "Informe o valor de z : " TO nZ
11    INPUT "Informe o valor de r : " TO nR
12    INPUT "Informe o valor de h : " TO nH
13    INPUT "Informe o valor de n : " TO nN
14    ? ( ( (nX ^ 3) * nZ )/ nR ) - ( ( 4*nN ) * nH  )
15
16 RETURN
```

- $t = \frac{y}{2} + \frac{3k^2}{4n}$

Listing A.28: Resposta

```
1  /*
2  Resposta
3  */
4  PROCEDURE Main
5  LOCAL nY, nK, nN
6
7     CLS
8
9     INPUT "Informe o valor de y : " TO nY
10    INPUT "Informe o valor de k : " TO nK
11    INPUT "Informe o valor de n : " TO nN
12    ? ( (nY) / 2 ) + ( ( (3*nK)^2 ) / 4*nN )
13
14 RETURN
```

5.Crie um programa que leia dois valores para as variáveis cA e cB, e efetue a troca dos valores de forma que o valor de cA passe a possuir o valor de cB e o valor de cB passe a possuir o valor de cA. Apresente os valores trocados.

Dica : Utilize uma variável auxiliar cAux.

6.São dadas três variáveis nA,nB e nC. Escreva um programa para trocar seus valores da maneira a seguir:

- nB recebe o valor de nA
- nC recebe o valor de nB
- nA recebe o valor de nC

Dica : Utilize uma variável auxiliar nAux.

7.Leia uma temperatura em graus Fahrenheit e apresentá-la convertida em graus Celsius. A fórmula de conversão é $nC = (nF - 32) * (5/9)$, onde nC é o valor em Celsius e nF é o valor em Fahrenheit.

8.Uma oficina mecânica precisa de um programa que calcule os custos de reparo de um motor a diesel padrão NVA-456. O custo é calculado com a fórmula $nCusto = \frac{nMecanicos}{0.4} * 1000$. O programa deve ler um valor para o número de mecânicos (nMecanicos) e apresentar o valor de custo (nCusto). Os componentes da equação do custo estão listados abaixo :

- nCusto = Preço de custo de reparo do motor.
- nMecanicos = Número de mecânicos envolvidos.
- 0.4 = Constante universal de elasticidade do cabeçote.
- 1000 = Constante para conversão em valor monetário.

9.Escreva um programa que receba um valor (par ou ímpar) e imprima na tela os 3 próximos sequenciais pares ou ímpares.

Por exemplo

.:Resultado:.

```
Informe o número : 4
     6,     8 e    10.
```

outro exemplo (com um valor ímpar)

.:Resultado:.

```
Informe o número : 5
    7,    9 e    11.
```

10.No final do capítulo anterior nós mostramos um exemplo de um programa que calcula a quantidade de números entre dois valores quaisquer (incluindo esses valores). A listagem está reproduzida a seguir :

codigos/pratica_var.prg

```
1
2  /*
3  Descrição: Calcula quantos números existem entre dois intervalos
4               (incluídos os números extremos)
5  Entrada: Limite inferior (número inicial) e limite superior
       (número final)
6  Saída: Quantidade de número (incluídos os extremos)
7  */
8  #define UNIDADE_COMPLEMENTAR 1 // Deve ser adicionada ao
       resultado final
9  PROCEDURE Main
10 LOCAL nIni, nFim // Limite inferior e superior
11 LOCAL nQtd // Quantidade
12
13    ? "Informa quantos números existem entre dois intervalos"
14    ? "(Incluídos os números extremos)"
15    INPUT "Informe o número inicial : " TO nIni
16    INPUT "Informe o número final : " TO nFim
17    nQtd := nFim - nIni + UNIDADE_COMPLEMENTAR
18    ? "Entre os números " , nIni, " e " , nFim, " existem ",
          nQtd, " números"
19
20 RETURN
```

Modifique esse programa (salve-o como excluidos.prg) para que ele passe a calcular a quantidade entre dois valores quaisquer, excluíndo esses valores limites.

A.4 Resposta aos exercícios de fixação sobre condicionais - Capítulo 7

1.(ASCENCIO; CAMPOS, 2014, p. 45) Faça um programa que receba o número de horas trabalhadas e o valor do salário mínimo, calcule e mostre o salário a receber, seguindo essas regras :

(a)a hora trabalhada vale a metade do salário mínimo.

(b)o salário bruto equivale ao número de horas trabalhadas multiplicado pelo valor da hora trabalhada.

(c)o imposto equivale a 3% do salário bruto.

(d)o salário a receber equivale ao salário bruto menos o imposto

Listing A.29: Resposta

```
/*
Resposta
*/
#define PI 3.14
PROCEDURE Main
LOCAL nRaio

   CLS
   ? "Cálculo do volume de uma esfera"
   INPUT "Informe o valor do raio da esfera " TO nRaio
   ? "O volume da esfera é : " , ( ( 4 * PI * nRaio ) ^ 3 ) / 3

RETURN
```

2.(ASCENCIO; CAMPOS, 2014, p. 64) Faça um programa que receba três números e mostre-os em ordem crescente. Suponha que o usuário digitará três números diferentes.

3.Altere o programa anterior para que ele mesmo não permita que o usuário digite pelo menos dois números iguais.

4.(ASCENCIO; CAMPOS, 2014, p. 78) Faça um programa para resolver equações de segundo grau.

(a)O usuário deve informar os valores de a, b e c.

(b)O valor de a deve ser diferente de zero.

(c)O programa deve informar o valor de delta.

(d)O programa deve informar o valor das duas raízes.

5.(FORBELLONE; EBERSPACHER, 2005, p. 46) Faça um programa que leia o ano de nascimento de uma pessoa, calcule e mostre sua idade e verifique se ela já tem idade para votar (16 anos ou mais) e para conseguir a carteira de habilitação (18 anos ou mais).

6.(FORBELLONE; EBERSPACHER, 2005, p. 47) O IMC (Índice de massa corporal) indica a condição de peso de uma pessoa adulta. A fórmula do IMC é igual ao peso dividido pelo quadrado da altura. Elabore um programa que leia o peso e a altura de um adulto e mostre a sua condição de acordo com a tabela abaixo :

(a)abaixo de 18.5 : abaixo do peso

(b)entre 18.5 : peso normal

(c)entre 25 e 30 : acima do peso

(d)acima de 30 : obeso

A.5 Resposta aos exercícios de fixação sobre estruturas de repetição - Capítulo 9

1.(FORBELLONE; EBERSPACHER, 2005, p. 65) Construa um programa que leia um conjunto de dados contendo altura e sexo (“M” para masculino e “F” para feminino) de 10 pessoas e, depois, calcule e escreva :

(a)a maior e a menor altura do grupo

(b)a média de altura das mulheres

(c)o número de homens e a diferença porcentual entre eles e as mulheres

2.(FORBELLONE; EBERSPACHER, 2005, p. 66) Anacleto tem 1,50 metro e cresce 2 centímetros por ano, enquanto Felisberto tem 1,10 metro e cresce 3 centímetros por ano. Construa um programa que calcule e imprima quantos anos serão necessários para que Felisberto seja maior do que Anacleto.

3.Adaptado de (DEITEL; DEITEL, 2001, p. 173) Identifique e corrija os erros em cada um dos comandos (Dica: desenvolva pequenos programinhas de teste com eles para ajudar a descobrir os erros.) :

(a)
```
nProduto := 10
c := 1
DO WHILE ( c <= 5 )
    nProduto *= c
ENDDO
++c
```

(b)
```
x := 1
DO WHILE ( x <= 10 )
    x := x - 1
```

```
ENDDO
```

(c)O código seguinte deve imprimir os valores de 1 a 10.

```
x := 1
DO WHILE ( x < 10 )
    ? x++
ENDDO
```